IBM z/OS Connect EE V3.0

Customization-Security and JWT Tokens

IBM Z
Wildfire Team –
Washington System Center

Lab Version Date: December 10, 2020

Table of Contents

Important: On the desktop there is a file named *Security CopyPaste.txt*. This file contains commands and other text used in this workshop. Locate that file and open it. Use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors. As a reminder text that appears in this file will be highlighted in yellow.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

General Exercise Information and Guidelines

- ✓ This exercise requires the completion of the *zCEE Basic Configuration* and *zCEE Basic Security Configurations* exercises before it can be performed.

- ✓ There are examples of server.xml scattered through this exercise. Your server.xml may differ depending on which exercises have been previously performed. Be sure the red lines in these examples are either added or already present.

- ✓ The acronyms RACF (resource access control facility) and SAF (system authorization facility) are used in this exercise. RACF is the IBM security manager product whereas SAF is a generic term for any security manager product, e.g. ACF2 or Top Secret or RACF. An attempt has been to use SAF when referring to information appropriate for any SAF product and to use RACF when referring to specific RACF commands or examples.

- ✓ Any time you have any questions about the use of IBM z/OS Explorer, Postman, 3270 screens, features or tools do not hesitate to ask for assistance.

- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *Security CopyPaste* file on the desktop.

- ✓ Please note that there may be minor differences between the screen shots in this exercise versus what you see when performing this exercise. These differences should not impact the completion of this exercise.

© Copyright IBM Corporation 2020 All rights reserved

Overview

This exercise demonstrates the use of OAuth 2.0 and OpenID Connect 1.0 for security from a REST client to an API hosted in a z/OS Connect server (API Provider) and from a COBOL application using z/OS Connect as a REST client to access a remote API (API Requester). Both cases (provider and requester) will be using a JSON Web Token (JWT) for security.

This exercise uses Liberty as the OAuth 2.0 authorization server. The OpenID Connect (OIDC) client support will be configured or provided either by Liberty configuration elements (API provider) or by z/OS Connect configuration elements (API Requester). Note, there are scores of products which have been certified for use as an OAuth authorization server, see URL https://openid.net/certification. The exercise does not attempt to provide an all-encompassing description of every permutation of token options. The point to remember is that regardless of which OAuth authorization server is being used, the fundamental concepts and terminology used in this exercise are still relevant.

The first part one of the exercise provides a basic introduction to OAuth 2.0 and OpenID Connect 1.0. Terms will be introduced and basic flows between a client and an OAuth authorization server will be described.

In part two of the exercise, a sample web application will be used to demonstrate the flow of a JWT token from a client when accessing a z/OS Connect API provider server. The required Liberty server XML configuration where the z/OS Connection is executing will be described, and tests will be performed to demonstrate the implications of security.

In the final part of the exercise, the use of OpenID Connect will be explored for an z/OS Connect API requester application. A simple API Requester COBOL application will be used to demonstrate the behavior of a z/OS Connect API requester application using JWT tokens for security when accessing an API. The required z/OS Connect server XML configuration be described, and again a few tests will be performed to demonstrate the presence of security.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

OAuth and OpenID Client

Let's begin by introducing definitions, terms and concepts. The information in this section was derived directly from the specifications for *OAuth 2.0* and *OpenID Connect Core 1.0*, see URLs https://tools.ietf.org/html/rfc6749 and https://openid.net/specs/openid-connect-core-1_0.html or from the Liberty and/or the z/OS Connect Knowledge Centers.

• *The OAuth 2.0 Authorization Framework [RFC6749] and OAuth 2.0 Bearer Token Usage [RFC6750] specifications provide a general framework for third-party applications to obtain and use limited access to HTTP resources. They define mechanisms to obtain and use Access Tokens to access resources but do not define standard methods to provide identity information.*

• *OpenID Connect 1.0 implements authentication as an extension to the OAuth 2.0 authorization process. Use of this extension is requested by Clients by including the openid scope value in the Authorization Request. Information about the authentication performed is returned in a JSON Web Token (JWT) called an ID Token OAuth 2.0 Authentication Servers implementing OpenID Connect are also referred to as OpenID Providers (OPs). OAuth 2.0 Clients using OpenID Connect are also referred to as Relying Parties (RPs)*

• *z/OS Connect EE security can operate with traditional z/OS security, for example, System Authorization Facility (SAF) and also with open standards such as Transport Layer Security (TLS), JSON Web Token (JWT), and OpenID Connect (OIDC).*

Some basic OAuth 2.0/OpenID Connect 1.0 terms

• *Access Token* – A credential that is used to access protected resources. An access token is a string that represents an authorization that is issued to the client. The access token is usually opaque to the client (it does not have to be opaque). See URL https://tools.ietf.org/html/rfc6749 Section 1.4 for more information.

• *Authorization Endpoint* - A service or endpoint on an OAuth authorization server that accepts an authorization request from a client to perform authentication and authorization of a user. The authorization endpoint returns an authorization grant, or code, to the client in the Authorization Code Flow. In the Implicit Flow, the authorization endpoint returns an access token to the client.

• *Authorization server* - The server that issues access tokens to the client after authenticating the resource owner and obtaining authorization. *In a z/OS Connect EE API requester scenario, the authorization server is called by the z/OS Connect EE server to retrieve an access token.*

• *Client* - An application that makes protected resource requests on behalf of the resource owner and with its authorization. *In a z/OS Connect EE API requester scenario, the client is a combination of the CICS, IMS, or z/OS application **and** the z/OS Connect EE server that calls the RESTful API on behalf of the CICS, IMS, or z/OS application.*

• *ID Token* – is an OpenID Connect token that is an extension to OAuth 2.0 specification access tokens. This token is a JSON Web Token (JWT). See URL

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

https://openid.net/specs/openid-connect-core-1_0.html#IDToken for more information about the extensions.

• *OpenID Connect Provider (OP)* - An OAuth 2.0 authorization server that is capable of providing claims to a client or Relying Party (RP) by using an ID token and an access token.

• *Relying Party (RP)* - An entity that relies on an OAuth authorization server to authenticate a user and obtain an authorization to access a user's resource. *For z/OS Connect it is either a Liberty server configured as an OpenID Connect Client, e.g. z/OS Connect API Provider or a client application that requires claims from an OpenID Provider (OP), e.g. z/OS Connect API Requester.*

• *Resource owner* - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user. *In a z/OS Connect EE API requester scenario, the resource owner might be the **user** of the CICS, IMS, or z/OS application.*

• *Resource server* - The server that hosts the protected resources and accepts and responds to protected resource requests by using access tokens. *In a z/OS Connect EE API requester scenario, the resource server is the **request endpoint** for the remote RESTful API*

• *Scope* - Privilege or permission that allows access to a set of resources of a third party. The OpenID Connect Core specification indicates that a scope parameter must be present in an authentication request and contains the *openid* scope value. (If no *openid* scope value is present, the request may still be a valid OAuth 2.0 request but is not an OpenID Connect request.)

• *Token Endpoint* - A resource (URL) on an OP that accepts an authorization grant, or code, from a client in exchange for an access token, ID token, and refresh token

• *OAuth token* - With OAuth 2.0, access tokens are used to access protected resources. An access token is normally a string that represents an authorization that is issued to the client. The string is usually opaque to the client. Opaque tokens require that the token recipient call back to the server that issued the token. *However, an access token can also be in the form of a JSON Web Token (JWT) which does not require a call back (introspection).*

In this document, when we refer to *OAuth* authorization server, we are referring to the authorization server that authenticates a client's credentials and provides an access token to the client for use for accessing resources in a RESTful manner. The format of this access token can be either opaque or in a JWT format. This document concentrates on JWT tokens.

When the OAuth authorization server is configured to provide an ID token to a token request containing the *openid* scope, this server will be referred to an *OpenID Connect Provider (OP).*

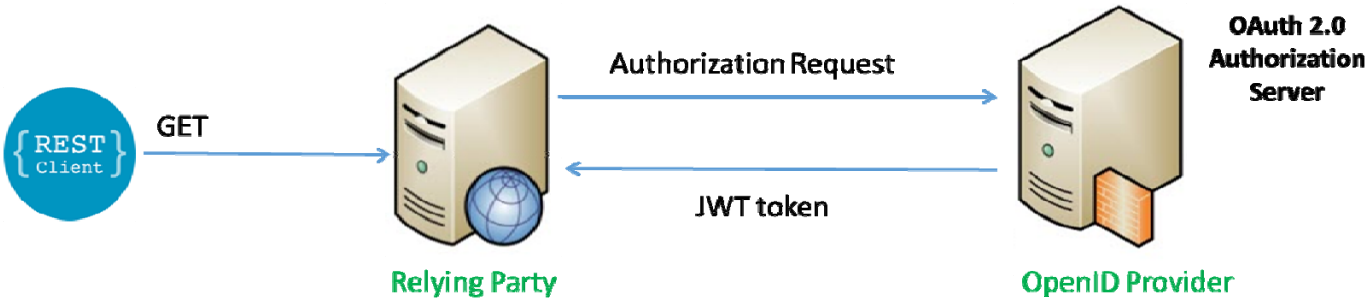
When we refer to *OpenID Connect*, we are referring the interaction from a client using a *Relying Party* (RP) to the authorization server (or *OP*).

• For a very good explanations on these terms and topics see these YouTube videos.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

- YouTube video *OAuth 2.0 and OpenID Connect (in plain English)*
  https://www.youtube.com/watch?v=996OiexHze0
- YouTube video OpenID Connect on Liberty
  https://www.youtube.com/watch?v=fuajCS5bG4c

The diagram below shows at a high level the flow to an authorization server for obtaining a token(some of the details of the back and forth flows between the RP and OP are omitted).



The REST client initiates a request to the Relying Party (RP). The RP can be an application running in Liberty or an entity running in the network. The RP sends an authorization request with client credentials to the OpenID Provider(OP). The credentials are authenticated and a JSON Web Token (JWT) is sent back to the RP for future use. z/OS Connect would appear in the above flow as a relying party in an API requester scenario where the REST client would be the COBOL application.

Next, we will explore this flow using a Liberty server as the OpenID Provider and another Liberty server as the Relying Party.

OAuth and z/OS Connect API Provider

In this section we will be using a sample application written by Edward McCarthy, IBM Australia. Edward McCarthy's application has been adapted for this exercise. This application is a simple web application that when accessed from a web browser obtains a JWT token from an OP server and displays the details in the browser. The Liberty server in which the application executes is the RP server.

OAuth/OpenID Connect Sample Application

Before the web application is invoked, explore the Liberty server's server XML configuration files.

