JWT / OpenID Connect

OpenID Connect is a protocol built on top of OAuth2 for enabling standardized exchange of identity information – the specification is available here: http://openid.net/connect/

PortalProtect can act as both an Identity Provider, and a Relying Party.

Please read the Oauth2 section here: OAuth2, and consider JWT as a specific type of Oauth2 bearer token, and OpenID Connect as a specification of the contents of the token – this is a bit simplified view, since OpenID Connect covers more than that, but it sets the scene.

The PortalProtect Authentication plugin; *JWTAuthenticationPlugin* implements support for both issuing OpenID Connect access and ID tokens, and for consuming tokens, parsing and validating them and making their information available to applications protected by PortalProtect.

Usage Scenario

The following describes a typical scenario, where PortalProtect is used as an Identity Provider, in this example; https://www.portalprotect.dk

In OpenID Connect, a “Relying Party” is the application or website which wants to access the authenticated user’s information – in this example https://www.example.com

The user browses https://www.example.com, and selects to authenticate – here, we assume that https://www.example.com is already registered as a valid client ID at www.portalprotect.dk

Now, the browser is being redirected to: https://www.portalprotect.dk/oauth2/auth?response_type=id_token&client_id=https%3A%2F%2Fwww.example.com&redirect_uri=https%3A%2F%2Fexample.com%2Foauth2response&scope=profile+email&state=12345 this is the authentication endpoint.

If we look at the parameters, there are:

response_type: id_token – possible types include code, token and id_token.
client_id: https://www.example.com – this is the registered client ID.
redirect_uri: https://example.com/oauth2response - The url to redirect back to
scope: profile+email – The type of information example.com wishes to get.
state: 12345 – sent back to example.com along with the token – allows it to correlate state.

Here, at portalprotect.dk, the application checks if the user is currently authenticated or not – if not, he is asked to login – e.g. using userid/password, nemid or any of the other available authentication methods. After login, the user is shown a page where he can see what information that example.com asks for, and he is given the option to confirm or deny providing this information to example.com.

Once he confirms, a JWT ID Token is created, and sent back to example.com on the url: https://example.com/oauth2response#id_token=xxxxxxx_jwt_token_xxxxxx&state=12345

The actual id_token contents is not sent to the server (notice the # in the url) but is accessible to javascript on the page. If the server wishes to get the token directly, a respone_type of “code” should be used instead – in this case, example.com is issued a short-living access code, and it needs to retrieve that code by making a call to https://www.portalprotect.dk/oauth2/token with its client id, client secret (password), and the code as parameter. Then it will get the id_token and eventually an access token returned.

Architecture / Design

The JWTAuthentication plugin has a list of tokens configured – a token configuration element contains information about token issuer, signing algorithm, certificates/keys needed to issue or verify it etc. You need a token configuration for each type of JWT token you wish to issue or consume.

When issuing a token, you can issue it to various clients – a client can be other parts of your site, or it can be 3rd party applications allowed to ask for user identity.

You can control how information about known clients are store – a class implementing the interface IOAuthDataStore provides the required configuration information needed. By default, PortalProtect has an implementation of it that reads the list of clients from either a property file, or from PortalProtect’s configuration for the session controller.

When a token is issued, it contains claims about the user – a claim can be the user’s id, username, and groups etc. – essentially any variable from the PortalProtect session.

When parsing tokens, you can use custom mapping - if defined, it allows you complete control over which fields are mapped, how they are mapped and allows you to do conversions/rewriting of values within them. This is useful, if e.g. the identity provider returns a firstName and a lastName as separate fields, but you need them combined - or if your userid needs to be based upon the userid from the provider, but you do not want an exact copy.

Configuration

For configuring Ceptor Gateway with OpenID Connect for Identity Provider, please refer to OpenID Connect Identity Provider for details on how to expose OpenID Connect discovery URL and JWK key lists to relying parties.

There is quite a lot of information that can be tailored to your needs by configuration – here is the relevant configuration located within the session controller for the JWTAuthentication plugin:

General

Property	Value
oauth2. datastoreclass	<classname – default: dk.itp.security.authentication.oauth.data.properties.Oauth2DataStoreProperties > Contains name of class implementing IOAuthDataStore and providing data about registered client IDs. Use *dk.itp.security.authentication.oauth.data.OAuthSQLStore* to read data from a database, and Use *dk.itp.security.authentication.oauth.data.OAuthIgniteStore* to read the data from Apache Ignite These two datastores are the same that API Management uses to store its configured API Partners and Applications within, so you should use the matching implementation here.
oauth2.accesstoken. datastoreclass	<classname - default: dk.itp.security.authentication.oauth.data.AccessTokenMemoryStore > Contains name of a class implementing *IAccessTokenDataStore* interface, to provide long-term storage of refresh tokens. You can use the *dk.itp.security.authentication.oauth.data.AccessTokenMemoryStore* for an in-memory store - note that this is only meant for testing. Use *dk.itp.security.authentication.oauth.data.AccessTokenSQLStore* to store access tokens in a database - in this case, oauth2. datastorename needs to be configured to point to the correct database datastore to use We also provide *dk.itp.security.authentication.oauth.data.AccessTokenIgniteStore* which uses Apache Ignite with persistence enabled as a clustered datastore. You can also implement your own store, or contact us to obtain another for your use.
oauth2. datastorename	<Name of datastore - default is datastore-primary> Name of datastore to use when saving access tokens or refresh tokens in a database.
oauth2.refreshtoken. datastoreclass	<classname - no default> Contains name of a class implementing *IRefreshTokenDataStore* interface, to provide long-term storage of refresh tokens. You can use the *dk.itp.security.authentication.oauth.data.RefreshTokenMemoryStore* for an in-memory store - note that this is only meant for testing. Use *dk.itp.security.authentication.oauth.data.RefreshTokenSQLStore* to store these in a database. We also provide *dk.itp.security.authentication.oauth.data.RefreshTokenIgniteStore* which uses Apache Ignite with persistence enabled as a clustered datastore. You can also implement your own store, or contact us to obtain another for your use.
oauth2.defaulttoken	<Token name – default is the first configured token> Sets the name of the token to be used as default if nothing else is provided.
oauth2.tokens	<List of token names – separated by comma or semicolon> List of the configured tokens. A token contains information about how to generate or parse it, which certificate/keys to use, crypto algorithm etc.
oauth2.tokens.jwks	<List of token names - separated by comma or semicolon> List the tokens (must be in *oauth2.tokens*) that should be included in the generated jwks.json key list. Any tokens *not* mentioned here are hidden and not included in the published jwks URL.
openid.scopes	<List of scope names – separated by comma or semicolon > List of the configured scopes A scope is e.g. profile or email – it specifies which information to provide and from which variables in the PP session. Special scopes are *openid* and *offline_access* - the first is always required to be supported, and the 2nd enables use of refresh_token in the authorization_code flow.
openid.fields	<List of field names – separated by comma or semicolon > For complex claim values, such as address, this specifies the sub items contained within.
openid. identityproviders	<List of identity provider names – separated by comma or semicolon > An identity provider is a foreign identity provider where we can lookup an access token and/or id_token using an authorization code.

