oidc$_{client}$

Release 1.2.1

March 17, 2021

Contents

CSC OIDC Client is a lightweight aiohttp web application used for interacting with OIDC servers.

CHAPTER 1

Table of Contents

1.1 Configuration

OIDC Client has three configuration levels, that take priority from top to bottom:

- Environment Variable
- Configuration File
- Default Value

Default values can be seen in the configuration file parser, they are the right-most values per row:

```
def parse_config_file(path):
    """Parse configuration file."""
    config = ConfigParser()
    config.read(path)
    config_vars = {
        "app": {
            "host": os.environ.get("HOST", config.get("app", "host")) or "0.0.0.0",
↪# nosec
            "port": os.environ.get("PORT", config.get("app", "port")) or 8080,
            "name": os.environ.get("NAME", config.get("app", "name")) or "oidc-client
↪",
            "session_key": os.environ.get("SESSION_KEY", config.get("app", "session_
↪key")) or secrets.token_hex(16),
        },
        "cookie": {
            "domain": os.environ.get("DOMAIN", config.get("cookie", "domain")) or
↪"localhost",
            "token_lifetime": int(os.environ.get("TOKEN_LIFETIME", config.get("cookie
↪", "token_lifetime"))) or 3600,
            "state_lifetime": int(os.environ.get("STATE_LIFETIME", config.get("cookie
↪", "state_lifetime"))) or 300,
            "secure": bool(strtobool(os.environ.get("SECURE", config.get("cookie",
↪"secure")))) or True,
```

(continues on next page)

(continued from previous page)

```
            "http_only": bool(strtobool(os.environ.get("HTTP_ONLY", config.get("cookie
↪", "http_only")))) or True,
        },
        "aai": {
            "client_id": os.environ.get("CLIENT_ID", config.get("aai", "client_id"))␣
↪or "public",
            "client_secret": os.environ.get("CLIENT_SECRET", config.get("aai",
↪"client_secret")) or "secret",
            "url_auth": os.environ.get("URL_AUTH", config.get("aai", "url_auth")) or␣
↪None,
            "url_token": os.environ.get("URL_TOKEN", config.get("aai", "url_token"))␣
↪or None,
            "url_userinfo": os.environ.get("URL_USERINFO", config.get("aai", "url_
↪userinfo")) or None,
            "url_callback": os.environ.get("URL_CALLBACK", config.get("aai", "url_
↪callback")) or None,
            "url_redirect": os.environ.get("URL_REDIRECT", config.get("aai", "url_
↪redirect")) or None,
            "url_revoke": os.environ.get("URL_REVOKE", config.get("aai", "url_revoke
↪")) or None,
            "scope": os.environ.get("SCOPE", config.get("aai", "scope")) or "openid",
            "iss": os.environ.get("ISS", config.get("aai", "iss")) or None,
            "aud": os.environ.get("AUD", config.get("aai", "aud")) or None,
            "jwk_server": os.environ.get("JWK_SERVER", config.get("aai", "jwk_server
↪")) or None,
        },
    }
    return namedtuple("Config", config_vars.keys())(*config_vars.values())
```

The default values can be overwritten and saved to file in the `config.ini` configuration file. The configuration file has three basic sections: `app` for application configuration, `cookie` for cookie settings and `aai` for oidc client configuration. In addition, a fourth extra section for ELIXIR use case is provided as `elixir`. Custom sections can be added freely following the same manner.

1.1.1 Application Configuration

```
# ************************************
# Configuration for oidc-client web server
# ************************************
[app]
# Hostname for oidc-client server
host=0.0.0.0

# Port for oidc-client server
port=8080

# Name for this API shown at root endpoint `/`
name=oidc-client

# Secret key to encrypt session storage, must be exactly 32 characters
# If left empty, a session key will be generated with secrets.token_hex(16)
# Share this key with other services, which need to decrypt the AIOHTTP_SESSION cookie
session_key=
```

1.1.2 Cookie Settings

```
# *********************************
# Configuration for cookie management
# *********************************
[cookie]
# Domain where cookie will be set
domain=localhost:8080

# Lifetime of access_token cookie in seconds (1 hour is a good value)
token_lifetime=3600

# Lifetime of oidc_state cookie in seconds (5 minutes is a good value)
state_lifetime=300

# If True, cookie can only travel via HTTPS, if False, cookie can travel in HTTP/HTTPS
secure=True

# If True, cookie can't be read by JavaScript, if False, cookie can be read with␣
↪JavaScript
http_only=True
```