© Copyright IBM Corporation 2020 All rights reserved

Relying Party (RP) - WLPRPSRV

Below is snippet of the server XML configuration file for the Liberty Relying Party server WLPPRSRV.

```
<httpEndpoint host="*" httpPort="26222" httpsPort="26223" id="defaultHttpEndpoint"/>

<!-- Auth Filter used to match access to our application over SSL -->
<authFilter id="jwtIvpSslAuthFilter">      1
  <requestUrl id="jwtIvpSslDemo"
      urlPattern="/CallRestApiWithJwtDemoWeb/ssl/InvokeRestWithJwt"
      matchType="contains" />
</authFilter>

<!-- Define OIDC Client called RP -->
<openidConnectClient id="RPssl"      2
  authFilterRef="jwtIvpSslAuthFilter" 3
  grantType="implicit"
  httpsRequired="true"
  scope="openid profile email photo"
  clientId="rpSsl"    4
  redirectToRPHostAndPort="https://wg31.washington.ibm.com:26223"
  authorizationEndpointUrl="https://wg31.washington.ibm.com:26213/oidc/endpoint/OP/authorize" 5
  issuerIdentifier="https://wg31.washington.ibm.com:26213/oidc/endpoint/OP" 6
  signatureAlgorithm="RS256"    7
  trustAliasName="JWT-Signer-Certificate"    8
  trustStoreRef="jwtTrustStore"     9
  authnSessionDisabled="true"
  disableLtpaCookie="true">
</openidConnectClient>

<!-- Key store that contains certificate used to sign JWT -->
<keyStore fileBased="false" id="jwtStore"     10
  location="safkeyring:///JWT.KeyRing"
  password="password" readOnly="true" type="JCERACFKS"/>
```

Notes:

1. The *authFilter* configuration element is used to select which inbound HTTP request will be processed by this security configuration. In this case, any inbound HTTP request containing the value */CallRestApiWithJwtDemoWeb/ssl/InvokeRestWithJwt* will be processed.
2. The *openidConnectClient* configuration element provides the connection properties for the OP server.
3. The *authFilterRef* attribute is used to relate the *authFilter jwtIvpSslAuthFilter* with this *openidConnectClient* element.
4. The *clientID* attribute is used to select the OP *localStore* which will handle authorization.
5. The *authorizationEndpoint* attribute provides the URL for the OP server authorization endpoint. For Liberty, this value is determined by the name of the OP defined in the Liberty OP server, see URL https://www.ibm.com/support/knowledgecenter/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/twlp_oidc_auth_endpoint.html
6. The *issuerIdentifier* attribute is the case-sensitive URL using the HTTPS scheme that contains the scheme, host and optionally port name and path components of the OP server.
7. The *signatureAlgorithm* attribute is used to verify the JWT token and must match the value define in the OP server.
8. The *trustAliasName* attribute is the label of a certificate in the designated key store. This certificate is used to validate the certificate embedded in the JWT token.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

9. The *trustStoreRef* attribute identity the key store which contains the certificate identified by the *trustAliasName* attribute.
10. The *keystore* attribute identifies RACF key ring which contains the certificate used to verify the JWT token.

OpenID Connect Provider (OP) - WLPOPSRV

Below is snippet of the server XML configuration file for the Liberty OP server WLPOPSRV.

```
<httpEndpoint host="*" httpPort="26212" httpsPort="26213" id="defaultHttpEndpoint"/>

<!-- Define OpenID Connect Provider called OP  -->
<openidConnectProvider id="OP"   1
  signatureAlgorithm="RS256"
  keyStoreRef="jwtStore"
  oauthProviderRef="OIDCssl" >
</openidConnectProvider>

<oauthProvider id="OIDCssl"     2
  httpsRequired="true"
  jwtAccessToken="true"
  autoAuthorize ="true"
  accessTokenLifetime="300">
  <autoAuthorizeClient>rpSsl</autoAuthorizeClient>
   <localStore>                          3
     <client name="rpSsl"
     displayname="rpSsl"
     grantType="implicit"
     redirect="https://wg31.washington.ibm.com:26223/oidcclient/redirect/RPssl"  4
     scope="openid profile scope1 email phone address"
     enabled="true"
     resourceIds="myZcee"/>
    </localStore>
    <localStore>
     <client name="zCEEClient" . . ./>
    </localStore>
</oauthProvider>

<!-- Key store that contains certificate used to sign JWT -->
<keyStore fileBased="false" id="jwtStore"           5
  location="safkeyring:///JWT.KeyRing"
  password="password" readOnly="true" type="JCERACFKS"/>

<!-- Define a basic user registry -->
<basicRegistry id="basicRegistry" realm="zCEERealm" 6
  <user name="auser" password="pwd"/>
  <user name="Fred" password="fredpwd"/>
  <user name="distuser1" password="pwd"/>
  <user name="distuser2" password="pwd"/>
</basicRegistry>
```

Notes:

1. The *openidConnectProvider* configuration element is the initial configuration element for adding OAuth authorization server support to this Liberty server. The attribute specified here identify the signature algorithm used to verify an ID token. The keystore containing the self-signed certificate that will be embedded into the token and a reference to a corresponding *oauthProvider* configuration element.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

2. The *oauthProvider* attribute is referenced by the *openidConnectProvider* configuration element. This configuration element defines the type of token to be generated (*jwtAccessToken=”true”*). The period of time for which the token will be valid (*accessTokenLifetime=”300”*).
3. The *localStore* configuration element provides configuration details for specific clients. In this case the RP will specify this *client name* when the authorization request is sent. The *scope* value of the client’s request will be validated with the scopes defined for this *localStore*.
4. The *OP* will use the value of the *redirect* URL to confirm the authenticity of the client.
5. This identifies the local SAF key ring contain the self-signed certificate which will be embedded in the JWT token.
6. This is the local registry used to authentication user identities and passwords. This Liberty server is configured to used basic authentication with a basic internal registry. RACF identity FRED is a member of the global administration group and USER1 is a member of the global invoke group. USER2 has no access to z/OS Connect.

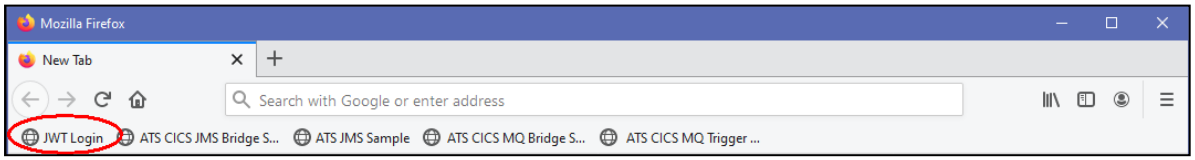
© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

Use the web application to obtain a JWT

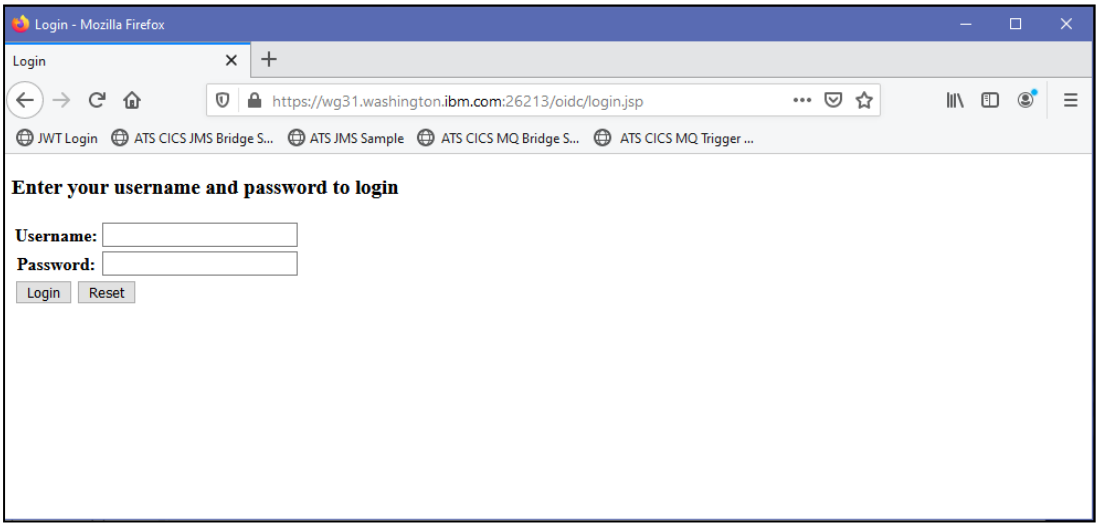
In this section the web application will be used to generate JWT tokens.

___1. Start a Firefox browser session and enter this URL, ***https://wg31.washington.ibm.com:26223/CallRestApiWithJwtDemoWeb/ssl/InvokeRestWithJwt***

or use the *JWT Login* bookmark on the *Toolbar* (see below).



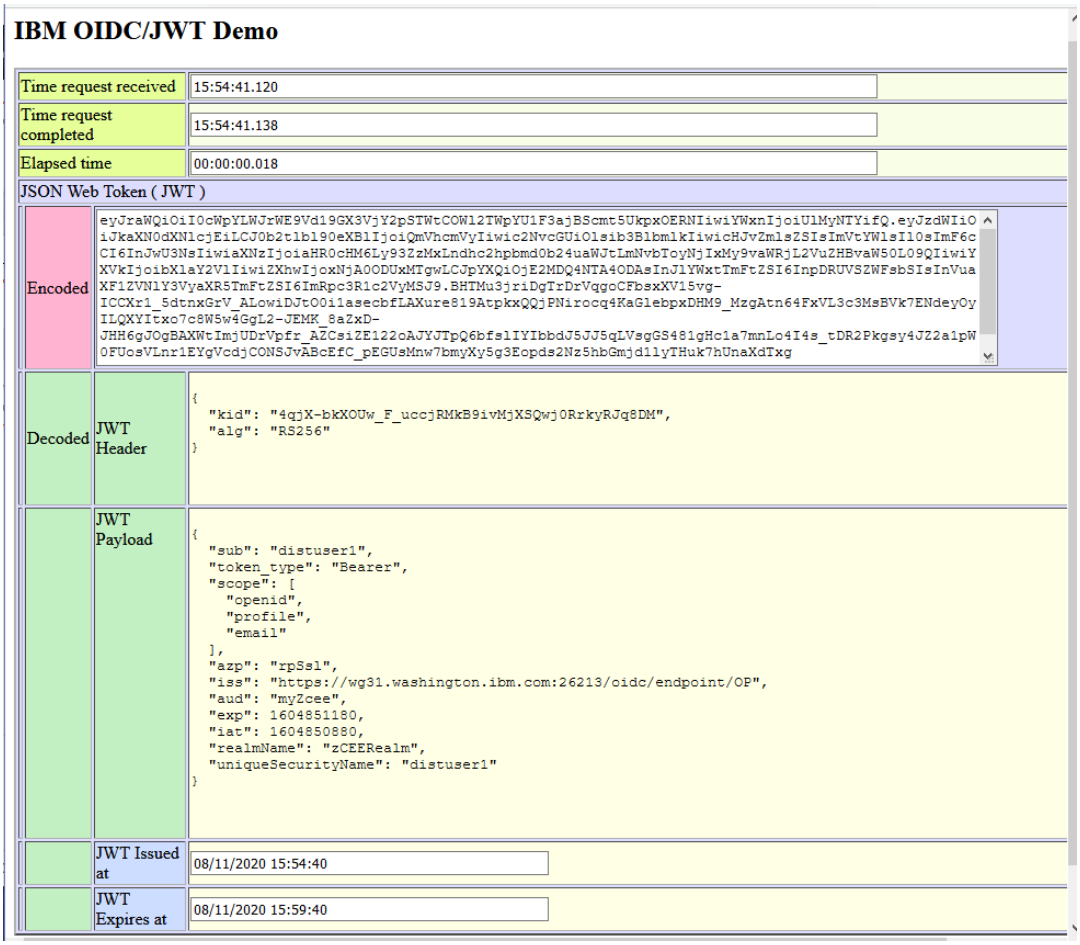
___2. A login screen will be displayed. Enter ***distuser1*** in the area beside *Username* and ***pwd*** in the area beside *Password*.



Tech-Tip: Note that based on the port, this pop-up is from the OP server.