Properties for OAuth2 Client Datastore

For the OAuth2DataStoreProperties implementation of the IOAuthDataStore, you can specify the following configuration:

Property	Value
oauth2. clientpropertiesfile	<File to load remaining properties from> If specified, the remaining properties are loaded from this file – if not, they are instead loaded from the configuration within portalprotect-configuration.xml.
oauth2.clients	<List of client names – separated by comma or semicolon > List the clients to load.
The remaining properties start with: **oauth2.client.xxxx**	Where xxxx is the client name
.clientid	<Client ID> Contains the client ID – in OpenID connect, the client ID must start with https://
.secret	<password – optionally obfuscated/encrypted> Client Secret – used when client gets issued an authorization code, and uses that code plus the client ID and client secret to access an access token and id token.
.allowedscopes	<scope name list> Lists the scopes this client is allowed to request – can be used to restrict certain clients so they can only access limited information about the user.
.allowedredirecturis	<redirect URI list> Lists of valid redirect URIs for this client.
.allowedlogouturis	<Logout URI list> List of valid logout URIs for this client.
.validgranttypes	<Grant types> List of valid grant types for this client, can contain; authorization_code,implicit,hybrid,refresh_token See OpenID Connect specification for details. In short; authorization_code requires a server to use the client secret to gain an access token, impliciet allows javascript in the browser to access it, and hybrid is a combination of the two. *refresh_token* is required to support the *offline_access* scope.
. accesstokenvalidityseconds	<Seconds – default: 60> Number of seconds that an issued access token is valid for.
. maximumexpirationminutes	<Minutes – default: 60> Maximum expiration time of a generated ID token in minutes.
. refreshtokenvalidityseconds	<Seconds – default: 60> Maximum refreshtoken validity seconds.
.tokenname	<Token name> Name of token to use for this client ID – specifies the token (algorithm, keys etc.)
.accesstokentype	<JWT or UUID> Specifies which type of access token to issue – a JWT access token contains information that can be read, where a UUID-type of access tokens is a unique ID key to the access token. The UUID is significantly smaller and does not contain any information in itself.

Custom Mapping

You can configure a number of custom mappers which you can use from tokens or identity providers.

Property	Value
oauth2.mappers	<comma or semicolon separated list of mappers> List of mappers to load - e.g. mapper1,mapper2
Property **Name starts with oauth2.mapper.xxxx where xxxx is the mapper name.**	**Value**
userid	Mapped value - result is placed into session as user ID
username	Mapped value - result is placed into session as username
groups	Mapped value - result is placed into session as list of groups
customerid	Mapped value - result is placed into session customer ID
isinternal	Mapped value - result is placed into session as boolean value of isInternal (must be true or false)
agreementid	Mapped value - result is placed into session as agreement ID
authlvl	Mapped value - result is placed into session as authentication level - must be a valid integer value
_state_XXXX where XXXX is name of statevariable	Mapped value - result is placed into session as a state variable - use e.g. _state_username to set the state variable *username* instead of the username field in the session.
XXXX where XXXX is name of state variable	Mapped value - result is placed into session as state variable with the given key.

Mapped Value

The mapped value is a string, where macros are replaced with the appropriate contents - a macro can be%{claim:XXXX} where XXXX is the name of a claim key in the JSON JWT token, or token returned from an identity provider such as facebook.

macros are in the form %{type:name} where type can be *claim, base64, urlencode, htmlencode, base64decode, urldecode* - the default is *claim* if not specified. If set to e.g. base64, the value will be treated as base64 and decoded.
Like with Ceptor Gateway - Scripts and Macros you can use scripts (javascript, python or groovy) and %{rewrite} macros to do more advanced transformations of values. When scripts are used, the variable *context* will point to an object which contains two variables; *session* pointing to the users Ceptor Session, and *jo* pointing to a JSONObject containing the claims to map.

Examples:

```
oauth2.mapper.xxxx.username=%{claim:firstName} %{claim.lastName}
oauth2.mapper.xxxx.salary=%{script}salary(); function salary() {var json = JSON.from(context.jo.toJSONString());
return (json.monthlySalary * 12) + json.yearlyBonus;}
oauth2.mapper.xxxx.valuesAreFrom=Some Custom Provider
```

Token

For each token contained in oauth2.token, the following is configured:

Property Name starts with oauth2.token.xxxx	Value
.issuer	<Issuer name> Name of issuer – e.g. https://www.portalprotect.dk
.validaudiences	<List of names, separated by comma or semicolon> List of audience names that are valid for this token
.algorithm	<JWT signing algorithm name – default is RS256> Must be a valid JWT signing algorithm name, supported algorithms are; HS256, HS384, HS512, RS256, Rs384, RS512, ES256, ES384, ES512, PS256, PS384, PS512

.keyid	<Key identifier> Key ID – for the JWT header “kid” field.
.claims	<List of claims – default: “sub=userid;groups=groups;name=username” > See the section about claim name/value paris below
. usernameAttribute Name	<Claim name – default: “name”> When parsing a claim issued by someone else – which attribute to look for the users name within.
. useridAttributeNa me	<Claim name – default: “sub”> When parsing a claim issued by someone else – which attribute to look for the users id witin.
.roleAttributeName	<Claim name – default: “groups”> When parsing a claim issued by someone else – which attribute to look for the list of user groups within.
.rolePattern	<Stringmatcher pattern – default: “*”> Which roles/groups to include in the token when creating claims. Example: “^admin*” to add all group names except those starting with admin.
. attributesToStoreIn Session	<Stringmatcher pattern – default: “*”> Which attributes to store within PPs session when parsing a token issued by others.
.relaxKeyChecks	<true | false – default: “false”> Set to true to relax key checking, meaning to allow weak keys to be used for signing.
.openidconnect	<true | false – default: “false”> Specifies if the token should be openid connect compliant or just a regular JWT token.
. expiresAtExactTime	<true | false – default: “false”> If true, when parsing a JWT token, with an expiration time within it, the token will expire at that exact time/date and will no longer be valid. If false, the expiration time will only be used at authentication time, and the resulting session will expire using normal idle timeout settings.
.requireSubject	<true | false - default: "true"> When parsing tokens, normally they require a subject (the "sub" claim) - if you set this to false, tokens are accepted without a subject. (Requires Ceptor 6.2.7 or later)
.signerCertificates	<List of certificate filenames> List of files containing valid certificates that can be used to verify the signature of this token.
. signerCertificatesU RL	<URL> Place to load additional signer certificates from – e.g. https://www.googleapis.com/oauth2/v1/certs The certificates must be in a JSON object, with key/value pairs where the value is the certificate.
. signerCertificatesR efreshIntervalMinut es . signerCertificatesR efreshIntervalHours	<Number of minutes / hours> - default: 60 minutes. If number of minutes is not specified, number of hours is read - otherwise, only number of minutes is used. Specifies the number of minutes to cache the certificates read from the authentication provider - set to 0 to re-read it every time.
. acceptedServerCe rtificates	<List of certificate files, separated by semicolon or comma> If the default cacerts trusted CA/root certificates is not enough, you can add additional certificates here. This applies to the *signerCertificatesURL*
.verifyServerCert	<true or false, default: true> Set to false to disable validation of the SSL server certificate for the *signerCertificatesURL*

. verifySSLHostname	<true or false, default: true> Set to false to dsable hostname validation – when true, the hostname in the URL must match the hostname in the SSL server certificate.
.keystore.provider	<JCE provider name> Name of the JCE provider to use when loading the keystore
.keystore.type	<JCE keystore type – default: “PKCS12”> Specifies the format of the keystore, e.g. PKCS12 or JKS.
.keystore.file	<filename> Specifies the filename of the keystore to load the keys from.
.keystore. password	<password> Password for the keystore – can optionally be encrypted/obfuscated
.keystore.privalias	<alias name> Alias of the private key within the keystore to use – if no alias, the first key found will be used.
.keystore.certalias	<alias name> Alias of the certificate within the keystore to use – if no alias, uses the first found certificate.
.jceprovider	<JCE provider name> Name of the JCE provider to use when signing or validating signature of the JWT token.
.secretkey	<Secret key> For algorithms starting with HS, a secret shared key is used – this should be avoided in production environments, since anyone in possession of the shared key used to validate JWT tokens can also is the same key to issue new tokens.
. clockSkewSeconds	<Time difference in seconds> - Default: 0, Requires v5.61 Set the clock skew allowed when validating expiration / not-before timestamps in the token - allows adjusting for time difference between machines.
.expirationminutes	<Minutes> - Default: 10 When creating a token, this is the expiration time as minutes in the future.
. notBeforeMinutesI nPast	<Minutes> - Default: 2 When creating a token, this is the number of minutes in the past to set *nbf* (not before) attribute to.
. customfieldmapper	<Name of custom mapper> If specified, when tokens are parse, they are not just copied - instead, the fields in the custom mapper are constructed based upon the claims provided as input. Note that *attributesToStoreInSession* has no effect if custom field mapping is enabled.

