Perun

Connecting OIDC service - Hands on

Dominik František Bučík

Outline

- OIDC and OAuth2
- Server app
- Browser app
- Own implementation
- Python Social Auth
- Good links

OIDC and OAuth2

What are these?

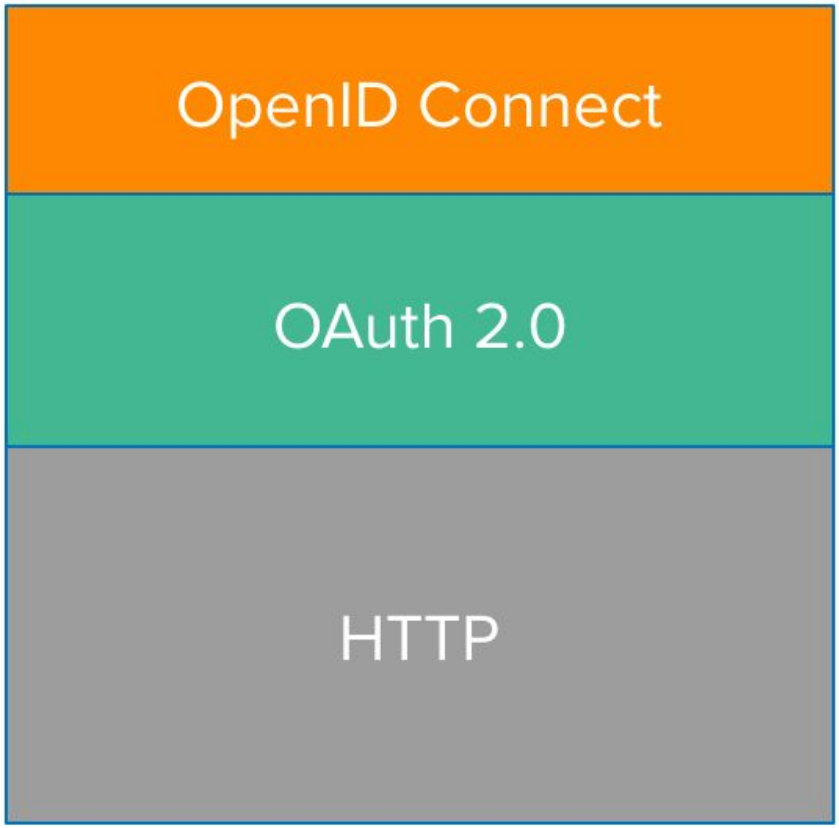
OpenID Connect and OAuth2

- OAuth 2 is **authorization protocol** by which user, owning a resources on resource server, delegates foreign application to use the resources under his/her name

- OpenID Connect (OIDC) is an extension of OAuth2 with **authentication** and API for fetching information about the user

- From the app point of view OIDC is similar to SAML2, but
  - there is no need to exchange metadata between IdP and SP
  - user can choose what data the application receives
  - Applications don’t have to be only from web (it supports mobile, desktop, command-line, SmartTV apps)

Same in picture



OpenID Connect is for **authentication**

OAuth 2.0 is for **authorization**

Picture source

Participating subjects

- **Resource owner** - user (i.e. me)
- **Resource server** - server managing users data, allows processing of the data, permission to process the data is called **scope**
  - Google Calendar API with scopes read, write
- **Client** - application which wants the permission to process users data
  - Business Calendar app
- **Authorization server** - server the user authenticates against, asks users what scopes they want to be released to the client, issues an **access token**
  - https://accounts.google.com/

Client registration

- **Client** has to be registered on the **Authorization server**
- **Client** specifies
  - type of the application (web | user-agent-based | native)
  - list of allowed URLs (where the user can land after login)
  - list of desired scopes (what info to get the about user)
- **Client** receives
  - **client_id**
  - **client_secret**
  - (used for authentication against **Authorization Server**)

Tokens - access and refresh

- **Access token**
  - represents authorization of client by user
  - usually in JWT format (JSON Web Token) - digitally signed JSON
  - **Resource Server** can parse token and verify the signature, or it can ask the **introspection endpoint** to validate it
  - User can invalidate the issued token at anytime
- **Refresh token**
  - Access token has short lifetime
  - Can be exchanged for new access_token by calling token endpoint of authorization server