© Copyright IBM Corporation 2020 All rights reserved

___3. Press the **Login** button to continue. The request will flow back to the *OP* server and a JWT token will be generated. When the request flows back to the *RP* server, the JWT token will be displayed in the web browser as shown below:



IBM OIDC/JWT Demo

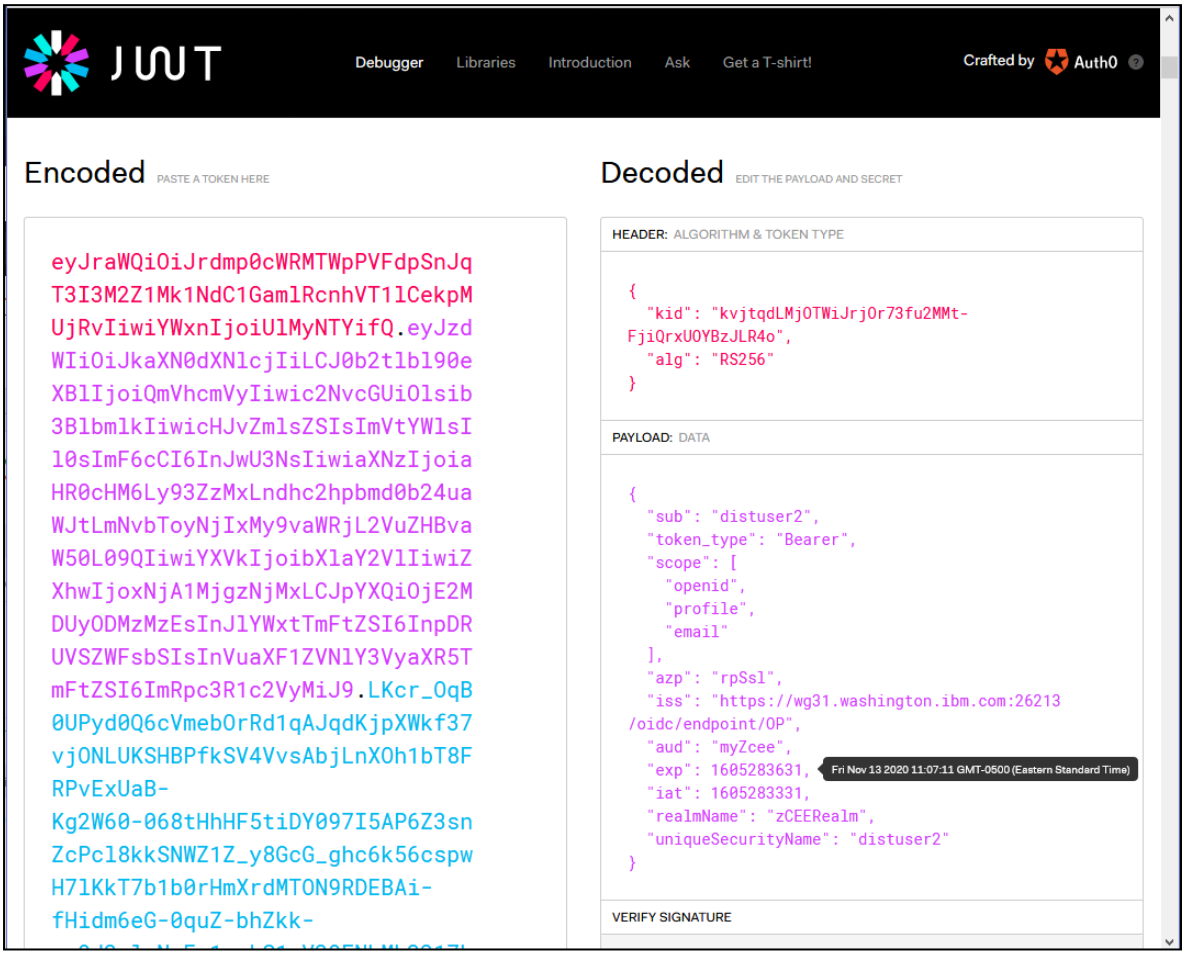
Time request received		15:54:41.120
Time request completed		15:54:41.138
Elapsed time		00:00:00.018
JSON Web Token (JWT)		
Encoded		eyJraWQiOiI0cWpYLWJrWE9Vd19GX3VjY2pSTWtCOWl2TWpYU1F3ajBScmt5UkpxOERNIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJkaXN0dXNlcjEiLCJ0b2tlbl90eXBlIjoiQmVhcmVyIiwic2NvcGUiOlsib3BlbmlkIiwicHJvZmlsZSIsImVtYWlsIl0sImF6cCI6InJwU3NsIiwiaXNzIjoiaHR0cHM6Ly93ZzMxLndhc2hpbmd0b24uaWJtLmNvbToyNjIxMy9vaWRjL2VuZHBvaW50L09QIiwiYXVkIjoibXlaY2VlIiwiZXhwIjoxNjA0ODUxMTgwLCJpYXQiOjE2MDQ4NTA4ODAsInJlYWxtTmFtZSI6InpDRUVSZWFsbSIsInVuaXF1ZVNlY3VyaXR5TmFtZSI6ImRpc3R1c2VyMSJ9.BHTMu3jriDgTrDrVqgoCFbsxXV15vg-ICCXr1_5dtnxGrV_ALowiDJtO0i1asecbfLAXure819AtpkxQQjPNirocq4KaGlebpxDHM9_MzgAtn64FxVL3c3MsBVk7ENdeyOyILQXYItxo7c8W5w4GgL2-JEMK_8aZxD-JHH6gJOgBAXWtImjUDrVpfr_AZCsiZE122oAJYJTpQ6bfslIYIbbdJ5JJ5qLVsgGS481gHc1a7mnLo4I4s_tDR2Pkgsy4JZ2a1pW0FUosVLnr1EYgVcdjCONSJvABcEfC_pEGUsMnw7bmyXy5g3Eopds2Nz5hbGmjd1lyTHuk7hUnaXdTxg
Decoded	JWT Header	{ "kid": "4qjX-bkXOUw_F_uccjRMkB9ivMjXSQwj0RrkyRJq8DM", "alg": "RS256" }
	JWT Payload	{ "sub": "distuser1", "token_type": "Bearer", "scope": ["openid", "profile", "email"], "azp": "rpSsl", "iss": "https://wg31.washington.ibm.com:26213/oidc/endpoint/OP", "aud": "myZcee", "exp": 1604851180, "iat": 1604850880, "realmName": "zCEERealm", "uniqueSecurityName": "distuser1" }
	JWT Issued at	08/11/2020 15:54:40
	JWT Expires at	08/11/2020 15:59:40

___4. Copy the contents of the *Encoded* box to the clipboard and open a new tab in the Firefox browser and go to URL https://jwt.io.

Tech-Tip: When copying a token from the web browser be sure that no leading or trailing spaces are included.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___5. Scroll on the new session and replace the contents of the box labeled *Encoded* with the encoded token in the clipboard. The results should look the example displayed below.



The contents in box labeled *Decoded* should be the same as what is display in the original Firefox session. The use of the https://jwt.io to decode a JWT token is useful because by hovering (see above) over certain fields in the *Decoded* Box, either an explanation or description of the field or the decoded value, e.g. expiration and/or issue time will be displayed.

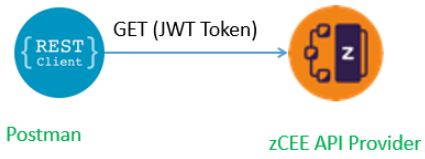
Next use the token that has been copied to the clipboard with *Postman* as a security token to access and z/OS Connect API.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

Using Postman to explore JWT security

In this section *Postman* will be used to send a **GET** request to a z/OS Connect API. Authentication will be done by sending a *Bearer* token containing the token copied to the clipboard in the previous section.

The flow in this section is display below.



The z/OS API provider server (ZCEEOPID) contains the server XML configuration below. In this scenario when an inbound API request is received where the path contains */cscvinc/employee*, this *openidConnectClient* configuration will be used to verify the JWT token embedded in the request. Once the JWT token is verified, the value of the *sub* field will be used for authentication.

```
<httpEndpoint host="*" httpPort="9100" httpsPort="9463" id="defaultHttpEndpoint"/>

<authFilter id="ATSAuthFilter">    1
   <requestUrl  id="ATSDemoUrl"
     name="ATSRefererUri" matchType="contains"
        urlPattern="/cscvinc/employee" />
 </authFilter>

 <!-- Define OIDC Client called ATS -->
 <openidConnectClient id="ATS"           2
      httpsRequired="true"
      authFilterRef="ATSAuthFilter"      3
      inboundPropagation="required"
      audiences="myZcee"         4
      issuerIdentifier="https://wg31.washington.ibm.com:26213/oidc/endpoint/OP" 5
      signatureAlgorithm="RS256"    6
      userIdentityToCreateSubject="sub"         7
      trustAliasName="JWT-Signer-Certificate"   8
      trustStoreRef="jwtTrustStore"     9
      authnSessionDisabled="true"
      disableLtpaCookie="true">
  </openidConnectClient>

 <keyStore fileBased="false" id="jwtTrustStore" 10
     location="safkeyring:///JWT.KeyRing"
      password="password" readOnly="true" type="JCERACFKS"/>
```


© Copyright IBM Corporation 2020 All rights reserved

Notes:
1. The *authFilter* configuration element is used to select which inbound HTTP request will be processed by this security configuration. In this case, any inbound HTTP request containing the value */cscvinc/emloyee* will be processed.
2. The *openidConnectClient* configuration element provides the connection properties for the OP server.
3. The *authFilterRef* attribute is used to relate the *authFilter jwtIvpSslAuthFilter* with this *openidConnectClient* element.
4. The *audience* attribute identifies the intended recipient of this token.
5. The *issuerIdentifier* attribute is the case-sensitive URL using the HTTPS scheme that contains the scheme, host and optionally port name and path components of the OP server.
6. The *signatureAlgorithm* attribute is used to verify the JWT token and must match the value define in the OP server.
7. The *userIdentityToCreateSubject* attribute identifies the field in the payload which identifies the subject or to whom the token refers, i.e. the authenticated user identity.
8. The *trustAliasName* attribute is the label of a certificate in the designated key store. This certificate is used to validate the certificate embedded in the JWT token.
9. The *trustStoreRef* attribute identity the key store which contains the certificate identified by the *trustAliasName* attribute.
10. The *keystore* attribute identifies RACF key ring which contains the certificate used to verify the JWT token.