Scopes

OpenID Scope names are mapped to claim names and attributes by configuration.

Property **Name starts with openid.scope. xxxx**	**Value**
.description	<String> Description of the scope
.idtoken	<List of claim name/value pairs> List of claim name/value pairs to include in the ID token for this scope.
.accesstoken	<List of claim name/value pairs> List of claim name/value pairs to include in the access token – (if the access token is a JWT token and not an UUID) for this scope.

.userinfo	<List of claim name/value pairs> List of claim name/value pairs to include in the userinfo for this scope. The userinfo is a JSON object that can be requested from https://xxxx/oauth2/userinfo by providing a valid access token as input.

Claim name/value pairs

A list of claim names is specified as e.g. sub=userid;groups=groups;name=username – it is a semicolon separated list of key/values.

The key is the name of the claim in the JWT token, and the value has special meaning; it refers to attributes within PortalProtect's session;

Value	**Meaning**
null	Leave the claim out – has the same meaning as if the claim was not present at all, but can be used as a placeholder in the configuration where it can be later changed to another attribute.
userid	Users ID
username	Users name
sessionid	PP Session ID
customerid	Customer ID
isinternal	True if user is internal, false if not
agreementid	Agreement ID
authmethod	Authentication method
authlvl	Authentication level
__literal	Literal value – if it starts with two underscore characters, it is taken as a litereral – e.g. "salary=__secret" will create the claim: {"salary": "secret"} In the JWT token.
<field name>	If the value matches a configured field name, e.g. "address" then it will be used as a complex/nested field – e.g. {"address": {"country": "DK", "street_address": "Kronprinsessegade 54", "postal_code": "1306"}}
_state_xxxx	Refers to a state variable within the session, e.g. state=_state_username picks out the value from the state variable named "username" instead of the field "username" within the session.
Anything else	Any other value is matched up against a state variable within the session.

Fields

Property	**Value**
openid.fields	<List of field names, separated by comma or semicolon> List of defined fields
openid.field.xxxx	<xxxx is the fieldname – claims name/value pairs> Example: street_address=address1;locality=city;region=state;postal_code=postal;country=country

Identity Providers

Property	**Value**

openid.identityproviders	<List of identity provider names – separated by comma or semicolon > An identity provider is a foreign identity provider where we can lookup an access token and/or id_token using an authorization code.
xxxx below is replaced with the name of the identity provider	
openid.idp.xxxx.clientid	<String> OpenID Connect Client ID
openid.idp.xxxx.secret	<Password – can be encoded/encrypted> OpenID Client Secret – a kind of password used when looking up authorization code
openid.idp.xxxx.tokenurl	<URL – https required> URL to token service – needed to exchange authorization_code for token
openid.idp.xxxx.facebook	<true | false> (requires minimum Ceptor v5.67) True if this identity provider is facebook
openid.idp.xxxx. facebookurl	URL to use when calling with access_token to retrieve fields to place within session. Default value is: `https://graph.facebook.com/me`
openid.idp.xxxx. facebookfields	<List of fields> Comma separated list of fields to send to the facebookurl - list of fields to query. Default (corresponding to what is available in facebook's default *public_info* scope) is: `id,cover,name,first_name,last_name,age_range,link,gender,locale,picture,timezone, updated_time,verified`
openid.idp.xxxx.linkedin	<true | false> (requires minimum Ceptor v5.70.0) True if this identity provider is linkedin
openid.idp.xxxx.linkedinurl	URL to use when calling with access_token to retrieve user profile information to place within session. Default value is: `https://api.linkedin.com/v1/people/~`
openid.idp.xxxx. linkedinfields	<List of fields> Comma separated list of fields to send to the linkedinurl- list of fields to query. Default (see available fields at https://developer.linkedin.com/docs/fields/basic-profile) is: `id,firstName,lastName,headline,num-connections,picture-url,formatted-name,summary,location, public-profile-url`
openid.idp.xxxx. customfieldmapper	<Name of custom mapper> If specified, when tokens are parse, they are not just copied - instead, the fields in the custom mapper are constructed based upon the claims provided as input. If mapping a JWT token, and a customfieldmapper is specified on an identity provider, this supercedes what might be specified on the individual token.