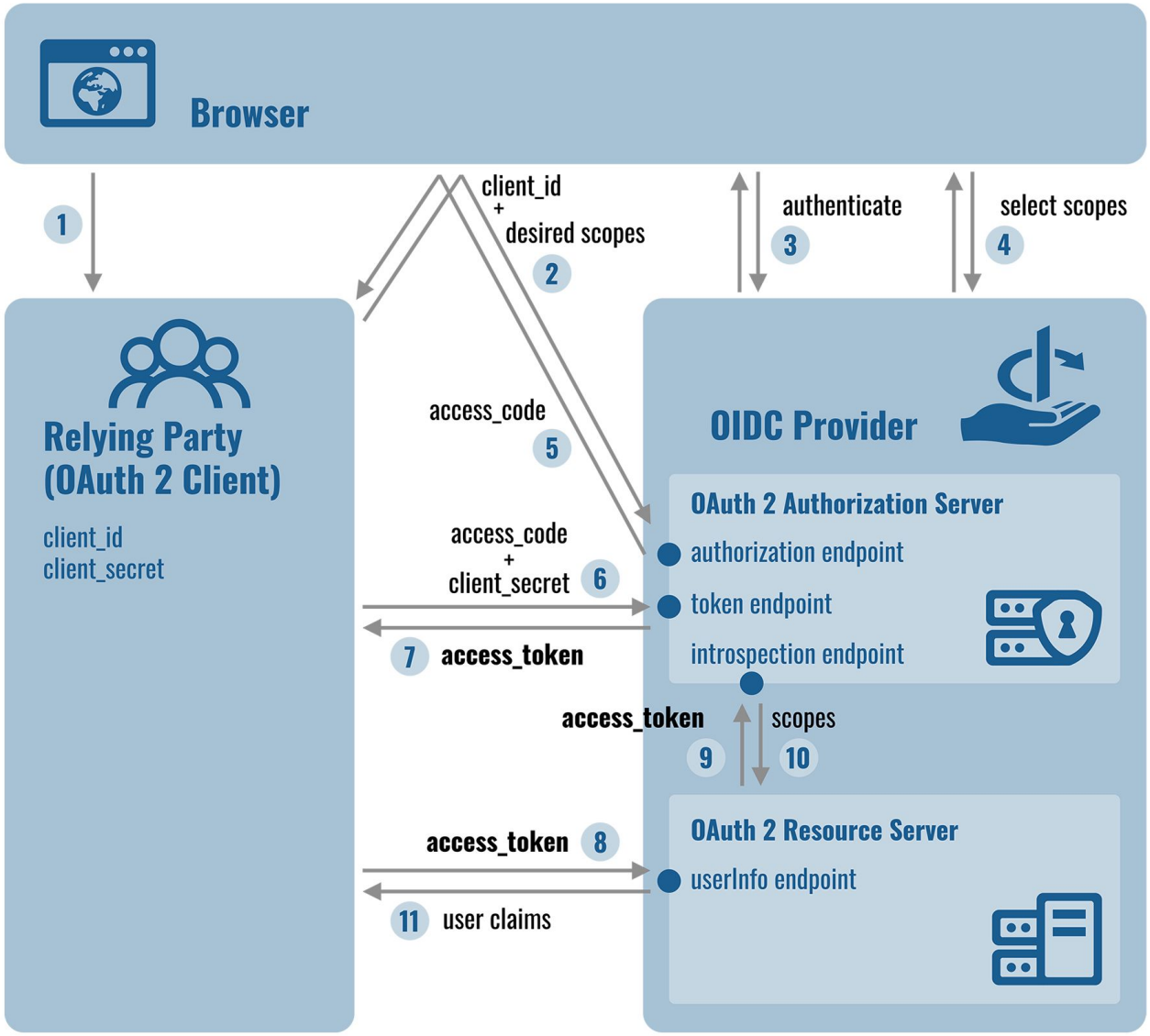
Grant flows

- Approach for obtaining the token based on application type
  - **authorization code grant** - shown in upcoming schema
  - **implicit code grant** - Authorization Server issues access token directly to the client (code is skipped)
  - **resource owner password credentials grant**
  - **device flow grant** - e.g. for SmartTV without keyboard