1.1.3 AAI Server Configuration

```
# ****************************
# Configuration for AAI server
# ****************************
[aai]
# Client ID of oidc-client
client_id=public

# Client secret for Client ID
client_secret=secret

# URL where authentication workflow begins
url_auth=https://login.elixir-czech.org/oidc/authorize

# URL that returns access token
url_token=https://login.elixir-czech.org/oidc/token

# URL for the userinfo endpoint at AAI
url_userinfo=https://login.elixir-czech.org/oidc/userinfo

# URL the AAI should return to after authentication
url_callback=localhost:8080/callback

# URL the OIDC Client should redirect to after authentication
url_redirect=localhost:5000

# URL to the token revocation endpoint at AAI
url_revoke=https://login.elixir-czech.org/oidc/revoke

# Claims requested for access token, for multiple values separate scopes by commas ','
scope=openid,ga4gh_passport_v1
```

(continues on next page)

(continued from previous page)

```
# Trusted issuers of access token, separate multiple issuers with commas ','
iss=https://login.elixir-czech.org/oidc/

# Intended audiences of access token, separate multiple audiences with commas ','
aud=audience1,audience2

# Server that returns JWK
jwk_server=https://login.elixir-czech.org/oidc/jwk
```

1.1.4 Environment Variables

The values in the configuration file can be overwritten with environment variables using the exact same name in all capital letters. For example:

To overwrite the web application port from `8080` to `3000`, one set the following environment variable:

```
export PORT=3000
```

Note: Environment variables `HOST` and `PORT` are used when running the web application with aiohttp. When running the web application in production server using gunicorn, environment variables `APP_HOST` and `APP_PORT` are used instead. More on this topic in the Setup Instructions.

1.2 Installation Instructions

Note: `oidc-client` requires python 3.6 or higher:

1.2.1 Development Server

For development, OIDC Client can be run without installation as a python module from the root folder `/oidc-client`:

```
cd oidc-client
python -m oidc_client.app
```

This starts the OIDC Client with `aiohttp.web.run_app`.

1.2.2 Installation

OIDC Client can be installed into the python environment with `pip install` from the root folder `/oidc-client`:

```
cd oidc-client
pip install .
```

The server can then be started with the following command:

```
start_oidc_client
```

This starts the OIDC Client with `aiohttp.web.run_app`.

1.2.3 Production Server

For production it is recommended to use gunicorn instead of aiohttp's default server for stability.

To start the production server, the web application must first be built, as described in the Installation section above. The host and port must also be provided beforehand in environment variables:

```
export APP_HOST=0.0.0.0
export APP_PORT=8080
```

The production server is started from the root folder `/oidc-client` with:

```
gunicorn oidc_client.app:init --bind $APP_HOST:$APP_PORT \
                              --worker-class aiohttp.GunicornUVLoopWebWorker \
                              --workers 4
```

1.2.4 Container Deployment

OIDC Client can also be built into a container and then be deployed. Builds must be initiated from the root folder `/oidc-client`.

To build OIDC Client into an image using `s2i`:

```
s2i build . centos/python-36-centos7 cscfi/oidc-client
```

To run the built image with docker:

```
docker run -d -p 8080:8080 cscfi/oidc-client
```

1.3 API Endpoints

OIDC Client consists of five endpoints: `/`, `/login`, `/logout`, `/callback` and `/token`.

1.3.1 Index

The index endpoint `/` is used as a healthcheck endpoint, it returns the name of the service as given in the configuration file.

1.3.2 Login

The login endpoint `/login` generates a state and saves this state to cookies, after which the user is redirected to the AAI server for authentication. Upon a successful authentication at the AAI, the user is returned to the `/callback` endpoint.

1.3.3 Logout

The logout endpoint `/logout` is used to destroy the access token cookie and to revoke the access token at the AAI. Upon a successful logout procedure, the user is returned to the `url_redirect` address from the configuration file.

1.3.4 Callback

The callback endpoint `/callback` acts as a landing site for the returning user from the AAI server. Upon returning to the OIDC Client from the AAI server, OIDC Client extracts `state` and `code` from the callback request, and uses these values to request a token from the AAI server. Upon a successful retrieval of an access token, the access token is saved to the browser cookies.

Some of the created cookies can be considered _unsafe_ (not *http_only*) for the purpose of displaying values in UI for logged in state.

1.3.5 Token

Display token from encrypted session storage for easy retrieval. Alternate way to inspect the access token is to look at the browser cookies.

1.3.6 Cookies

Cookies created and used by the OIDC Client and their default settings.

Cookie	Origin	Purpose	Lifetime	Secure	Http Only
AIO-HTTP_SESSION	/login	Store state at login to be checked upon callback. Store access token at callback to be displayed at token endpoint.	Session	True	True
access_token	/callback	Sent along same-domain requests for authorizing access to data	1 hour	True	True
logged_in	/callback	Used to display logged in state in UI	1 hour	True	False

CHAPTER 2

Indices and tables

- genindex
- modindex
- search