Sample Configuration

The following is a complete example of the configuration with all the fields specified above.

```
<group name=”OpenID_Connect” description=”OpenID Connect related settings”>
  <property name=”openid.scopes” value=”openid,email,profile,address,phone” description=”The list of configured
```

```
openid scopes" />
  <property name="openid.scope.openid.userinfo" value="sub=userid" description="Attributes to include in userinfo
for openid scope"/>
  <property name="openid.scope.openid.accesstoken" value="sub=userid" description="Attributes to include in access
token for openid scope"/>
  <property name="openid.scope.openid.idtoken" value="sub=userid" description="Attributes to include in id token
for openid scope"/>
  <property name="openid.scope.openid.description" value="Essential information" description="Description of
openid scope"/>
  <property name="openid.scope.email.userinfo" value="email=email1;email_verified=null" description="Attributes to
include in userinfo for email scope"/>
  <property name="openid.scope.email.idtoken" value="" description="Attributes to include in userinfo for email
scope"/>
  <property name="openid.scope.email.accesstoken" value="" description="Attributes to include in access token for
email scope"/>
  <property name="openid.scope.email.description" value="Email address" description="Description of email scope"/>
  <property name="openid.scope.profile.userinfo" value="name=username;family_name=null;given_name=null;
middle_name=null;nickname=null;preferred_username=null;profile=null;picture=null;website=null;gender=gender;
birthdate=birthdate;zoneinfo=null;locale=null;updated_at=null" description="Attributes to include in userinfo for
email scope"/>
  <property name="openid.scope.profile.idtoken" value="" description="Attributes to include in userinfo for email
scope"/>
  <property name="openid.scope.profile.accesstoken" value="" description="Attributes to include in access token
for email scope"/>
  <property name="openid.scope.profile.description" value="Email address" description="Description of email scope"
/>
  <property name="openid.scope.address.userinfo" value="address=address" description="Attributes to include in
userinfo for email scope"/>
  <property name="openid.scope.address.idtoken" value="" description="Attributes to include in userinfo for email
scope"/>
  <property name="openid.scope.address.accesstoken" value="" description="Attributes to include in access token
for email scope"/>
  <property name="openid.scope.address.description" value="Your address" description="Description of email scope"/>
  <property name="openid.scope.phone.userinfo" value="phone=mobilephone;phone=phone;phone_verified=null"
description="Attributes to include in userinfo for email scope"/>
  <property name="openid.scope.phone.idtoken" value="" description="Attributes to include in userinfo for email
scope"/>
  <property name="openid.scope.phone.accesstoken" value="" description="Attributes to include in access token for
email scope"/>
  <property name="openid.scope.phone.description" value="Your phone number" description="Description of email
scope"/>
  <property name="openid.fields" value="address" description="List of custom fields for which sub attributes
exists"/>
  <property name="openid.field.address" value="street_address=address1;locality=city;region=state;
postal_code=postal;country=country" description="Specifies which fields are included within the address JSON
object"/>
</group>
<group name="Oauth2" description="Oauth2 related settings">
  <property name="oauth2.clients" value="sample" description="List of Oauth2 token names"/>
  <property name="oauth2.client.sample.accesstokentype" value="JWT" description="Type of access token to generate
for use in bearer token – either JWT or UUID"/>
  <property name="oauth2.client.sample.clientid" value="https://www.example.com/" description="Client ID"/>
  <property name="oauth2.client.sample.secret" value="secret" description="Client Secret"/>
  <property name="oauth2.client.sample.allowedscopes" value="openid;profile;email" description="Allowed scopes"/>
  <property name="oauth2.client.sample.allowedredirecturis" value="https://www.example.com/oauth2" description="
Allowed redirectURIs"/>
  <property name="oauth2.client.sample.validgranttypes" value="authorization_code,implicit,hybrid" description="
Valid oauth2 grant types"/>
  <property name="oauth2.client.sample.accesstokenvalidityseconds" value="3600" description="Number of seconds an
issued access token is valid for"/>
  <property name="oauth2.client.sample.maximumexpirationminutes" value="120" description="Maximum number of
minutes an expiration time can be set in the future, 0 or -1 for no expiration"/>
  <property name="oauth2.client.sample.refreshtokenvalidityseconds" value="-1" description="Number of seconds a
refresh token is valid for – 0 or -1 to disable"/>
  <property name="oauth2.client.sample.tokenname" value="sample" description="Which oauth2 token name to use for
this client"/>
</group>
<group name="Oauth2_JWT" description="Oauth2 JWT related settings">
  <property name="oauth2.datastoreclass" value="dk.itp.security.authentication.oauth.data.properties.
Oauth2DataStoreProperties" description="Java class to handle reading Oauth2 datastore"/>
  <property name="oauth2.defaulttoken" value="sample" description="Name of the default token to use if nothing
```

```
else is requested”/>
  <property name=”oauth2.tokens” value=”sample;sample2” description=”List of Oauth2 JWT tokens names”/>
  <property name=”oauth2.tokens.jwks” value=”sample” description=”List of Oauth2 JWT tokens names which are
included in the JWK key list”/>

  <property name=”oauth2.token.sample.validaudiences” value=”https://www.example.com/” description=”List of valid
audiences, used when validating JWT token”/>
  <property name=”oauth2.token.sample.issuer” value=”https://www.portalprotect.dk” description=”OAUth2 Issuer”/>
  <property name=”oauth2.token.sample.algorithm” value=”RS256” description=”Algorithm to use for JWT token”/>
  <property name=”oauth2.token.sample.expirationminutes” value=”10” description=”Token Expiration in minutes”/>
  <property name=”oauth2.token.sample.notBeforeMinutesInPast” value=”2” description=”Number of minutes in past to
set in token”/>
  <property name=”oauth2.token.sample.keyid” value=”k1” description=”Key ID for this token”/>
  <property name=”oauth2.token.sample.claims” value=”sub=userid;groups=groups;name=username” description=”Claims
to add to token”/>
  <property name=”oauth2.token.sample.usernameAttributeName” value=”name” description=”Name of attribute to
contain the username”/>
  <property name=”oauth2.token.sample.roleAttributeName” value=”groups” description=”Name of attribute roles
/groups are stored in”/>
  <property name=”oauth2.token.sample.rolePattern” value=”^admin*” description=”Only groups matching this pattern
are set”/>
  <property name=”oauth2.token.sample.relaxKeyChecks” value=”true” description=”True to not perform key strength
checks”/>
  <property name=”oauth2.token.sample.openidconnect” value=”true” description=”True to create openid connect
compatible token”/>
  <property name=”oauth2.token.sample.attributesToStoreInSession” value=”*” description=”Attributes from ID token
to store in session”/>
  <property name=”oauth2.token.sample.signerCertificates” value=”” description=”List of valid signer certificates
when receiving JWT token”/>
  <property name=”oauth2.token.sample.jceprovider” value=”BC” description=”Name of JCE provider to use for creating
/validating tokens”/>
  <property name=”oauth2.token.sample.keystore.type” value=”PKCS12” description=”Keystore to load private key
/certificate from”/>
  <property name=”oauth2.token.sample.keystore.file” value=”${portalprotect.home}/config/x509/nemid/kr.pfx”
description=””/>
  <property name=”oauth2.token.sample.keystore.password” value=”Zaq12wsx” description=””/>
  <property name=”oauth2.token.sample.keystore.provider” value=”BC” description=””/>
  <property name=”oauth2.token.sample.keystore.privalias” value=”” description=””/>
  <property name=”oauth2.token.sample.keystore.certalias” value=”” description=””/>

  <property name=”oauth2.token.sample2.issuer” value=”https://www.portalprotect.dk” description=”OAUth2 Issuer”/>
  <property name=”oauth2.token.sample2.algorithm” value=”HS256” description=”Algorithm to use for JWT token”/>
  <property name=”oauth2.token.sample2.expirationminutes” value=”10” description=”Token Expiration in minutes”/>
  <property name=”oauth2.token.sample2.notBeforeMinutesInPast” value=”2” description=”Number of minutes in past to
set in token”/>
  <property name=”oauth2.token.sample2.claims” value=”sub=userid;groups=groups;name=username” description=”Claims
to add to token”/>
  <property name=”oauth2.token.sample2.usernameAttributeName” value=”name” description=”Name of attribute to
contain the username”/>
  <property name=”oauth2.token.sample2.roleAttributeName” value=”groups” description=”Name of attribute roles
/groups are stored in”/>
  <property name=”oauth2.token.sample2.rolePattern” value=”^admin*” description=”Only groups matching this pattern
are set”/>
  <property name=”oauth2.token.sample2.relaxKeyChecks” value=”false” description=”True to not perform key strength
checks”/>
  <property name=”oauth2.token.sample2.openidconnect” value=”true” description=”True to create openid connect
compatible token”/>
  <property name=”oauth2.token.sample2.attributesToStoreInSession” value=”*” description=”Attributes from ID token
to store in session”/>
  <property name=”oauth2.token.sample2.jceprovider” value=”BC” description=”Name of JCE provider to use for
creating/validating tokens”/>
  <property name=”oauth2.token.sample2.secretkey” value=”secret” description=”Secret key”/>
  <property name=”openid.identityproviders” value=”google” description=”OpenID Connect identity providers, where
we can exchange an authorization code for an id/access token”/>
  <property name=”openid.idp.google.clientid” value=”371213948273-79eceu24cm64ft69pln0hk2lfapok1bq.apps.
googleusercontent.com” description=”Client ID”/>
  <property name=”openid.idp.google.secret” value=”{encoded}806F9FE0C7CBC28D5777D6DE91772DA4961482568956695A”
description=”Client Secret”/>
  <property name=”openid.idp.google.tokenurl” value=”https://accounts.google.com/o/oauth2/token” description=”URL
to use when exchanging authorization code for a token”/>
</group>
```

Example Request Flow

Below, is an example of the request and response flow with the example configuration, using cURL (https://curl.haxx.se/) from a command-line to generate the requests.

In this example, portalprotect is running locally on port the default port; 4443.

Run curl like this:

```
curl –insecure -s -D - -b cookies.txt -c cookies.txt “https://localhost:4443/oauth2/auth?client_id=https://www.example.com/&redirect_uri=https://www.example.com/oauth2&response_type=code&scope=profile+email”
```

This will create a cookies.txt file, looking like this:

This will create a cookies.txt file, looking like this:

```
# Netscape HTTP Cookie File
 # http://curl.haxx.se/docs/http-cookies.html
 # This file was generated by libcurl! Edit at your own risk.
 #HttpOnly_localhost        FALSE        /            TRUE        0            sslsessionid
5C5CC94AAE5F796D4C21CEB67D0570715F45DD92BE9153E99CF16E0662B358EF3825DCBD0B89EDF248524EA7A13399FB74DB4859F73605CA794D0B
AD6F72EA21ED7A950F1B3CFE81C886FFEEFA518145C1D789C5AD7FA1D1A0A62EADB8A54CA18B41934D77F123E5B7
 #HttpOnly_localhost        FALSE
/            FALSE        0            defaultselectedServer      aM9eMlE3kXkSL6iOdh4qzVV3wkk=
```

Now, log into PP using your browser – then press F12, find the sslsessionid cookie, and copy that. Edit the cookies.txt file, and replace the sslsessionid cookie with the one you got from the browser – the one representing your authenticated session.