OpenID Connect

- OAuth2
  - provides way to log in
  - does not specify how to get info about user, every service has different API
- OpenID Connect
  - **userInfo endpoint** - API for fetching the info about user
  - **scopes** - openid, profile, email, address, phone
  - **claims** - sub, name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, updated_at, email, email_verified, address, phone_number, phone_number_verified
  - mapping of scopes to claims
  - **id_token** which can (bud does not have to) contain claims
  - metadata in JSON format available on `/.well-known/openid-configuration`

Data and requests flow



UserInfo response


```
{
  "sub": "3e65bd2aa4c818bd3579023939b546b69e1@einfra",
  "name": "Josef Novák",
  "preferred_username": "pepa",
  "given_name": "Josef",
  "family_name": "Novák",
  "nickname": "Pepan",
  "profile": "https://www.muni.cz/en/people/3988",
  "picture": "https://secure.gravatar.com/avatar/f320c89e39d15da1608c8fc31210b8ca",
  "website": "http://pepovo.wordpress.com/",
  "gender": "male",
  "zoneinfo": "Europe/Prague",
  "locale": "cs-CZ",
  "updated_at": "1508428216",
  "birthdate": "1975-01-01",
  "email": "pepa@gmail.com",
  "email_verified": true,
  "phone_number": "+420 603123456",
  "phone_number_verified": true,
  "address": {
    "street_address": "Severní 1",
    "locality": "Dolní Lhota",
    "postal_code": "111 00",
    "country": "Czech Republic"
  }
}
```

Introspection response


```
{
  "active":true,
  "scope":"address phone openid profile email",
  "expires_at":"2017-10-19T18:50:16+0200",
  "exp": 1508431816,
  "sub": "3e65bd2aa4c818bd3579023939b546b69e1@einfra",
  "user_id":"3e65bd2aa4c818bd3579023939b546b69e1@einfra",
  "client_id":"7652ad4c-4ee6-4ad1-b571-3576574f383e",
  "token_type":"Bearer"
}
```


- specified on https://tools.ietf.org/html/rfc7662#section-2.2

Metadata example

- https://login.cesnet.cz/oidc/.well-known/openid-configuration


```
{
    request_parameter_supported: true,
    claims_parameter_supported: false,
    introspection_endpoint: "https://login.cesnet.cz/oidc/introspect",
  - scopes_supported: [
        "openid",
        "profile",
        "email",
        "address",
        "phone",
        "offline_access",
        "groupNames"
    ],
    issuer: "https://login.cesnet.cz/oidc/",
  + userinfo_encryption_enc_values_supported: […],
  + id_token_encryption_enc_values_supported: […],
    authorization_endpoint: "https://login.cesnet.cz/oidc/authorize",
    service_documentation: "https://login.cesnet.cz/oidc/about",
  + request_object_encryption_enc_values_supported: […],
    device_authorization_endpoint: "https://login.cesnet.cz/oidc/devicecode",
  + userinfo_signing_alg_values_supported: […],
```

OIDC terminology

- „**client**“ is equivalent to „**Relying Party**“
- Authorization Server + Resource Server with userInfo endpoint is called an „**OpenID Provider (OP)**“

- SAML mapping
  - RP == SAML2 SP
  - OP == SAML2 IdP

Server app

No GUI? No problem....

Web service running on server

- Apache module which protects your application
  - Application doesn’t need to be aware of federated login
  - Useful for closed source applications
  - Data are available via environment variables

- Module project home: https://www.mod-auth-openidc.org/

- GitHub repo: https://github.com/zmartzone/mod_auth_openidc

Prerequisites

- Install dependencies and necessary software

```
#> apt install apache2 git vim wget libjansson4 libhiredis0.13 libcurl3
```

- Download latest version of libcjose and mod-auth-openidc

```
#> wget
https://github.com/zmartzone/mod_auth_openidc/releases/download/v2.3.0/libcjose
0_0.5.1-1.stretch.1_amd64.deb
```

```
#> wget
https://github.com/zmartzone/mod_auth_openidc/releases/download/v2.3.3/libapach
e2-mod-auth-openidc_2.3.3-1.stretch.1_amd64.deb
```

- Install packages

```
#> dpkg -i libcjose0_0.5.1-1.stretch.1_amd64.deb
libapache2-mod-auth-openidc_2.3.3-1.stretch.1_amd64.deb
```