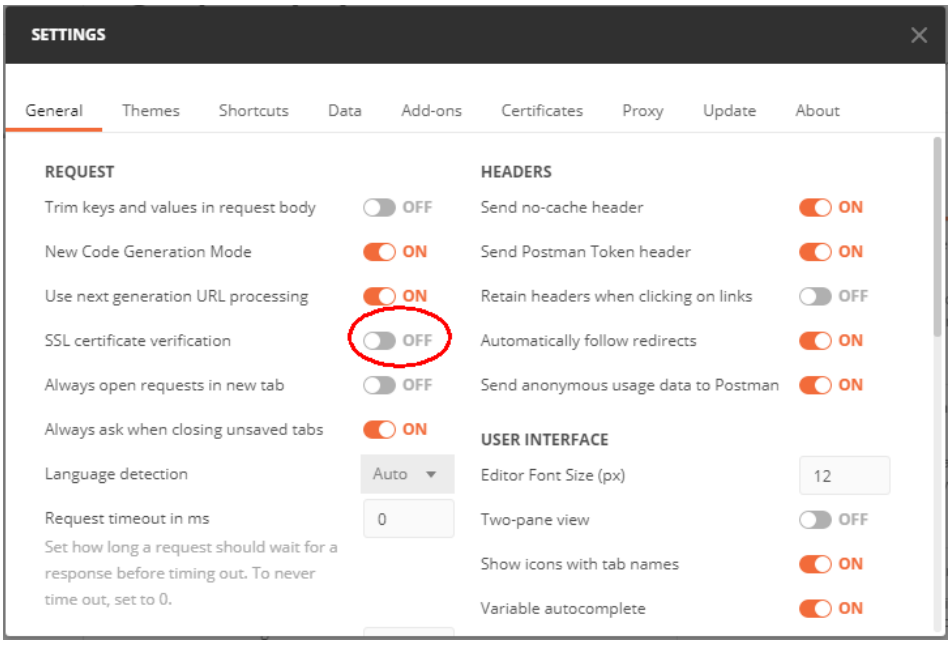
___1. Before any testing can be performed, mapping between the distributed or non z/OS identities and RACF or SAF identities must be done. Submit member ***JWTMACMP*** in data set *USER1.ZCEE30.CNTL*. The commands in this member map the distributed identities in the basic registry of the OP provider with local RACF identities. Note that the distributed identity *auser* is not mapped. This will be important later.

```
RACMAP ID(FRED)  MAP USERDIDFILTER(NAME('Fred')) +
  REGISTRY(NAME('*')) WITHLABEL('zCEE JWT FRED')
RACMAP ID(USER1)  MAP USERDIDFILTER(NAME('distuser1')) +
  REGISTRY(NAME('*')) WITHLABEL('zCEE JWT distuser1')
RACMAP ID(USER2)  MAP USERDIDFILTER(NAME('distuser2')) +
  REGISTRY(NAME('*')) WITHLABEL('zCEE JWT distuser2')

SETROPTS RACLIST(IDIDMAP) REFRESH
```

Tech-Tip: Be aware that Liberty configuration *safCredentials* element has a required attribute *mapDistributedIdentities* which enables the mapping of distributed identities to RACF identities for the entire server. At the same time, the Liberty configuration *openidConnectClient* element has an attribute *mapIdentityToRegistryUser* which defaults to *false*. In this exercise we have done the above explicit mappings to map distributed identities to SAF identities. If these had been set to true, Liberty would use the distributed identity as is and done the mapping. In this scenario, the distributed identity Fred would have been valid since the distributed identity matched the SAF identity. Using the other distributed identities would have generated RACF error since they are not valid RACF identities.

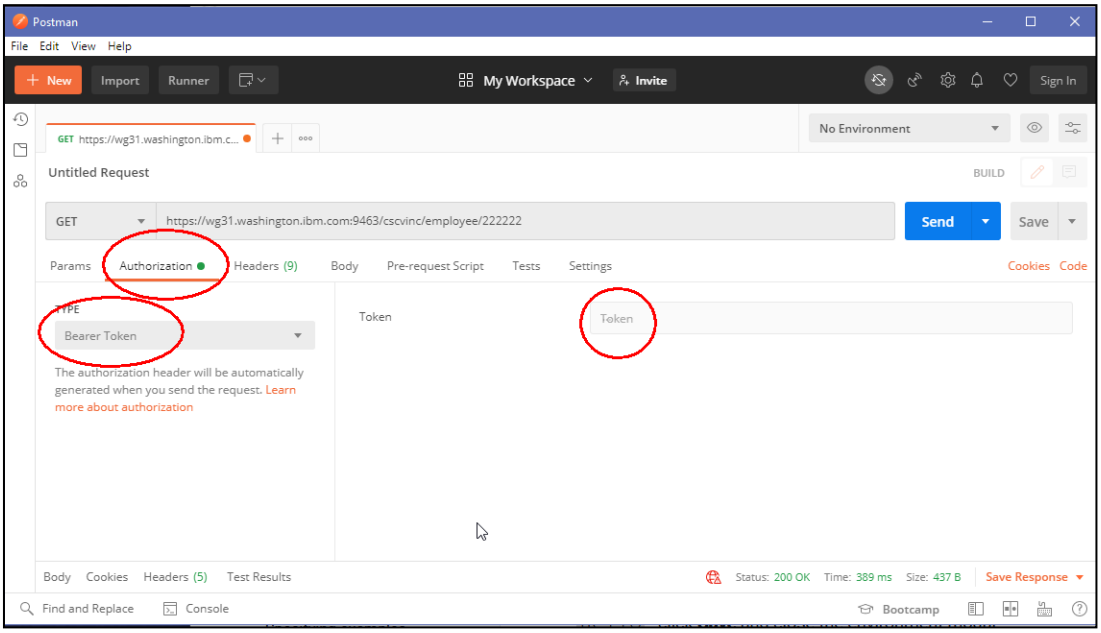
© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___2. Start a Postman session and select the settings icon (the wrench on the tool bar) to open the *Settings* window.



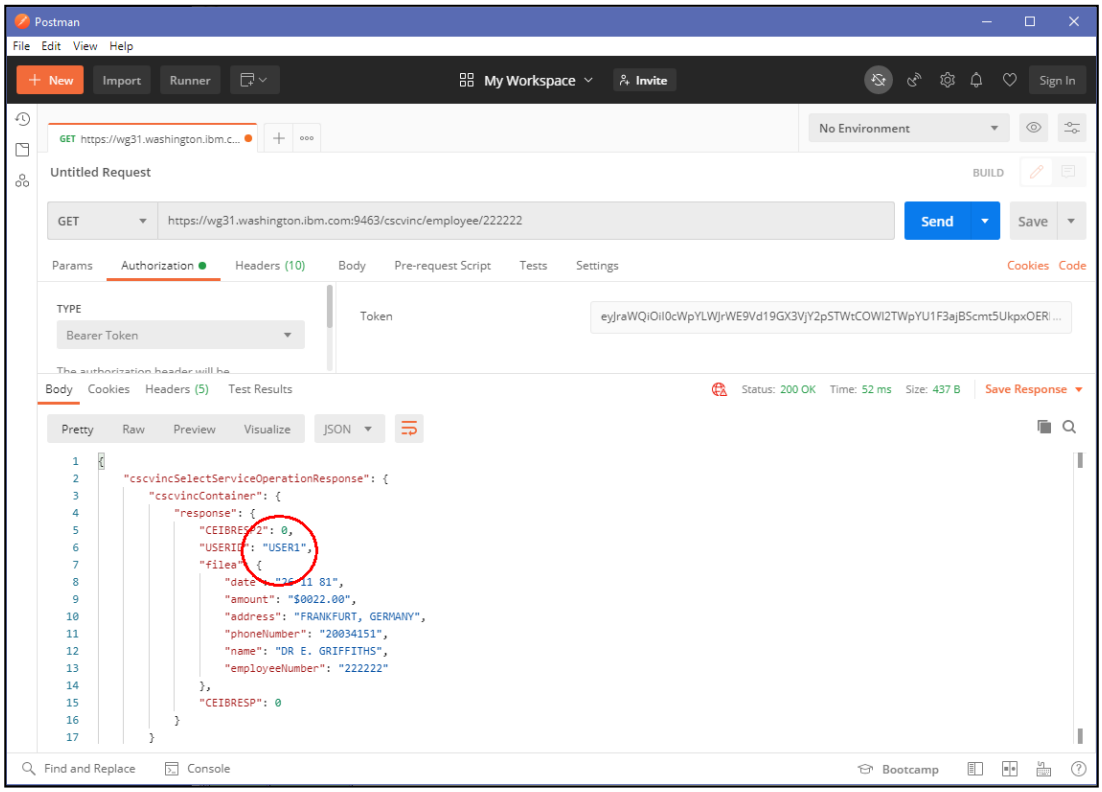
___3. In the *General* tab, ensure *SSL certificate verification* is turned off (see above).

___4. Go to the Authorization tab and use the pull-down arrow to select *Bearer Token*. In the area beside *Token*, paste the JWT token copied from the Firebox browser session. Select GET for the method and enter URL ***https://wg31.washington.ibm.com:9463/cscvinc/employee/222222*** as the URL. Press the **SEND** button to continue.



© Copyright IBM Corporation 2020 All rights reserved

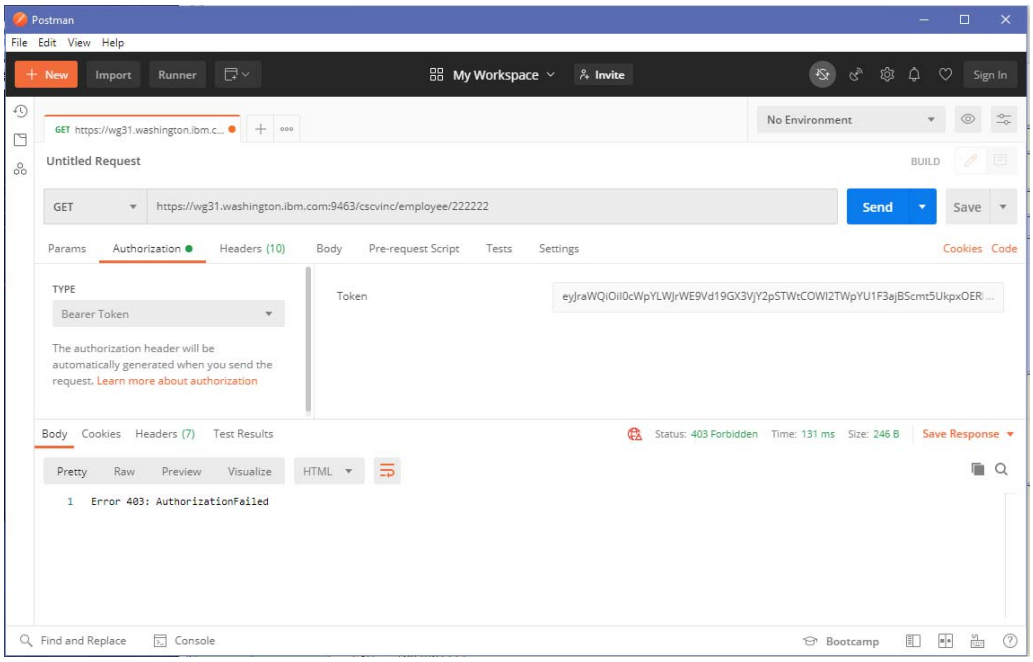
___5. The results should look something like this:



The JWT token has been sent to the Liberty server in which z/OS Connect is running. The token was verified based on the *openidConnectClient* configuration element. The value of the *sub* field, *distuser1* was extracted and mapped to RACF identity *USER1*. This identity was propagated to CICS (the API invoked a CICS program) and returned in the response container as the value of the *USERID* field.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___6. Optional, wait approximately 5 minutes (*accessTokenLifetime="300")* and the same request will fail with a HTTP status of *403 Forbidden* and Error *403, Authorization failed*. This is because the token has expired.

