WebLogic Custom Security Provider for OAUTH2/JWT authentication on Oracle Service Bus

Overview

Up to and including version 12.1.3 the Oracle Service Bus does not support OAUTH2/JWT inbound and outbound authentication out of the box. Starting with version 12.2.1, the OSB supports it through the use of OWSM policies but without the certification and flexibility needed to use a third-party IDP such as Azure Entra ID.

Furthermore, the use of OWSM policies may not be a proper solution for those who are used to managing authentication and authorization through the simple management of users and groups of the integrated authentication provider of WebLogic. As if that wasn't enough, OAUTH2 introduces the need to adopt identities defined by very long and opaque strings (client_id), that are difficult to re-associate to a given consumer without appropriate mechanisms of credential mappings and in this OWSM is of no help.

Fortunately, since the old versions of WebLogic there is the possibility to extend the product to support custom authentication schemes. The library proposed in this project is based in particular on a Custom Identity Assertion Provider that brings to the OSB an implementation of OAUTH2 authentication based on signed JWT tokens, currently only for inbound OSB security. This custom provider overcomes the rigidities of OWSM by offering more flexibility and control and optionally supports identity mapping to translate client_ids to WebLogic realm users.

In addition to the JWT-based authentication scheme, the provider also offers support for the legacy Basic Auth to simplify the progressive adoption of JWT authentication by different consumers on the same Proxy Service, without the need to create different Proxies for each authentication scheme.

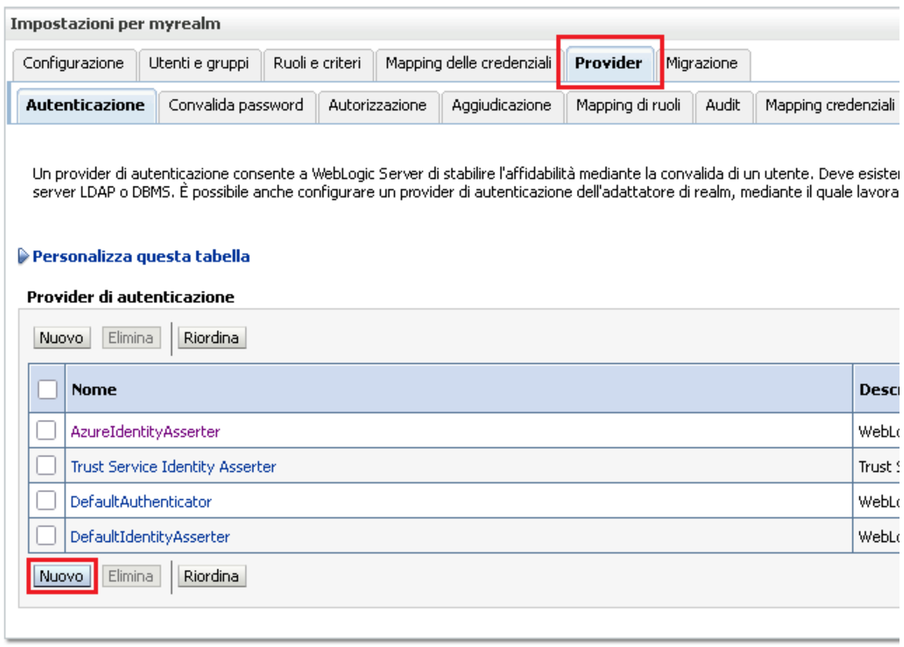
It has currently been tested on all most recent Oracle Service Bus 12.1.3, 12.2.1.4 and 14.1.2 with Azure Entra ID as the IDP.

Installation

For in-depth information on Custom Providers, please refer to the product documentation (see references). In short, first you need to stop WebLogic and copy the provider packages into the folder:

```
<WEBLOGIC_HOME>/wlserver/server/lib/mbeantypes
```

Once you have restarted WebLogic, as shown in the following screenshots, you just need to create a new provider using the "Provider" tab of the Realm settings in WebLogic Console, selecting the "CustomIdentityAsserter" (CIA) item which is the identifier of this provider. We then need to reorder the providers to move ours to the top.