Now, run this again(the first time was just to get a cookies.txt file created):

```
curl –insecure -s -D - -b cookies.txt -c cookies.txt “https://localhost:4443/oauth2/auth?client_id=https://www.example.com/&redirect_uri=https://www.example.com/oauth2&response_type=code&scope=profile+email”
```

You should get a response looking like this:

```
HTTP/1.1 302 Found
 X-Content-Type-Options: nosniff
 Strict-Transport-Security: max-age=31536000 ; includeSubDomains
 X-Frame-Options: sameorigin
 X-XSS-Protection: 1; mode=block
 Content-Security-Policy: script-src ‘unsafe-inline’
 Location: /confirm_oauth.jsp
 Via: 1.1 dispatcher1 (PortalProtect/5.31)
 Content-Length: 0
```

This redirects you to the confirmation page, where you have to confirm that you want to allow www.example.com access to your information.

Pretend you are doing this, by calling /oauth2/confirm like this:

```
curl –insecure -s -D - -b cookies.txt -c cookies.txt “https://localhost:4443/oauth2/confirm”
```

This will reward you with a response similar to this:

```
HTTP/1.1 302 Found
 X-Content-Type-Options: nosniff
 Strict-Transport-Security: max-age=31536000 ; includeSubDomains
 X-Frame-Options: sameorigin
 X-XSS-Protection: 1; mode=block
 Content-Security-Policy: script-src ‘unsafe-inline’
 Location: https://www.example.com/oauth2?
code=Y2IyMmUzOTItYjY1NS00Zjc5LWFkMmQtNjEwYjVjMzY1NzFiYmNhY2EzNDItMjE4Yi00YzEyLWE2ZWMtNTYyYWM1NmM1NzA5
 Via: 1.1 dispatcher1 (PortalProtect/5.31)
 Content-Length: 0
```

As you can see, you got a redirect to https://www.example.com with your authentication code as an URL parameter.

Now, www.example.com needs to request a token using the provided code:

This is done like this – note it is sent as a POST request, which is required by the OpenID Connect specification:

```
curl –insecure -s -D - --data “client_id=https://www.example.com/&client_secret=secret&grant_type=authorization_code&redirect_uri=https://www.example.com/oauth2&code=Y2IyMmUzOTItYjY1NS00Zjc5LWFkMmQtNjEwYjVjMzY1NzFiYmNhY2EzNDItMjE4Yi00YzEyLWE2ZWMtNTYyYWM1NmM1NzA5” https://localhost:4443/oauth2/token
```

Which gives you this response

```
HTTP/1.1 200 OK
 X-Content-Type-Options: nosniff
 Strict-Transport-Security: max-age=31536000 ; includeSubDomains
 X-Frame-Options: sameorigin
 X-XSS-Protection: 1; mode=block
 Content-Security-Policy: script-src 'unsafe-inline'
 Set-Cookie:
sslsessionid=5C5CC94AAE5F796D4C21CEB67D0570715F45DD92BE9153E99CF16E0662B358EF3825DCBD0B89EDF272B1289138B990D104F8AFDC2
3A0F033A20C80C91EB8BBD5276262189DBC6E7A6FB0819378A60C886DD0154CA7A5D3EDB57045C485B05144AA979D2759078D13D5; Path=/;
HttpOnly; Secure
 Cache-Control: no-store
 Pragma: no-cache
 Content-Type: application/json;charset=UTF-8
 Set-Cookie: defaultselectedServer=aM9eMlE3kXkSL6iOdh4qzVV3wkk=; path=/; HttpOnly
 Cache-Control: no-cache="Set-Cookie"
 Via: 1.1 dispatcher1 (PortalProtect/5.31)
 Content-Length: 1406

{"access_token":"eyJraWQiOiJrMSIsImFsZyI6IlJTMjU2In0.
eyJpc3MiOiJodHRwczovL3d3dy5wb3J0Ywxwcm90ZWN0LmRrIiwic3ViIjoia2ltcmFzIiwiYXVkIjoiaHR0cHM6Ly93d3cuZXhhbXBsZS5jb20vIiwiZX
hwIjoxNDU1MzYxODU3LCJqdGkiOiJxZXZEY200bEJEcllsWVFsa1RhanZBIiwiaWF0IjoxNDU1MzU4MjU3LCJuYmYiOjE0NTUzNTgxMzcsImdyb3VwcyI6
W10sIm5hbWUiOiJLaW0gUmFzbXVzc2VuIn0.g5OISsc4R3lmJIvC4_RJK2NC_S8raNG-
jIyXP_855Uwvy_tuIuPUHHssp_COcmSPyEuDNZp_GPVuwG9tKM2YKHO36uv86g7KtnPbX4zdhA-TJWej0MsMRLnw-
zworuT2HqrdRko4PjNrDqrQ07S4JYV4PNP6kN4M-q8dsKzpqHdqXB_A_-vdvlzOc_7GL8AjrnSKuA9MlmNhJ7UYwwNHTJSq-
mEQg_rLVuDvFXT0aVJA3G2DFQ-
nnVGmcBWh2d0Eku45KtWvpgfBH4AuSeZzoccN1wyDujklxzlFPJQ6GJbTWUFRZbgzmTkZRYMSYQxHv6XgjJonA6pN_eW4YicdJg","id_token":"
eyJraWQiOiJrMSIsImFsZyI6IlJTMjU2In0.
eyJpc3MiOiJodHRwczovL3d3dy5wb3J0Ywxwcm90ZWN0LmRrIiwic3ViIjoia2ltcmFzIiwiYXVkIjoiaHR0cHM6Ly93d3cuZXhhbXBsZS5jb20vIiwiZX
hwIjoxNDU1MzU4ODU3LCJqdGkiOiI0ZlVScFFYRUMzamt5UGJYUnhfcHBBIiwiaWF0IjoxNDU1MzU4MjU3LCJuYmYiOjE0NTUzNTgxMzcsImdyb3VwcyI6
W10sIm5hbWUiOiJLaW0gUmFzbXVzc2VuIiwiYXV0aF90aW1lIjoiMTQ1NTM1NzQ2MCJ9.PiwrdPeks-OXK-IogWfRv9Pt3aThp-
FRLijOYzMyOkq5OsciiNMgY8a1fEz5EBGE2VxKhxHC7Lg_P-
vtWYU9YBYXnYMiQ_iCyaBE_0UjWvPHdxssESgdaxHmIEo4WnwaRV28pzhbiSGZmGAGzGjMv7AN037aYlyJReSZ8AoxZHT9m7f3dSVoMvPl1i3EzMXfxPCy
QJf8_qA1AuI3P5aob5bg5ckllUNPX7VHdGr3w_roX4WCZhXaZaU0RyfQwj2C70wxBKPz1-XRBvCXmKLQgmZI3EgjJckNW-
H_A6OOIRDxJWfxvYyTHyPH9TamA7aEkD7j0UeSyMUeWnIzyqA3ow","token_type":"bearer","expires_in":3600}
```

The response contains an access token, which you can use to request access to the users userinfo, and an id_token from which you can see the users identity.

If you take the ID token, and paste it into the JWT debugger on http://jwt.io then it will parse it and display the result. In this case, the result looks like this:

```
{
   "iss": "https://www.portalprotect.dk",
   "sub": "kimras",
   "aud": "https://www.example.com/",
   "exp": 1455358857,
   "jti": "4fURpQXEC3jkyPbXRx_ppA",
   "iat": 1455358257,
   "nbf": 1455358137,
   "groups": [],
   "name": "Kim Rasmussen",
   "auth_time": "1455357460"
 }
```

In the above example, the token was requested using the OpenID Connect grant type authorization_code flow which first gives the application a code, which needs to be exchanged for a token by the webserver at www.example.com providing its client_id and client_secret.