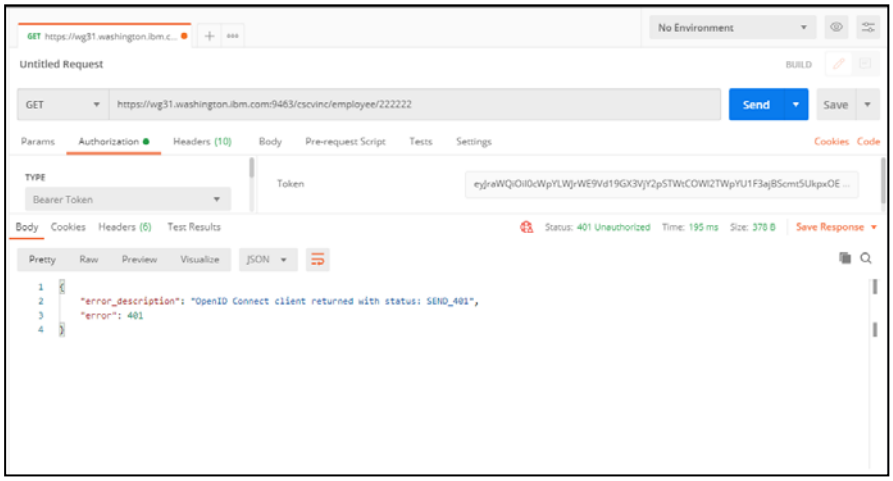


___7. Next test with a distributed identity which does not have access to z/OS Connect. Close all open Firefox browser session and obtain JWT token for distributed identity *distuser2* (mapped to RACF identity *USER2*).

Tech-Tip: Close all open Firefox browser sessions and start a new session and enter this URL, https://wg31.washington.ibm.com:26223/CallRestApiWithJwtDemoWeb/ssl/InvokeRestWithJwt

© Copyright IBM Corporation 2020 All rights reserved

___8. Replace the *Bearer Token* in Postman with the new JWT token for *distuser2* and press the **Send** button.



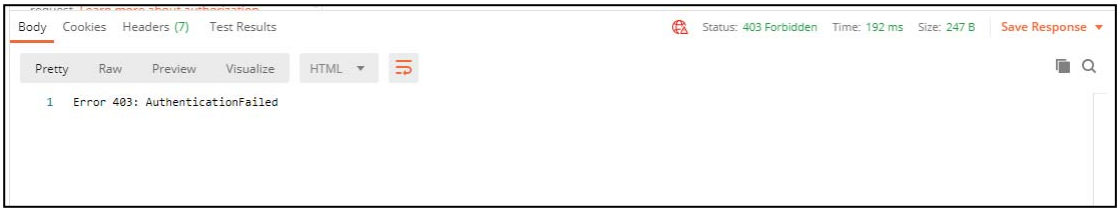
Tech-Tip: When replacing a bearer token in Postman be sure to remove all characters of the existing token before pasting the contents of the new token into the token area.

___9. In this case the request failed with an HTTP status code of *401 Unauthorized.* The distributed identity was authenticated by the *OP* server and a token was issued. The subject of the token was mapped to RACF identity *USER2*. An attempted was made to access the z/OS Connect API but it failed because the mapped RACF identity *USER2* was not authorized to use z/OS Connect.

```
ICH408I USER(USER2   ) GROUP(SYS1    ) NAME(WORKSHOP USER2      )
  ATSZDFLT.zos.connect.access.roles.zosConnectAccess
  CL(EJBROLE )
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

___10. Next test with a distributed identity which has not been mapped to a RACF identity Close all open Firefox browser session and obtain JWT token for distributed identity *auser*.

___11. Replace the Bearer Token in Postman with the new JWT token *auser* and press the **Send** button.



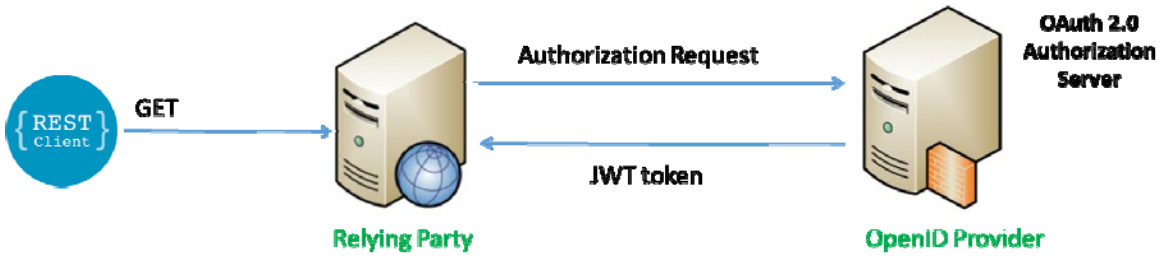
© Copyright IBM Corporation 2020 All rights reserved

___12. In this case the request failed with an HTTP status code of *403 Forbidden*. The distributed identity was authenticated by the OP server and a token was issued. The subject of the token was not mapped to a RACF identity.

```
ICH408I USER(ATSSERV ) GROUP(ATSGRP  ) NAME(LIBERTY SERVER      )
   DISTRIBUTED IDENTITY IS NOT DEFINED:
   auser zCEERealm
 ICH408I USER(ATSSERV ) GROUP(ATSGRP  ) NAME(LIBERTY SERVER      )
   DISTRIBUTED IDENTITY IS NOT DEFINED:
   auser zCEERealm
```

___13. Optional, create a JWT token for distributed identity *Fred* and use it in Postman.

To recap the complete scenario. A request was made to an application running in a Liberty server (RP) and authorization request flowed to an authorization server (OP) with a request for a JWT token.



The JWT token was then sent in a request to a z/OS Connect API Provider where it was used to authenticate the request and obtain an authorization identity.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

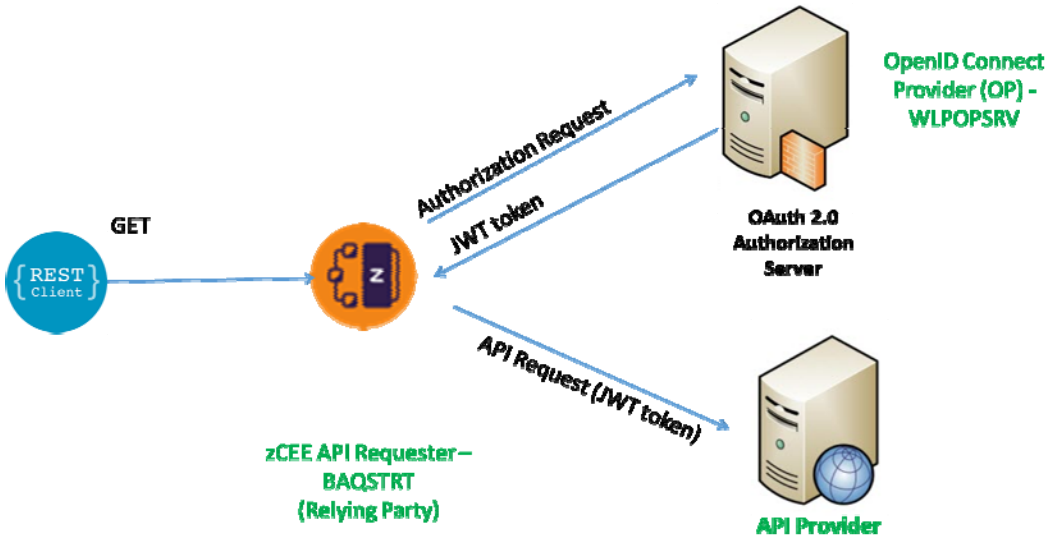
OAuth 2.0 and z/OS Connect EE API requesters

Securing access to a RESTful API from a z/OS Connect API requester application ben done using OAuth 2.0, an API key or a non-OAuth JWT token.

z/OS Connect provides two ways of calling an API secured with a JWT:

1. *Use OAuth 2.0 support when the request is part of an OAuth 2.0 flow.*
   If a JWT is returned as an access token from an authorization server, the access token is sent in an API request by z/OS Connect EE in the HTTP authorization header.
2. *Use custom JWT support when you need to send a JWT in a custom flow, for example:*
   - *When you need to specify the HTTP verb that is used in the request to the authentication server.*
   - *When you need to specify how the JWT is returned from the authentication server (for example, in an HTTP header or in a custom field in a JSON response message).*
   - *When you need to use a custom header name for sending the JWT to the request endpoint.*

In this exercise we will explore using the OAuth 2.0 solution (item 1 above) using a JWT token. as shown in the diagram below.



In this diagram the REST client is an MVS batch program. The batch program will connect to a z/OS Connect server and invoke an API requester. During his process the z/OS Connect server will perform the function of an OIDC RP. This means the z/OS Connect server will contact the OIDC OP and request an OIDC JWT token. If this request is authenticated, the OIDC OP will return an OIDC JWT. The API request will be sent to the API Provider along with the JWT token. If the API provider is a z/OS Connect API provider, the token will be validated, and the distributed identity mapped to a RACF identity. Standard Liberty and z/OS Connection will use this mapped identity for their authorization purposes. In this scenario this API is the same API used in the previous section of this exercise. The mapped identity will be propagated to CICS where the mapped identity will be returned in the response message.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

API Requester Sample Application

___1. Let’s begin by exploring the BAQRINFO copy book. This is the means by which an application program can provide credentials that will be used to obtain a token. (API security where keys are involved are handle using a different method). There are two structures defined in this copy book that are relevant for third party tokens. Structure *BAQ-OAUTH* is the structure used for obtaining OAuth 2.0 tokens. Structure *BAQ-AUTHTOKEN* is the structure used for obtaining non-OAuth tokens. Which structure used is determine ultimately by the nature of the OAuth authorization server.

```
          03 BAQ-REQUEST-INFO-USER.
             05 BAQ-OAUTH.
                07 BAQ-OAUTH-USERNAME           PIC X(256).
                07 BAQ-OAUTH-USERNAME-LEN       PIC S9(9) COMP-5 SYNC
                                                  VALUE 0.
                07 BAQ-OAUTH-PASSWORD           PIC X(256).
                07 BAQ-OAUTH-PASSWORD-LEN       PIC S9(9) COMP-5 SYNC
                                                  VALUE 0.
                07 BAQ-OAUTH-CLIENTID           PIC X(256).
                07 BAQ-OAUTH-CLIENTID-LEN       PIC S9(9) COMP-5 SYNC
                                                  VALUE 0.
                07 BAQ-OAUTH-CLIENT-SECRET      PIC X(256).
                07 BAQ-OAUTH-CLIENT-SECRET-LEN  PIC S9(9) COMP-5 SYNC
                                                  VALUE 0.
                07 BAQ-OAUTH-SCOPE-PTR          USAGE POINTER.
                07 BAQ-OAUTH-SCOPE-LEN          PIC S9(9) COMP-5 SYNC
                                                  VALUE 0.
             05 BAQ-AUTHTOKEN.
                07 BAQ-TOKEN-USERNAME           PIC X(256).
                07 BAQ-TOKEN-USERNAME-LEN       PIC S9(9) COMP-5 SYNC
                                                  VALUE 0.
                07 BAQ-TOKEN-PASSWORD           PIC X(256).
                07 BAQ-TOKEN-PASSWORD-LEN       PIC S9(9) COMP-5 SYNC
                                                  VALUE 0.
```

In this scenario the authorization server is an OAuth server so will be populating the fields in the BAQ-OAUTH structure.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___2. View member *GETAPIPT* in data set *USER1.ZCEE30.SOURCE* and scroll down until the code below is displayed. *Note that this code uses a technique described in the Getting Started Guide for passing environment variables from JCL to a COBOL program. This technique allows the testing of various OAuth configuration permutations by simply changing environment variables in the JCL rather than recompiling the program.*