Enabling module

- Enable modules in Apache

```
#> a2enmod auth_openidc cgi ssl
```

- Restart Apache

```
#> systemctl restart apache2
```

- Check Apache configuration

```
#> apache2ctl -t
```

```
Syntax OK
```

Configuration template

```
OIDCProviderMetadataURL https://login.cesnet.cz/oidc/.well-known/openid-configuration
OIDCProviderMetadataRefreshInterval 3600
OIDCClientID your_client_id_replace_with_yours
OIDCClientSecret your_client_secret_replace_with_yours
OIDCScope "openid profile"
OIDCRedirectURI /oauth2callback
OIDCCryptoPassphrase randompassword

  <Location /oauth2callback>
    #non-existent location for returning from OIDC server
    AuthType openid-connect
    Require valid-user
  </Location>
  <Location /cgi-bin/>
    #actually protected URLs
    AuthType openid-connect
    Require valid-user
  </Location>
```

Apache configuration

- Open configuration file from Apache

```
#> nano /etc/apache2/sites-available/000-default.conf
```

- Append text from the previous slide at the end of the file before the </VirtualHost> tag
- Enable the configuration file and restart Apache

```
#> a2ensite 000-default.conf
#> service apache2 restart
```

CGI Script - to get some output

- Put following to the /usr/lib/cgi-bin/index.cgi

```
#!/bin/bash

echo -e "Content-type: text/plain\n"

echo -e "Printing environment variables for OIDC\n\n"

set | grep OIDC
```

- Make the file executable for all users:

```
chmod +x /usr/lib/cgi-bin/index.cgi
```

Log in!

- visit the protected URL

```
https://[machine_name]/cgi-bin/
```

```
Printing environment variables for OIDC


OIDC_CLAIM_aud=22602425-bf20-4e36-b096-dd38417918c0
OIDC_CLAIM_auth_time=1524500299
OIDC_CLAIM_exp=1524501330
OIDC_CLAIM_iat=1524500730
OIDC_CLAIM_iss=https://login.cesnet.cz/oidc/
OIDC_CLAIM_jti=8dd2b3e5-b173-47e6-8b15-3e01839f4ae8
OIDC_CLAIM_kid=rsa1
OIDC_CLAIM_nonce=l3lwZIKhzQY7Xyml4socbAWLU11bscRW2OC5C721PVk
OIDC_CLAIM_sub=a78efcaa0459f492590tefa3cece12875a0a3504@einfra
```

Browser app

JavaScript geeks stay focused

JS Library

- Certified OIDC library is **oidc-client-js**
- https://github.com/IdentityModel/oidc-client-js/
- Runs in browser, cannot protect client_secret -> **implicit grant flow**
- Registration procedure is the same , only in „**Grant Types**“ choose „**implicit**“ and you don’t need Client Secret
- If the client uses Resource Server other than userInfo you have to deal with CORS headers

Implement callback

```
#> vim /var/www/html/callback.html
```

- Copy content from the https://controlc.com/54822ab8
  - Password to view: perunProxyIdp

Index page

```
#> nano /var/www/html/index.html
```

- Copy content from the https://controlc.com/a4351859
  - Password to view: perunProxyIdp

JS File with App logic

```
#> nano /var/www/html/app.js
```

- Copy content from the https://controlc.com/ad914313
  - Password to view: perunProxyIdp
  - Fill the configuration variable according to your client

JS Library

- Clone the repository to your application folder using git

```
#> git clone https://github.com/IdentityModel/oidc-client-js.git
/var/www/html/oidc-client-js-1.4.1
#> cd /var/www/html/oidc-client-js-1.4.1
#> git checkout 1.4.1
```

Test it!

- Visit your machine URL
- After login you should see username
- For more info open your web-browser console - id_token is logged by the application script to it
  - Object contains all requested scopes, validity time, etc.

- Virtual machines and registered clients will be deleted one week after this Hands On :(

Own implementation

Hard stuff… Not hard at all

Own authorization code impl

- Own implementation is easy
- Login in 3 steps (authorization code flow)
  1) authentication of user and obtaining access code
  2) exchange access code and client_secret for access token
  3) exchange access token for user claims
- First has to obtain URLs from

  `https://login.cesnet.cz/oidc/.well-known/openid-configuration`
  - authorization endpoint
  - token endpoint
  - userInfo endpoint

Step 1

- Redirect browser to the authorization endpoint

```
https://login.cesnet.cz/oidc/authorize?
 response_type=code
 &scope=openid+email[+other_scopes...]
 &client_id=<client_id>
 &redirect_uri=<where to redirect back>
 &state=<random value against XSRF attack>
```