Since the client_secret needs to be protected, it cannot be used directly by the browser – so for browser applications, the implicity flow is used instead – in this case the id_token and access_token are provided directly to the browser

```
curl –insecure -s -D - -b cookies.txt -c cookies.txt "https://localhost:4443/oauth2/auth?client_id=https://www.
example.com/&redirect_uri=https://www.example.com/oauth2&response_type=id_token&scope=profile+email&nonce=12345"
```

Notice that the response_type is this time set to *id_token*, and not *code*. This causes the implicit authentication flow to be used. Also note, that the request now contains a nonce parameter with a supposedly unique value.

This gives you this response, showing you the confirmation page:

```
HTTP/1.1 302 Found
 X-Content-Type-Options: nosniff
 Strict-Transport-Security: max-age=31536000 ; includeSubDomains
 X-Frame-Options: sameorigin
 X-XSS-Protection: 1; mode=block
 Content-Security-Policy: script-src 'unsafe-inline'
 Location: /confirm_oauth.jsp
 Via: 1.1 dispatcher1 (PortalProtect/5.31)
 Content-Length: 0
```

As before, the user confirms by sending:

```
curl –insecure -s -D - -b cookies.txt -c cookies.txt "https://localhost:4443/oauth2/confirm"
```

This rewards you with this response:

```
HTTP/1.1 302 Found
 X-Content-Type-Options: nosniff
 Strict-Transport-Security: max-age=31536000 ; includeSubDomains
 X-Frame-Options: sameorigin
 X-XSS-Protection: 1; mode=block
 Content-Security-Policy: script-src 'unsafe-inline'
 Location: https://www.example.com/oauth2#id_token=eyJraWQiOiJrMSIsImFsZyI6IlJTMjU2In0.
eyJpc3MiOiJodHRwczovL3d3dy5wb3J0YWxwcm90ZWN0LmRrIiwic3ViIjoia2ltcmFzIiwiYXVkIjoiaHR0cHM6Ly93d3cuZXhhbXBsZS5jb20vIiwiZX
hwIjoxNDU1MzYxMDI2LCJqdGkiOiIySjRzeU9BNWVaNEZKSDdUbU5TcWZnIiwiaWF0IjoxNDU1MzYwNDI2LCJuYmYiOjE0NTUzNjAzMDYsImdyb3VwcyI6
W10sIm5hbWUiOiJLaW0gUmFzbXVzc2VuIiwiYXV0aF90aW1lIjoiMTQ1NTM2MDIxNiIsIm5vbmNlIjoiMTIzNDUifQ.
Q1KKMPyPlogv08NBh8ArCohBJZr6VvsJ6LysOi2EfuNCkmGA7AtTDJ1ktD-xtzokiyHVGjywbv9xKlQYI4z3wtQhLWMMzUIcwFM-
2lMFZRzo6hujNVd3GruRDSjsnbkpw50veP1xYHf-EfBOuzKjrOIefioKqm0vS6AINNDYS2q0ZB-VnEv2nfwdsdVy8ziwwFQaoD8dgL6DVYP-
gxSLLmn4zINqcgfW86qhiODTc6mvBf_dP-rJ0pk9xnulyw4-
JSCW4EydeqpdVeLQ0pQca7vgudqABhn0uFDqOreBRyTtZKwPAzpLio0zPLG5oHDXTxSZ3r1eRMRGVO4vncN2Q
 Via: 1.1 dispatcher1 (PortalProtect/5.31)
 Content-Length: 0
```

The ID token is now provided directly to the browser in the redirect URL – notice that it is added with # instead of an URL parameter with ? – this ensures that the browser does not send the id_token directly to the server, but it is instead available to javascript code retrieved from the servers redirect URL.

You can verify the contents of the ID token by pasting it into the debugger on http://jwt.io

If you wish both an id_tokan and an access_token, the initial request has to look like this:

```
curl –insecure -s -D - -b cookies.txt -c cookies.txt "https://localhost:4443/oauth2/auth?client_id=https://www.
example.com/&redirect_uri=https://www.example.com/oauth2&response_type=id_token+token&scope=profile+email&nonce=12345"
```

The response after confirming, will then look like this:

```
HTTP/1.1 302 Found
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block
Content-Security-Policy: script-src 'unsafe-inline'
Location: https://www.example.com/oauth2#access_token=eyJraWQiOiJrMSIsImFsZyI6IlJTMjU2In0.
eyJpc3MiOiJodHRwczovL3d3dy5wb3J0YWxwcm90ZWN0LmRrIiwic3ViIjoia2ltcmFzIi
wiYXVkIjoiaHR0cHM6Ly93d3cuZXhhbXBsZS5jb20vIiwiZXhwIjoxNDU1MzY1MTE2LCJqdGkiOiJob1lmOVJqN19zeldWUUtORFlZMUt3IiwiaWF0Ijox
NDU1MzYxNTE2LCJuYmYiOjE0NTUzNjEzOTYsImdyb3
VwcyI6W10sIm5hbWUiOiJLaW0gUmFzbXVzc2VuIn0.eoZJm8Qq9mZ_coiSkyIv5DpZsTZfDnnubN1YIFxWLMDhuFe3ql-oRhtS_JzrIt6Ikwo-3sNz-
WspKa4SPH02paeK7lXoZaohwBj5zkxHN9DZ4DfhMCIZT1
M04ztnbm8985I_CDsK9VrV-dDu0mwQ1pW126TEBA9H8qXBKq2FswsufXLYXU3_0IfyepELwvzR1PAXtv1qHS9qo2ScOIzJPxDF5ZMnwFmKT0Xa39-
d8jeMGqtWKCEBneUd2B5lyLDHHCcnDzFxJ16fKKrBg9KyJ
0qq1CvHPVM3fH-l2JibnZnygE1a-q1hp2mmVPBy22mZ2rU6acrxKwinn9x5eW74Q&id_token=eyJraWQiOiJrMSIsImFsZyI6IlJTMjU2In0.
eyJpc3MiOiJodHRwczovL3d3dy5wb3J0YWxwcm90ZWN0LmRrIi
wic3ViIjoia2ltcmFzIiwiYXVkIjoiaHR0cHM6Ly93d3cuZXhhbXBsZS5jb20vIiwiZXhwIjoxNDU1MzYyMTE2LCJqdGkiOiJqNFV4bTRhS09qQmVNaVMz
eUhVODBRIiwiaWF0IjoxNDU1MzYxNTE2LCJuYmYiOj
E0NTUzNjEzOTYsImdyb3VwcyI6W10sIm5hbWUiOiJLaW0gUmFzbXVzc2VuIiwiYXV0aF90aW1lIjoiMTQ1NTM2MDIxNiIsIm5vbmNlIjoiMTIzNDUiLCJh
dF9oYXNoIjoibERMdTZ5T1NDTi1naTA5Qm9QaF9CQS
J9.T0HpvclYfVzKSoBOpTMsc4tUSZv3a-RrqoNP3Y6Gj7BSZHOzJNyCftwzlv6Vu3xeVztD-nUdxIXm7BNj_CmckMiCSMILdSRm8GZmmk-
IUBiev5styZXi4CfJM8xvWjQVw_JY1FN6Xt_YfataUo5Fnl50RXG91
pq1ruS1Kg5BiZabSGLANey3Pui6MGwtPNgmBWVjNgxsUCYRuBdGpBV6z90ycZBNRX71wXIYF17FSqH_Sfq2aNqidgyBK7X1S1BVhLm7SXWs-
FLZUM0oVIHsCZHzbfkM0l59LbqDmBa1mwE3IphyEqSvS-PF8HDvG
oYVdEvKWl1NcaTh404uORN8eg
Via: 1.1 dispatcher1 (PortalProtect/5.31)
Content-Length: 0
```

It contains both an access token and an id token.