```
      ************************************************
      **   Get the BAQ-OAUTH-USERNAME environment variable
      ************************************************
           MOVE "ATSOAUTHUSERNAME" TO envVariableName.
           PERFORM CALL-CEEENV THRU CALL-CEEENV-END
           IF valueLength NOT = 0 THEN
              MOVE VAR(1:valueLength) TO BAQ-OAUTH-USERNAME 1
              MOVE valueLength TO BAQ-OAUTH-USERNAME-LEN   2
              DISPLAY "BAQ-OAUTH-USERNAME:  "
                  BAQ-OAUTH-USERNAME(1:BAQ-OAUTH-USERNAME-LEN)
           ELSE
               DISPLAY "BAQ-OAUTH-USERNAME: Not found"
           END-IF.
      ************************************************
      **   Get the BAQ-OAUTH-PASSWORD environment variable
      ************************************************
           MOVE "ATSOAUTHPASSWORD" TO envVariableName.
           PERFORM CALL-CEEENV THRU CALL-CEEENV-END
           IF valueLength NOT = 0 THEN
              MOVE VAR(1:valueLength) TO BAQ-OAUTH-PASSWORD 3
              MOVE valueLength TO BAQ-OAUTH-PASSWORD-LEN  4
              DISPLAY "BAQ-OAUTH-PASSWORD:  "
                   BAQ-OAUTH-PASSWORD(1:BAQ-OAUTH-PASSWORD-LEN)
           ELSE
               DISPLAY "BAQ-OAUTH-PASSWORD: Not found"
           END-IF.
           MOVE "rpSsl"     to BAQ-OAUTH-CLIENTID. 5
           MOVE 5           to BAQ-OAUTH-CLIENTID-LEN. 6
           MOVE "secret"    to BAQ-OAUTH-CLIENT-SECRET. 7
           MOVE 6           to BAQ-OAUTH-CLIENT-SECRET-LEN.      8
           MOVE "openid"    to BAQ-OAUTH-SCOPE.       9
           MOVE 6           to BAQ-OAUTH-SCOPE-LEN.  10
           SET BAQ-OAUTH-SCOPE-PTR TO ADDRESS OF BAQ-OAUTH-SCOPE.
```

Notes:

1. Set the value of the OAuth resource manager's name from the environment variable.
2. Set the length of the OAuth resource manager's name.
3. Set the value of the OAuth resource manager's password from the environment variable.
4. Set the length of the OAuth resource manager's password.
5. Set the value of the OAuth client identity. Note, this field will be ignored since the client ID will be specified in the server XML configuration.
6. Set the length of the OAuth client identity.
7. Set the value of the OAuth client's password. Note, this field will be ignored since the client ID password will be specified in the server XML configuration.
8. Set the length of the OAuth client's password.
9. Set the value of the OAuth scope. Note, this value must match a scope as defined to the OP provider for this client.
10. Set the length of the OAuth scope.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

z/OS Connect API requester server - BAQSTRT

z/OS Connect API requester server BAQSTRT will fulfill the role of the OpenID Connect Replying Party in the earlier diagram.

___1. Edit the *server.xml* configuration file for the *myServer* server, e.g. */var/zosconnect/servers/myServer/server.xml* and change the include for file *keyringMutual.xml* to an include of file *keyringOutboundMutua.xml*,

<include location="/${server.config.dir}/includes/keyingOutboundMutual.xml"/>

and add the *includes* for file *oauth.xml and shared.xml* as shown below*:*

<include location="${server.config.dir}/includes/shared.xml"/>

<include location="${server.config.dir}/includes/oauth.xml"/>

```
<include location="${server.config.dir}/includes/ipicIDProp.xml"/>
<include location="${server.config.dir}/includes/safSecurity.xml"/>
<include location="${server.config.dir}/includes/keyringOutboundMutual.xml"/>
<include location="${server.config.dir}/includes/groupAccess.xml"/>
<include location="${server.config.dir}/includes/shared.xml"/>
<include location="${server.config.dir}/includes/oauth.xml"/>
```

The contents of *oauth.xml* are shown below.

```
 <zosconnect_apiRequesters location="/var/zcee/shared/apiRequesters"
   idAssertion="ASSERT_ONLY">
   <apiRequester name="cscvinc_1.0.0" requireSecure="false"/>
</zosconnect_apiRequesters>

<zosconnect_endpointConnection id="cscvincAPI" 1
  host="https://wg31.washington.ibm.com" port="9463"
  authenticationConfigRef="myoAuthConfig"/>   2

<zosconnect_oAuthConfig id="myoAuthConfig"   3
  grantType="password"   4
  authServerRef="myoAuthServer"/>       5

<zosconnect_authorizationServer id="myoAuthServer"   6
  tokenEndpoint="https://wg31.washington.ibm.com:26213/oidc/endpoint/OP/token" 7
  basicAuthRef="tokenCredential"   8
  sslCertsRef="OutboundSSLSettings" />

<zosconnect_authData id="tokenCredential"   9
  user="zCEEClient" password="secret"/>
```

Notes:

1. The API requester archive file references this endpoint Connection.
2. Identity the authentication configuration element.
3. The configuration element is for OAuth (*zosconnect_oAuthConfig*). A non-OAuth authorization server would have been identified by a *zosconnect.authToken* element.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

4. Set the grant type of *password*, i.e. obtain a token for the identity provided by the application in the *BAQ-OAUTH-USERNAME* field. A grant type of *client_credentails* would have obtain a token for the identity provided in the *BAQ-OAUTH-CLIENTID* field unless overridden by the values specified in the *zosconnect_authdata* configuration element.
5. Identify the authorization server configuration element.
6. The authorization server configuration element.
7. Provide the token endpoint URL. The authorization server should provide this value. For example Liberty provides this value in its Knowledge Center at URL https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_oidc_token_endpoint.html
8. Provide the client credentials. These credentials are used to authenticate the client to the authorization server.

Review the authorization server's XML configuration.

OpenID Connect Provider (OP) - WLPOPSRV

Below is snippet of the server XML configuration file for the Liberty OP server WLPOPSRV.

```
<httpEndpoint host="*" httpPort="26212" httpsPort="26213" id="defaultHttpEndpoint"/>

<!-- Define OpenID Connect Provider called OP  -->
<openidConnectProvider id="OP"   1
  signatureAlgorithm="RS256"
  keyStoreRef="jwtStore"
  oauthProviderRef="OIDCssl" >
</openidConnectProvider>

<oauthProvider id="OIDCssl"     2
  httpsRequired="true"
  jwtAccessToken="true"
  autoAuthorize ="true"
  accessTokenLifetime="300">
  <autoAuthorizeClient>rpSsl</autoAuthorizeClient>
   <localStore>
     <client name="rpSsl" . . ./>
    </localStore>
  <autoAuthorizeClient>zCEEClient</autoAuthorizeClient>
     <localStore>
       <client name="zCEEClient"        3
       secret="secret"
       displayname="zCEEClient"
       scope="openid profile scope1 email phone address"
       enabled="true"
       resourceIds="myZcee"/>
     </localStore>
</oauthProvider>

<!-- Key store that contains certificate used to sign JWT -->
<keyStore fileBased="false" id="jwtStore"            4
  location="safkeyring:///JWT.KeyRing"
  password="password" readOnly="true" type="JCERACFKS"/>

<!-- Define a basic user registry -->
<basicRegistry id="basicRegistry" realm="zCEERealm"  5
  <user name="auser" password="pwd"/>
  <user name="Fred" password="fredpwd"/>
  <user name="distuser1" password="pwd"/>
  <user name="distuser2" password="pwd"/>
</basicRegistry>
```


© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

Notes:

1. The *openidConnectProvider* configuration element is the initial configuration element for adding OAuth authorization server support to this Liberty server. The attribute specified here identify the signature algorithm used to verify an ID token. The keystore containing the self-signed certificate that will be embedded into the token and a reference to a corresponding *oauthProvider* configuration element.
2. The *oauthProvider* attribute is referenced by the *openidConnectProvider* configuration element. This configuration element defines the type of token to be generated (*jwtAccessToken=”true”*). The period of time for which the token will be valid (*accessTokenLifetime=”300”*).
3. The *localStore* configuration element provides configuration details for specific clients. In this case the RP will specify this *client name* when the authorization request is sent. The *scope* value of the client’s request will be validated with the scopes defined for this *localStore*.
4. This identifies the local SAF key ring contain the self-signed certificate which will be embedded in the JWT token.
5. This is the local registry used to authentication user identities and passwords. This Liberty server is configured to used basic authentication with a basic internal registry.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

z/OS Connect API Provider - ZCEEOPID

The zCEE server (ZCEEOPID) contains the server XML configuration below. In this scenario when an inbound API request is received where the path contains */cscvinc/employee*, this *openidConnectClient* configuration will be used to verify the JWT token embedded in the request. Once the JWT token is verified, the value of the *sub* field will be used for authentication.

```
<authFilter id="ATSAuthFilter">    1
   <requestUrl  id="ATSDemoUrl"
     name="ATSRefererUri" matchType="contains"
        urlPattern="/cscvinc/employee" />
 </authFilter>

 <!-- Define OIDC Client called ATS -->
 <openidConnectClient id="ATS"           2
      httpsRequired="true"
      authFilterRef="ATSAuthFilter"      3
      inboundPropagation="required"
      audiences="myZcee"         4
      issuerIdentifier="https://wg31.washington.ibm.com:26213/oidc/endpoint/OP" 5
      signatureAlgorithm="RS256"    6
      userIdentityToCreateSubject="sub"         7
      trustAliasName="JWT-Signer-Certificate"   8
      trustStoreRef="jwtTrustStore"      9
      authnSessionDisabled="true"
      disableLtpaCookie="true">
  </openidConnectClient>

 <keyStore fileBased="false" id="jwtTrustStore" 10
     location="safkeyring:///JWT.KeyRing"
      password="password" readOnly="true" type="JCERACFKS"/>
```

Notes:
1. The *authFilter* configuration element is used to select which inbound HTTP request will be processed by this security configuration. In this case, any inbound HTTP request containing the value */cscvinc/emloyee* will be processed.
2. The *openidConnectClient* configuration element provides the connection properties for the OP server.
3. The *authFilterRef* attribute is used to relate the *authFilter jwtIvpSslAuthFilter* with this *openidConnectClient* element.
4. The *audience* attribute identifies the intended recipient of this token.
5. The *issuerIdentifier* attribute is the case-sensitive URL using the HTTPS scheme that contains the scheme, host and optionally port name and path components of the OP server.
6. The *signatureAlgorithm* attribute is used to verify the JWT token and must match the value define in the OP server.
7. The *userIdentityToCreateSubject* attribute identifies the field in the payload which identifies the subject or to whom the token refers, i.e. the authenticated user identity.
8. The *trustAliasName* attribute is the label of a certificate in the designated key store. This certificate is used to validate the certificate embedded in the JWT token.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

9. The *trustStoreRef* attribute identity the key store which contains the certificate identified by the *trustAliasName* attribute.
10. The *keystore* attribute identifies RACF key ring which contains the certificate used to verify the JWT token.

___2. If job ***ZCEETLSC*** (TLS client role) in data set *USER1.ZCEE30.CNTL* has not submitted for another exercise, submit it now and ensure it completes successfully

___3. The out bound Key Ring (*ZCEE.KeyRing*) used by server BAQSTRT now needs to have to have additional certificate authority (CA) certificates connect. This is required because the communication between a RP and an OP must use TLS. Submit job ***JWTDIGTR*** in data set *USER1.ZCEE30.CNTL* for execution. This job invokes the commands below to add existing certificates the key rings used in this section of the exercise.