Step 2

- OP returns browser to the URL specified in parameter **redirect_uri** from the first step and adds two parameters - **state** and **code**
- Client has to verify that value of the state is equal to the value of the state from the first step
- Makes HTTP x-www-form-urlencoded POST request to the token endpoint

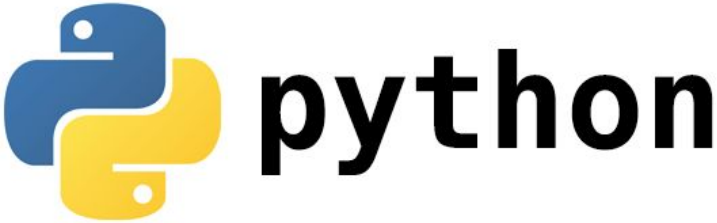
```
#>curl -d "grant_type=authorization_code&code=<value of
code>&redirect_uri=<value from the first
step>&client_id=<client_id>&client_secret=<client_secret>"
https://login.cesnet.cz/oidc/token
```

Step 3

- Token endpoint returns JSON message containing
  - in **access_token** - value of the access token
  - in **expires_in** - length of the validity
  - in **scope** - list of allowed scopes if it differs
  - in **id_token** - value of the id_token
- Client makes HTTP request to the userInfo endpoint with HTTP header Authorization

```
#>curl -H "Authorization: Bearer <access_token>"
https://login.cesnet.cz/oidc/userinfo
```

- Receives back JSON message with user claims



Python Social Auth

Make easy even easier

python-social-auth

- is an easy to setup social authentication/registration mechanism with support for several Python frameworks and auth providers
  - E.g. Django, Flask, Pyramid, CherryPy, Webpy
- Used in Galaxy portals

- Available at https://github.com/python-social-auth

- Only Client ID and Client Secret is needed to have EINFRA AAI integrated

Final words

- Got lost? Read some of these
  - Very nice OIDC and informal OIDC describing presentation https://www.slideshare.net/vladimirdzhuvinov/openid-connectexplained
  - Human readable OIDC description (+ pictures) https://connect2id.com/learn/openid-connect
  - Protocol web - https://openid.net/connect/
  - OIDC Core spec (hardcore stuff) - https://openid.net/specs/openid-connect-core-1_0.html
  - OIDC Certified libraries (RPs, OPs, …) - https://openid.net/developers/certified/

pic

Perun

Thank you for your attention

http://perun-aai.org

bucik@ics.muni.cz

cesnet

cerit scientific cloud

Resources

- Images:
- Perun logo + Proxy IdP OIDC Flow chart
- https://image.shutterstock.com/image-vector/info-icon-flat-vector-illustration-260nw-1420920695.jpg
- https://www.tirasa.net/images/blog/old/oidc.png
- https://d2wakvpiukf49j.cloudfront.net/media/uploads/zinnia/2017/05/19/oauth-retina-preview.1509495022.jpg
- https://d33wubrfki0l68.cloudfront.net/9ef5593f84648b223311c06be35560777b7dcf36/d16d7/assets-jekyll/blog/spring-boot-2.1/oauth2-and-oidc-a4379ecfcfd75f820b98f6a05951f33e33384532d89c410f9decf4ac7db2c5b8.png
- https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcSeDcv3oWK9dA1RwkpJqRn2RBYSYqCM39cbg2NHAgHwpOyp6C_n
- https://www.google.com/url?sa=i&url=http%3A%2F%2Fjmpplast.sk%2Fjs-logo%2F&psig=AOvVaw3cXO1uR1J94HMAps2Ao7PI&ust=1573912731533000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCOjb3vyv7OUCFQAAAAAdAAAAABAD
- https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRYgcs1ERvgMi3rDswp7zcfbtyqfxunhYm5nFa-B5s_jBk4nzvM
- https://image.shutterstock.com/image-vector/illustration-broken-computers-servers-white-260nw-484575997.jpg
- https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcQJMe2ruELFSmad5Jm4JcvBHq5ga98yzm3iD7LOTyIbVIlhKANM
- https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcQ_aIDlpNnbKRpINy3-Tkms2LTIBTjRXl3rzQ-A9KeZVFIgLCB2