The access_token is rather long, if you modify the configuration, and change **oauth2.client.sample.accesstokentype** from **JWT** to **UUID**, you will get something like this instead:

Request for token:

```
curl –insecure -s -D - -b cookies.txt -c cookies.txt "https://localhost:4443/oauth2/auth?client_id=https://www.
example.com/&redirect_uri=https://www.example.com/oauth2&response_type=id_token&scope=openid+profile+email&nonce=12345
```

Response redirects to confirm_oauth2.jsp, we confirm it

```
curl –insecure -s -D - -b cookies.txt -c cookies.txt "https://localhost:4443/oauth2/confirm"
```

```
HTTP/1.1 302 Found
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block
Content-Security-Policy: script-src 'unsafe-inline'
Location: https://www.example.com
/oauth2#access_token=YjIzNjQ4OTMtYTdmNS00NGQ0LTliMjAtY2E3ZDcwYmIwYTQ4&id_token=eyJraWQiOiJrMSIsImFsZyI6IlJTMjU2In0.
eyJpc3MiOiJo
dHRwczovL3d3dy5wb3J0Ywxwcm90ZWN0LmRrIiwic3ViIjoia2ltcmFzIiwiYXVkIjoiaHR0cHM6Ly93d3cuZXhhbXBsZS5jb20vIiwiZXhwIjoxNDU1Mz
YzMDA2LCJqdGkiOiItcmoxZ1hZckE4ckgwQkhpenhs
eDh3IiwiaWF0IjoxNDU1MzYyNDA2LCJuYmYiOjE0NTUzNjIyODYsImdyb3VwcyI6W10sIm5hbWUiOiJLaW0gUmFzbXVzc2VuIiwiYXV0aF90aW1lIjoiMT
Q1NTM2MDIxNiIsIm5vbmNlIjoiMTIzNDUifQ.VgMty
NP66dVjDNwtOSEpfH6HtdorwidDmMHdu7q2K4WhE1GlTExGepDxdMHwY4MaX70XLxQRjn81oK-BrgQJViRtHN7QciNO1hOfZPjccMBxesTB6LyoL9-
fllvi_sDPIxI0tvNiSLV-liDrZjyb5VGAsvqIAPpXunko5
P6JvqKFQowbjRcfb9UryUH_VUQmoEo5Q5XE0f2KJ9zlQuJuWVCH6aT-txvuP46rKLG5Kc80_f2A6ZjnsQRrIZZuotFMqF_o5q2myKEMrPnMLk-
NKfQbzclTqsW8E4b0c28R4MatPuIU_ex3Zji57N4v7KCYAf53a
nVWyAZWArB859RSHQ
Via: 1.1 dispatcher1 (PortalProtect/5.31)
Content-Length: 0
```

Using our access_token, we request the userinfo

```
curl –insecure -s -D – "https://localhost:4443/oauth2/userinfo" -H "Authorization: Bearer
YjIzNjQ4OTMtYTdmNS00NGQ0LtliMjAtY2E3ZDcwYmIwYTQ4"
```

And we get this back here:

```
HTTP/1.1 200 OK
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block
Content-Security-Policy: script-src 'unsafe-inline'
Cache-Control: no-store
Pragma: no-cache
Content-Type: application/json;charset=UTF-8
Set-Cookie: defaultselectedServer=aM9eMlE3kXkSL6iOdh4qzVV3wkk=; path=/; HttpOnly
Cache-Control: no-cache="Set-Cookie"
Via: 1.1 dispatcher1 (PortalProtect/5.31)
Content-Length: 62

{"sub":"kimras","name":"Kim Rasmussen","email":"kr@asseco.dk"}
```

Note that the configuration of the scopes decided which scope maps to which claims/attributes in the id_token, the access_token and userinfo respectively.

Using Microsoft as an Identity Provider

If you need to use Microsoft as an identity provider, you need to go to: https://apps.dev.microsoft.com/ - here, you can add your application and obtain a client_id for it, and a client_secret to use when exchanging the authorization code for a token.

Then you can use configuration similar to this:

```
<property name="openid.identityproviders" value="microsoft" description="OpenID Connect identity providers, where
we can exchange an authorization code for an id/access token"/>


<property name="openid.idp.microsoft.clientid" value="xxxxx - your client ID xxxxx" description="Client ID"/>
<property name="openid.idp.microsoft.secret" value="xxxx your client secret ----" description=""/>
<property name="openid.idp.microsoft.tokenurl" value="https://login.microsoftonline.com/common/oauth2/v2.0/token"
description="URL to use when exchanging authorization code for a token"/>

<property name="oauth2.tokens" value="sample;sample2;google;microsoft" description="List of OAuth2 JWT tokens
names"/>

<property name="oauth2.token.microsoft.attributesToStoreInSession" value="*" description=""/>
<property name="oauth2.token.microsoft.issuer" value="https://login.microsoftonline.com/9188040d-6c67-4c5b-b112-
36a304b66dad/v2.0" description=""/>
<property name="oauth2.token.microsoft.jceprovider" value="BC" description=""/>
<property name="oauth2.token.microsoft.openidconnect" value="true" description=""/>
<property name="oauth2.token.microsoft.signerCertificates" value="" description=""/>
<property name="oauth2.token.microsoft.signerCertificatesURL" value="https://login.microsoftonline.com/common
/discovery/v2.0/keys" description=""/>
<property name="oauth2.token.microsoft.useridAttributeName" value="preferred_username" description="Name of
attribute to contain the userid, if not sub"/>
<property name="oauth2.token.microsoft.usernameAttributeName" value="name" description="Name of attribute to
contain the username"/>
<property name="oauth2.token.microsoft.validaudiences" value="xxxxx your client ID xxxxx" description="We need to
specify which audience is valid for the token"/>
```

Using Facebook as an Identity Provider

If you need to use facebook as identity provider, go to https://developers.facebook.com/apps/ and register your application - here, you need to obtain the clientid (appid) and secret.

Since facebook does not support OpenID Connect, but its own variant of it, your configuration differs - there is no JWT ID Token, but instead an access token that is sent to facebook, and they return the parsed attributes that you have access to read.
To make using facebook as easy as possible within Ceptor, our JWT authentication plugin supports facebook, but it needs to be configured to know that it should do so like the configuration below:

```
<property name="openid.idp.facebook.clientid" value="xxxx your client ID xxxxx" description="Client ID"/>
<property name="openid.idp.facebook.secret" value="xxxx your client secret xxxx" description=""/>
<property name="openid.idp.facebook.facebook" value="true" description="Use facebook connect, since facebook does
not support OpenID Connect - it prefers its own broken variation"/>
<property name="openid.idp.facebook.facebookfields" value="id,cover,name,first_name,last_name,age_range,link,
gender,locale,picture,timezone,updated_time,verified,email" description=""/>
<property name="openid.idp.facebook.facebookurl" value="https://graph.facebook.com/me" description=""/>
<property name="openid.idp.facebook.tokenurl" value="https://graph.facebook.com/oauth/access_token" description="
URL to use when exchanging authorization code for a token"/>
```

Using LinkedIn as an Identity Provider

If you need to use linkedIn as identity provider, go to https://www.linkedin.com/developer/apps and register your application - here, you need to obtain the clientid and secret.

Since linkedIn does not support the full OpenID Connect specification, your configuration differs - there is no JWT ID Token, but instead an access token that is sent to facebook, and they return the parsed attributes that you have access to read.
To make using linkedIn as easy as possible within Ceptor, our JWT authentication plugin supports linkedIn, but it needs to be configured to know that it should do so like the configuration below:

```
<property name="openid.idp.linkedin.clientid" value="xxxx your client ID xxxx" description=""/>
<property name="openid.idp.linkedin.secret" value="xxxx your client secret xxxx" description=""/>
<property name="openid.idp.linkedin.tokenurl" value="https://www.linkedin.com/oauth/v2/accessToken" description=""
/>
<property name="openid.idp.linkedin.linkedin" value="true" description=""/>
<property name="openid.idp.linkedin.linkedinurl" value="https://api.linkedin.com/v1/people/~" description=""/>
<property name="openid.idp.linkedin.linkedinfields" value="id,firstName,lastName,headline,num-connections,picture-
url,formatted-name,summary,location,public-profile-url" description=""/>
```

You can also configure the Ceptor Gateway to handle the redirection back and forth by creating a location such as this:

```
{
  "content.preload": false,
  "name": "OpenID Connect LinkedIn",
  "description": "OpenID Connect authorization code flow example using LinkedIn",
  "conditions": [{
    "values": [
      "/openid/linkedin",
      "/openid/linkedin/"
    ],
    "type": "path"
  }],
  "authentication": {
    "plugins": ["io.ceptor.authentication.AuthenticatorOpenIDConnect"],
    "openidconnect": {
      "identityprovider.name": "linkedin",
      "authenticationplugin": 48,
      "redirecturl": "https://%{HTTP_HOST}/openid/linkedin",
      "response.name": "Redirect to page after login",
      "scope": "r_basicprofile",
      "authorize.url": "https://www.linkedin.com/oauth/v2/authorization",
      "client.id": "xxxx your client ID xxxx"
    }
  }
}
```

To see more about configuring the gateway, refer to Gateway Configuration and Location - Authentication specifically.