```
RACDCERT ID(LIBSERV) CONNECT +
 ( LABEL('OpenIdProv-CertAuth') RING(zCEE.KeyRing) CERTAUTH)

 RACDCERT ID(LIBSERV) CONNECT +
 ( LABEL('zCEE-CertAuth') RING(zCEE.KeyRing) CERTAUTH)

 RACDCERT ID(ATSSERV) CONNECT +
 ( LABEL('OpenIdProv-CertAuth') RING(zCEE.KeyRing) CERTAUTH)

 SETR RACLIST(digtring) REFRESH
```

___4. The API requester batch job used used in these exercise uses RACF pass tickets for the connection to the API requester server. Submit job ***BBGPTKT*** in data set *USER1.ZCEE30.CNTL* to define the required RACF resources. This job invokes the commands below:

```
SETROPTS CLASSACT(PTKTDATA) RACLIST(PTKTDATA)
SETROPTS GENERIC(PTKTDATA)

RDEFINE PTKTDATA BBGZDFLT SSIGNON(KEYMASK(123456789ABCDEF0)) +
APPLDATA('NO REPLAY PROTECTION')

RDEFINE PTKTDATA IRRPTAUTH.BBGZDFLT.* UACC(NONE)
PERMIT IRRPTAUTH.BBGZDFLT.* ID(LIBSERV,USER1) +
CLASS(PTKTDATA) ACC(UPDATE)

SETROPTS RACLIST(PTKTDATA) REFRESH
```

At the point the configuration of BAQSTRT has been updated and they Key Rings used by BAQSTRT and ZCEEOPID have been updated. Use MVS modify commands to update these servers.

___5. Enter MVS commands ***F BAQSTRT,REFRESH,CONFIG*** to refresh the zCEE API requester's runtime configuration.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___6. Enter MVS commands ***F BAQSTRT,REFRESH,KEYSTORE*** to refresh the zCEE API requester's Key Rings.

___7. Enter MVS commands ***F ZCEEOPID,REFRESH,KEYSTORE*** to refresh the zCEE API provider's Key Rings.

___8. Now explore the JCL that will be used for the tests. Open member ***GETAPIPT*** in data set *USER1.ZCEE30.CNTL. Note that this JCL uses 3 environment variables which are not part of the standard product, ATSAPPLID, ATSOAUTHUSERNAME and ATSOAUTHPASSWORD. These variables are being used to expediate the testing process.*

```
//GETAPI  EXEC PGM=GETAPIPT,PARM='111111'
//STEPLIB  DD DISP=SHR,DSN=USER1.ZCEE30.LOADLIB
//         DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//         DD DISP=SHR,DSN=JOHNSON.ZCEE.SDFHLOAD
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEOPTS  DD *
 POSIX(ON),
 ENVAR("BAQURI=wg31.washington.ibm.com",
 "BAQPORT=9080",
 "BAQUSERNAME=USER1",
 "ATSAPPLID=BBGZDFLT",
 "ATSOAUTHUSERNAME=distuser1",
 "ATSOAUTHPASSWORD=pwd")
```

When this job executes it will contact the z/OS Connect API requester server BAQSTRT (listening on port 9080). The z/OS Connect API requester server will send the credentials from the application along with credentials from the server XML configuration to the OP provider server, WLPOPSRV. The credentials will be authenticated and an OIDC JWT token will be returned to BAQSTRT. BAQSTRT will then send the API request to server, ZCEEOPID (a z/OS Connect API provider server). The JWT token will be authenticated and the API invoked. The results will be returned to the batch program with the results including the identity propagated to CICS.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___9. Submit this job as is and when it completes it should have the output below:

```
ATSPTKT-BAQUSERNAME: USER1
ATSPTKT-ATSAPPLID:   BBGZDFLT
ATSPTKT-BAQPASSWORD: P1S6MLRU
BAQ-OAUTH-USERNAME:  distuser1
BAQ-OAUTH-PASSWORD:  pwd
EmployeeNumber: 111111
EmployeeName:   C. BAKER
Address:        OTTAWA, ONTARIO
Phone:          51212003
Date:           26 11 81
Amount:         $0011.00
EIBRESP:        00000000
EIBRESP2:       00000000
USERID:         USER1
HTTP CODE:      0000000200
```

The results show that the OAuth username of ***distuser1*** was used to create a JWT token. The token was sent to the z/OS Connect API Provider and mapped to RACF identity ***USER1***. The identity ***USER1*** was propagated to CICS and the CICS transaction executed under this identity.

___10. Now change *ATSOAUTHUSERNAME* to ***FRED*** and *ATSOAUTHPASSWORD* to ***fred*** and resubmit the job.

```
"ATSOAUTHUSERNAME=Fred",
"ATSOAUTHPASSWORD=fred",
```

The request should fail with these messages:

```
ATSPTKT-BAQUSERNAME: USER1
ATSPTKT-ATSAPPLID:   BBGZDFLT
ATSPTKT-BAQPASSWORD: QMC6PUHX
BAQ-OAUTH-USERNAME:  Fred
BAQ-OAUTH-PASSWORD:  fred
Error code: 0000000500
Error msg:{"errorMessage":"BAQR1092E: Authentication or authorization failed for
the z/OS Connect EE server."}


Error origin:ZCEE
```

This request failed because the use name and password combination of Fred/fred is not valid on the OAuth server. Remember the password for Fred is **fredpwd.** Note, if you view the output of the OP Server (WLPOPSRV), you will see message *CWWKS1100A: Authentication did not succeed for user ID Fred. An invalid user ID or password was specified.*

___11. Change the password in the JCL to ***fredpwd*** and resubmit the job.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___12. This time the job should completed with a return code of 200 and these results.

```
ATSPTKT-BAQUSERNAME: USER1
ATSPTKT-ATSAPPLID:   BBGZDFLT
ATSPTKT-BAQPASSWORD: TR1ZTQFR
BAQ-OAUTH-USERNAME:  Fred
BAQ-OAUTH-PASSWORD:  fredpwd
EmployeeNumber: 111111
EmployeeName:   C. BAKER
Address:        OTTAWA, ONTARIO
Phone:          51212003
Date:           26 11 81
Amount:         $0011.00
EIBRESP:        00000000
EIBRESP2:       00000000
USERID:         FRED
HTTP CODE:      0000000200
```

The identity ***FRED*** was propagated to CICS and the CICS transaction executed under this identity. Remember distributed user ***Fred*** was mapped to RACF identity ***FRED***.

___13. Now change *ATSOAUTHUSERNAME* to ***distuser2*** and *ATSOAUTHPASSWORD* to ***pwd*** and resubmit the job.

```
"ATSOAUTHUSERNAME=distuser2",
"ATSOAUTHPASSWORD=pwd",
```

The request should fail with these messages:

```
ATSPTKT-BAQUSERNAME: USER1
ATSPTKT-ATSAPPLID:   BBGZDFLT
ATSPTKT-BAQPASSWORD: YJZRXQJX
BAQ-OAUTH-USERNAME:  distuser2
BAQ-OAUTH-PASSWORD:  pwd
Error code: 0000000403
Error msg:{"errorMessage":"BAQR1144E: Authentication or authorization failed for
the z/OS Connect EE server."}

Error origin:ZCEE
```

This request failed because ***USER2*** is not authorized to access z/OS Connect. Remember distributed user ***distuser2*** was mapped to RACF identity ***USER2*** which is not authorized to access the z/OS Connect server.

Review the SYSLOG using SDSF and these messages should appear.

```
ICH408I USER(USER2   ) GROUP(SYS1    ) NAME(WORKSHOP USER2      )
  ATSZDFLT.zos.connect.access.roles.zosConnectAccess
  CL(EJBROLE )
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```


© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___14. Finally, change *ATSOAUTHUSERNAME* to ***auser*** and *ATSOAUTHPASSWORD* to ***pwd*** and resubmit the job.

```
"ATSOAUTHUSERNAME=auser",
"ATSOAUTHPASSWORD=pwd",
```

The request should fail with these messages:

```
ATSPTKT-BAQUSERNAME: USER1
ATSPTKT-ATSAPPLID:   BBGZDFLT
ATSPTKT-BAQPASSWORD: YJZRXQJX
BAQ-OAUTH-USERNAME:  auser
BAQ-OAUTH-PASSWORD:  pwd
Error code: 0000000403
Error msg:{"errorMessage":"BAQR1144E: Authentication or authorization failed for
the z/OS Connect EE server."}



Error origin:ZCEE
```

This request failed because ***auser*** was not mapped to a RACF identity. Review the SYSLOG with SDSF and these messages should appear.

```
ICH408I USER(ATSSERV ) GROUP(ATSGRP  ) NAME(LIBERTY SERVER      )
  DISTRIBUTED IDENTITY IS NOT DEFINED:
  auser zCEERealm
ICH408I USER(ATSSERV ) GROUP(ATSGRP  ) NAME(LIBERTY SERVER      )
  DISTRIBUTED IDENTITY IS NOT DEFINED:
  auser zCEERealm
```

___15. Next change edit *oauth.xml* in directory */var/zcee/myServer/includes/* and change *grantType* from *password* to ***client_credentials***

```
<zosconnect_oAuthConfig id="myoAuthConfig"
  grantType="client_credentials"
  authServerRef="myoAuthServer"/>
```

___16. Refresh the server's configuration with MVS modify command

F BAQSTRT,REFRESH,CONFIG

___17. Resubmit ***GETAPIPIT*** **a**nd observe the results.

The request should fail with an HTTP 403 error code. Look in the SYSLOG and you should see these messages

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

___18. Look in the SYSLOG and you should find these messages. These messages indicated that the z/OS Connect API requester obtained a JWT token from the authorization server with the client identity as the subject rather than the user identity. The attempt to map the subject to a RACF identity failed, there was no mapping.

```
ICH408I USER(ATSSERV ) GROUP(ATSGRP  ) NAME(LIBERTY SERVER      )
  DISTRIBUTED IDENTITY IS NOT DEFINED:
  zCEEClient zCEERealm
ICH408I USER(ATSSERV ) GROUP(ATSGRP  ) NAME(LIBERTY SERVER      )
  DISTRIBUTED IDENTITY IS NOT DEFINED:
  zCEEClient zCEERealm
```

___19. Use the RACMAP command in job JWTMACMP to map distributed identity zCEEClient to a valid RACF identity, e.g. FRED, USER1, or USER2. Do not forget to refresh the IDIDMAP resources.

___20. Submit ***GETAPIPT*** again and see if the results based on the mapped RACF identity occur.

Summary

In this exercise used Liberty as an authorization server to create JWT tokens for providing security when invoking an API in a z/OS Connect API provider server for when invoking an API using z/OS Connect API requester client.

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

Appendix

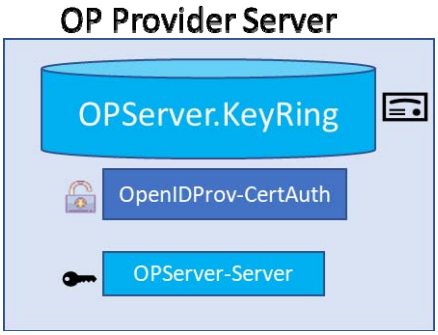
This section provides the RACF commands used to create the certificates and key rings used by the z/OS Connect API provider and requester servers and the authorization server. Also provided in this section the server XML configuration elements for the keys stores of the servers is provided.

Authorization Server (WLPOPSRV)

• These are the RACF commands that were used to the create certificate authority (CA) and the server certificate used by the OAUTH authorization server.

```
RACDCERT CERTAUTH GENCERT +
SUBJECTSDN(CN('OpenIdProv')OU('CertAuth')) +
WITHLABEL('OpenIdProv-CertAuth') +
TRUST SIZE(1024) NOTAFTER(DATE(2029/12/31))

RACDCERT ID(ATSSERV) GENCERT +
SUBJECTSDN(CN('OpenId-Server') OU('ATS') O('IBM') C('USA')) +
WITHLABEL('OpenIdProv-Server') +
ALTNAME(DOMAIN('wg31.washington.ibm.com')) +
SIGNWITH(CERTAUTH LABEL('OpenIdProv-CertAuth')) +
NOTAFTER(DATE(2029/12/31)) SIZE(2048)

RACDCERT ADDRING(OpServer.KeyRing) ID(ATSSERV)

RACDCERT ID(ATSSERV) CONNECT +
( LABEL('OpenIdProv-CertAuth') RING(OpServer.KeyRing) CERTAUTH)

RACDCERT ID(ATSSERV) CONNECT +
(ID(ATSSERV) RING(OpServer.KeyRing) LABEL('OpenIdProv-Server'))
```

Below is a visual representation of the keyring the above commands created.



© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

- The authorization server's server (WLPOPSRV) XML configuration for TLS and key rings.

```
<httpEndpoint host="*" httpPort="26212" httpsPort="26213"
  id="defaultHttpEndpoint"/>

<sslDefault sslRef="defaultSSLSettings"/>
<ssl clientAuthentication="false" id="defaultSSLSettings"
  keyStoreRef="OpServerKeyStore"
  trustStoreRef="OpServerKeyStore"/>
<keyStore fileBased="false" id="OpServerKeyStore"
  location="safkeyring:///OpServer.KeyRing"
  password="password" readOnly="true" type="JCERACFKS"/>
```

- The contents of the authorization server's server (WLPOPSRV) key ring.

```
Ring:
     >OpServer.KeyRing<
 Certificate Label Name              Cert Owner     USAGE      DEFAULT
 --------------------------------    ------------   --------   -------
 OpenIdProv-CertAuth                 CERTAUTH       CERTAUTH     NO
 OpenIdProv-Server                   ID(ATSSERV)    PERSONAL     YES
```

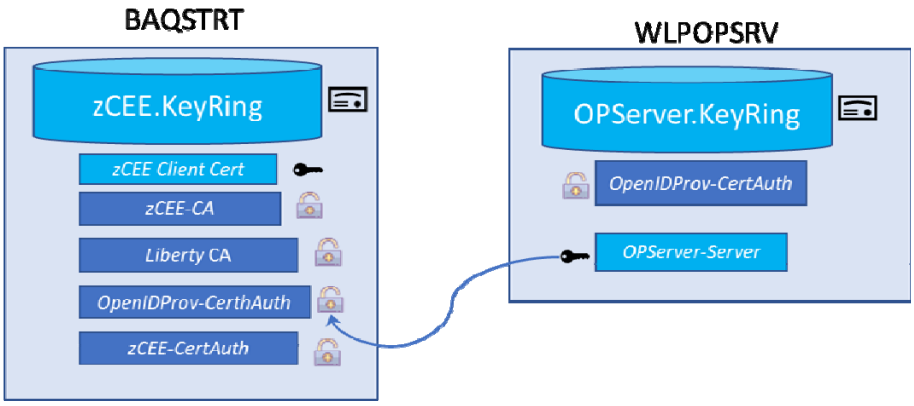

© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

z/OS Connect API Requester server (BAQSTRT)

• These are the RACF commands that were used to the create certificate authority (CA) and the server certificate used by the z/OS Connect API requester server (BAQSTRT).

```
/* Create personal certficate for zCEE outbound client request */
racdcert id(libserv) gencert subjectsdn(cn('zCEE Client Cert') +
ou('ATS') o('IBM')) withlabel('zCEE Client Cert') signwith(certauth
label('zCEE CA')) notafter(date(2022/12/31))

 /* Create zCEE outbound key ring and connect certificates */
racdcert id(libserv) addring(zCEE.KeyRing)

racdcert id(libserv) connect(ring(zCEE.KeyRing) +
          label('zCEE CA') certauth usage(certauth))

racdcert id(libserv) connect(ring(zCEE.KeyRing) +
          label('Liberty CA') certauth usage(certauth))

 /* Connect CA certificate to Liberty inbound key ring */
racdcert id(libserv) connect(ring(Liberty.KeyRing) +
          label('zCEE CA') certauth usage(certauth))

 /* Connect default personal certificate */
racdcert id(libserv) connect(ring(zCEE.KeyRing) +
          label('zCEE Client Cert') default)
```

Below is a visual representation of the keyring the above commands created and by job *JWTDIGT* and the handshake flow between the OP Provider server and the z/OS Connect RP server.



© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

- The z/OS Connect EE API requester server's XML configuration for TLS and key rings.

```
<sslDefault sslRef="DefaultSSLSettings"
  outboundSSLRef="OutboundSSLSettings" />

<ssl id="DefaultSSLSettings"
   keyStoreRef="CellDefaultKeyStore"
   trustStoreRef="CellDefaultKeyStore"
   clientAuthenticationSupported="true"
   clientAuthentication="true"/>

<keyStore id="CellDefaultKeyStore"
   location="safkeyring:///Liberty.KeyRing"
   password="password" type="JCERACFKS"
   fileBased="false" readOnly="true" />

<ssl id="OutboundSSLSettings"
   keyStoreRef="OutboundKeyStore"
   trustStoreRef="OutboundKeyStore"/>

<keyStore id="OutboundKeyStore"
   location="safkeyring:///zCEE.KeyRing"
   password="password" type="JCERACFKS"
   fileBased="false" readOnly="true" />
```

- The contents of the z/OS Connect EE API requester server's inbound and outbound key rings.

```
Digital ring information for user LIBSERV:

  Ring:
       >zCEE.KeyRing<
  Certificate Label Name              Cert Owner     USAGE      DEFAULT
  --------------------------------    ------------   --------   -------
  zCEE CA                             CERTAUTH       CERTAUTH     NO
  Liberty CA                          CERTAUTH       CERTAUTH     NO
  zCEE Client Cert                    ID(LIBSERV)    PERSONAL     YES
  OpenIdProv-CertAuth                 CERTAUTH       CERTAUTH     NO
  zCEE-CertAuth                       CERTAUTH       CERTAUTH     NO

  Ring:
       >Liberty.KeyRing<
  Certificate Label Name              Cert Owner     USAGE      DEFAULT
  --------------------------------    ------------   --------   -------
  Liberty Client Cert                 ID(LIBSERV)    PERSONAL     YES
  Liberty CA                          CERTAUTH       CERTAUTH     NO
  zCEE CA                             CERTAUTH       CERTAUTH     NO
```


© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

API Provider z/OS Connect server (ZCEEOPID)

• These are the RACF commands used to the create certificate authority (CA) and the server certificate used by the z/OS Connect API requester server (ZCEEOPID).

```
RACDCERT CERTAUTH GENCERT +
SUBJECTSDN(CN('zCEE')OU('CertAuth')) WITHLABEL('zCEE-CertAuth') +
TRUST SIZE(1024) NOTAFTER(DATE(2029/12/31))

* Create a certificate to be server-side cert in the zCEE server
RACDCERT ID(ATSSERV) GENCERT +
SUBJECTSDN(CN('wg31.washington.ibm.com') +
OU('ATS') O('IBM') C('USA')) WITHLABEL('zCEE-Server') +
SIGNWITH(CERTAUTH LABEL('zCEE-CertAuth')) +
NOTAFTER(DATE(2029/12/31)) SIZE(2048)

*Create a key ring for the userid the server runs under
RACDCERT ADDRING(zCEE.Server.KeyRing) ID(ATSSERV)

* Connect Cert Auth certificate to user keyring
RACDCERT ID(ATSSERV) CONNECT +
( LABEL('zCEE-CertAuth') RING(zCEE.Server.KeyRing) CERTAUTH)

RACDCERT ID(ATSSERV) CONNECT +
(ID(ATSSERV) RING(zCEE.Server.KeyRing) LABEL('zCEE-Server'))

RACDCERT ID(ATSSERV) CONNECT +
( LABEL('zCEE-Client-CertAuth') RING(zCEE.Server.KeyRing) CERTAUTH)
```

• The z/OS Connect EE API requester server’s XML configuration for TLS and key rings.

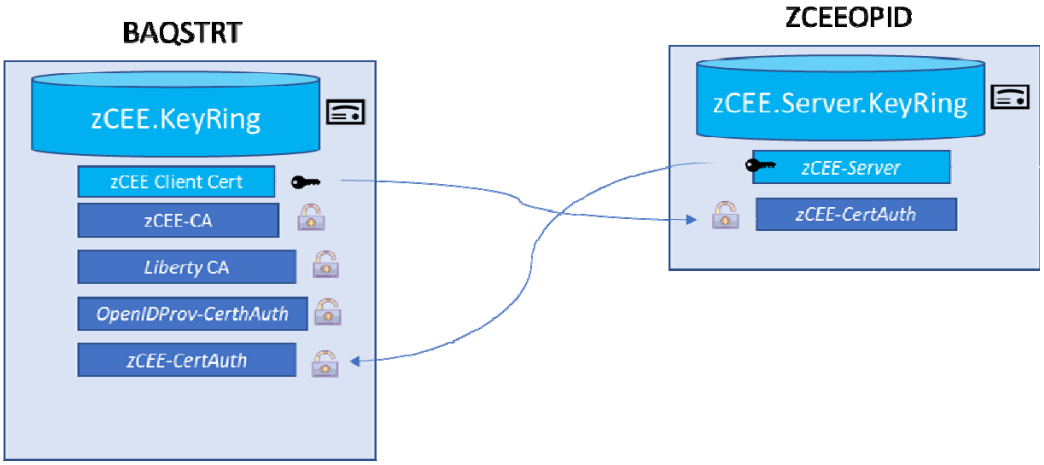
```
<safCredentials
      mapDistributedIdentities="true"
      unauthenticatedUser="ZCGUEST"
      suppressAuthFailureMessages="false"
      profilePrefix="ATSZDFLT" />

 <ssl clientAuthentication="false"
      id="defaultSSLSettings"
      keyStoreRef="racfCert"
      trustStoreRef="racfCert"/>

 <keyStore fileBased="false" id="racfCert"
      location="safkeyring:///zCEE.Server.KeyRing"
      password="password" readOnly="true" type="JCERACFKS" />
```

• The contents of the z/OS Connect EE API requester server’s key ring.

```
Ring:
>zCEE.Server.KeyRing<
Certificate Label Name              Cert Owner     USAGE      DEFAULT
--------------------------------    ------------   --------   -------
zCEE-CertAuth                       CERTAUTH       CERTAUTH     NO
zCEE-Server                         ID(ATSSERV)    PERSONAL     YES
```


© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

• Below is a visual representation of the TLS hand-share between the API requester server and the API provider server.



© Copyright IBM Corporation 2020 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)