Implementing Datastore Interfaces

There are 3 datastore interfaces; `IOAuthDataStore`, `IAccessTokenDataStore` and `IRefreshTokenDataStore` - these are used for storing OpenID Connect client data, access-token/authorization codes, and for long-term storage of refresh tokens.

Please contact Asseco Support for guidance and examples.

IOAuthDataStore

This datastore is used to retrieve information about clients which are authorized to use Ceptor as an OpenID Provider / Identity Provider.

IOAuthDataStore

```
package dk.itp.security.authentication.oauth.data;

/**
 * Interface to OAuth2 data store - implementers retrieve client ID information,
 */
public interface IOAuthDataStore {
        /**
         * Gives the datastore its configuration
         * @param config Configuration
         */
        public void setConfiguration(Properties config);

        /**
         * Return a list of known client IDs
         * @return List of known client IDs
         */
        public Collection<String> getKnownClientIDs();

        /**
         * Returns known ClientData from a clientID
         *
         * @param clientID Client ID
         * @return Client Data - includes secret, valid scopes, valid redirect URIs etc.
         */
        public ClientData getClientData(String clientID);
}
```

ClientData

```
package dk.itp.security.authentication.oauth.data;

/**
 * Contains known data about an OAuth2 Client.
 * A client is not PortalProtect as a client to someone else, but a client towards PortalProtect where we are the
server
 */
public class ClientData {
        public enum AccessTokenType {
                JWT, UUID
        }

        /** Client ID */
        public String clientID;
        /** Client secret */
        public String clientSecret;
        /** Valid OAuth2 grant type; implicit, authorization_code */
        public Collection<String> validGrantTypes;
        /** Valid scopes, e.g. email, profile, name, photo */
        public Collection<String> allowedScopes;
        /** Allowed redirect URIs */
        public Collection<String> allowedUris;
        /** Allowed logout URIs */
        public Collection<String> allowedLogoutUris;
        /** Number of seconds the access token is valid for */
        public int accessTokenValiditySeconds;
        /** Number of seconds the refresh token is valid for */
        public int refreshTokenValiditySeconds;

        /** The maximum expiration time in minutes allowed */
        public int maximumIdTokenExpirationMinutes;
        /** Name of token configuration to use with this client */
        public String tokenname;
        /** Type of access token to generate */
        public AccessTokenType accessTokenType;


        /** A list of roles for this client - could be roles for an API partner */
        public List<String> roles;

        public Map<String, Object> properties = new LinkedHashMap<>();
}
```

IAccessTokenDataStore

This datastore is used for storage of access tokens, authorization codes, and their related id/refresh tokens.

IAccessTokenDataStore

```
package dk.itp.security.authentication.oauth.data;

/**
 * Interface which must be implemented by data-stores which handle access tokens and authorization codes
 */
public interface IAccessTokenDataStore {
        /**
         * Lookup a previously stored set of information for a particular client ID and access token
         *
         * @param clientId Client ID - might be null if session is created from a ticket/bearer token
         * @param refreshToken Refresh Token
         * @return RefreshTokenInfo loaded from datastore.
         */
        public AccessTokenInfo lookupToken(String clientId, String accessToken);

        /**
         * Lookup a previously stored set of information for a particular client ID and authorization code
         * An access token can only be looked up once - after that, it is marked as being used and cannot be
reused.
         *
         * @param clientId Client ID
         * @param refreshToken Refresh Token
         * @return RefreshTokenInfo loaded from datastore.
         */
        public AccessTokenInfo lookupTokenFromCode(String clientId, String authorizationCode);

        /**
         * Store the info in the data store.
         *
         * @param info Access Token to store
         */
        public void storeAccesToken(AccessTokenInfo info);

        /**
         * Revoke an existing access token
         *
         * @param info Access Token to revoke
         */
        public void revokeAccessToken(AccessTokenInfo info);

        /**
         * Called to initialize this datastore, after configuration has been set
         * @param sessioncontroller
         */
        public void start(PTSServer sessioncontroller);

        /**
         * Gives this datastore a chance to stop and release any resources it might have
         */
        public void stop();

        /**
         * Gives the datastore its configuration
         * @param config Configuration
         */
        public void setConfiguration(Properties config);
}
```

AccessTokenInfo

```
package dk.itp.security.authentication.oauth.data;

/**
 * Contains information about a refresh token
 */
public class AccessTokenInfo implements Serializable {
        private static final long serialVersionUID = 1L;

        /** The access token */
        public String access_token;
        /** An ID token that might have be generated along with the access token */
        public String id_token;
        /** Client ID */
        public String clientid;
        /** Redirect URL */
        public String redirectUri;
        /** Userinfo generated from the session */
        public String userinfo;
        /** Refresh token, if any */
        public String refreshToken;
        /** Expiration time in seconds of the access token*/
        public int accessTokenExpireSeconds;
        /** Authorization code which can be used to obtain this access token */
        public String authorizationCode;

        // --- Information needed for introspection ---
        /** The requested scope */
        public String scope;
        /** Expires at (seconds) */
        public long expiresAt;
        /** Issued at */
        public long issuedAt;
        /** Subject */
        public String sub;
        /** Username */
        public String username;
        /** Audience */
        public String audience;
        /** Issuer */
        public String issuer;
        /** Token ID */
        public String jti;
}
```

IRefreshTokenDataStore

This datastore is used for long-term storage of refresh tokens.

IRefreshTokenDataStore

```
package dk.itp.security.authentication.oauth.data;

/**
 * Interface which must be implemented by data-stores which handle refresh tokens
 */
public interface IRefreshTokenDataStore {
        /**
         * Lookup a previously stored set of session data for a particular client ID and refreshToken
         *
         * @param clientId Client ID
         * @param refreshToken Refresh Token
         * @return RefreshTokenInfo loaded from datastore.
         */
        public RefreshTokenInfo lookupToken(String clientId, String refreshToken);

        /**
         * Store the info in the data store.
         *
         * @param info Refresh Token to store
         */
        public void storeRefreshToken(RefreshTokenInfo info);

        /**
         * Revoke an existing refresh token
         *
         * @param info Refresh token to revoke
         */
        public void revokeRefreshToken(RefreshTokenInfo info);

        /**
         * Gives the datastore its configuration
         * @param config Configuration
         */
        public void setConfiguration(Properties config);

        /**
         * Called to initialize this datastore, after configuration has been set
         * @param sessioncontroller
         */
        public void start(PTSServer sessioncontroller);

        /**
         * Gives this datastore a chance to stop and release any resources it might have
         */
        public void stop();
}
```

RefreshTokenInfo

```
package dk.itp.security.authentication.oauth.data;

/**
 * Contains information about a refresh token
 */
public class RefreshTokenInfo implements Serializable {
        private static final long serialVersionUID = 1L;

        /** The refresh token */
        public String refreshToken;
        /** The client ID this token is valid for */
        public String clientID;
        /** Expire time in msecs */
        public long expiresAt;
        /** Created at in msecs */
        public long createdAt;

        public byte[] persistedSession;
}
```