Impostazioni per myrealm

Configurazione | Utenti e gruppi | Ruoli e criteri | Mapping delle credenziali | **Provider** | Migrazione

Autenticazione | Convalida password | Autorizzazione | Aggiudicazione | Mapping di ruoli | Audit | Mapping credenziali

Un provider di autenticazione consente a WebLogic Server di stabilire l'affidabilità mediante la convalida di un utente. Deve esiste
server LDAP o DBMS. È possibile anche configurare un provider di autenticazione dell'adattatore di realm, mediante il quale lavora

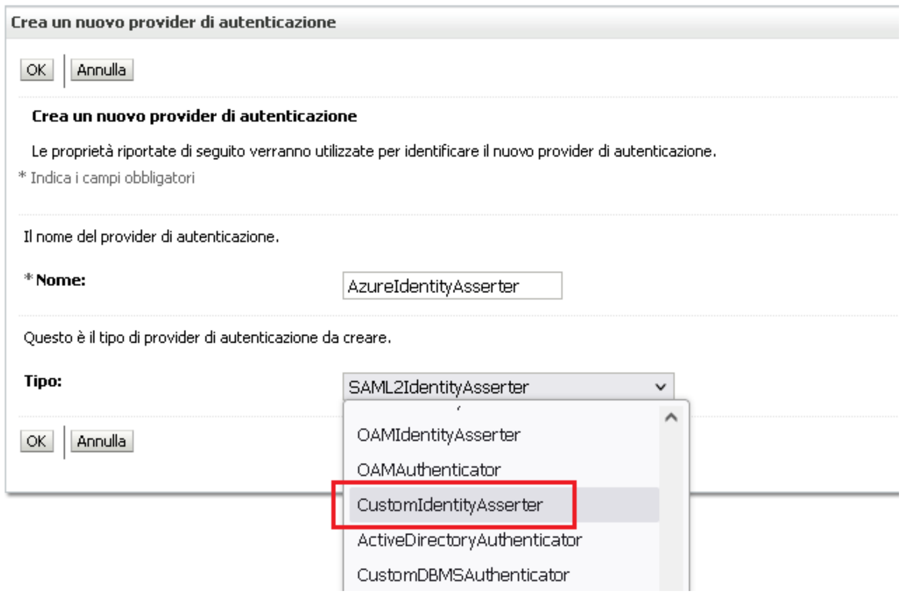
Personalizza questa tabella

Provider di autenticazione

Nuovo | Elimina | Riordina

	Nome	Desc
☐	AzureIdentityAsserter	WebL
☐	Trust Service Identity Asserter	Trust
☐	DefaultAuthenticator	WebL
☐	DefaultIdentityAsserter	WebL

Nuovo | Elimina | Riordina


Crea un nuovo provider di autenticazione

OK | Annulla

Crea un nuovo provider di autenticazione

Le proprietà riportate di seguito verranno utilizzate per identificare il nuovo provider di autenticazione.
* Indica i campi obbligatori

Il nome del provider di autenticazione.

* **Nome:** AzureIdentityAsserter

Questo è il tipo di provider di autenticazione da creare.

Tipo: SAML2IdentityAsserter

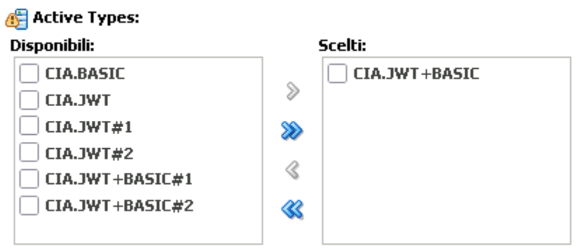
OK | Annulla

Token Types

Installing the provider registers in the system the presence of new types of "Tokens" that can be used to secure the Proxy Services. In WebLogic Identity Asserter terminology, Token Types are a declarative way to show which authentication schemes a given provider supports and which are active at a given time, i.e. which can be selected for authentication of a Proxy Service. The provider proposed in this project supports the JWT and BASIC schemes and allows to select them individually or in a combined way through the "JWT+BASIC" type.

The "JWT+BASIC" typology is useful because on a given Proxy Service it is possible to select only one type of token at a time and the combined token allows to keep both authentication schemes active at the same time on a single Proxy Service. This allows to implement a progressive migration of consumers from the BASIC scheme to the JWT scheme in a progressive way, without having to create different Proxy Services for each scheme.

As you can see from the image below the provider offers similar types but with a different suffix (#1 and #2). This makes it possible to create multiple instances of the same provider and differentiate their configuration to support different IDPs.


Active Types:

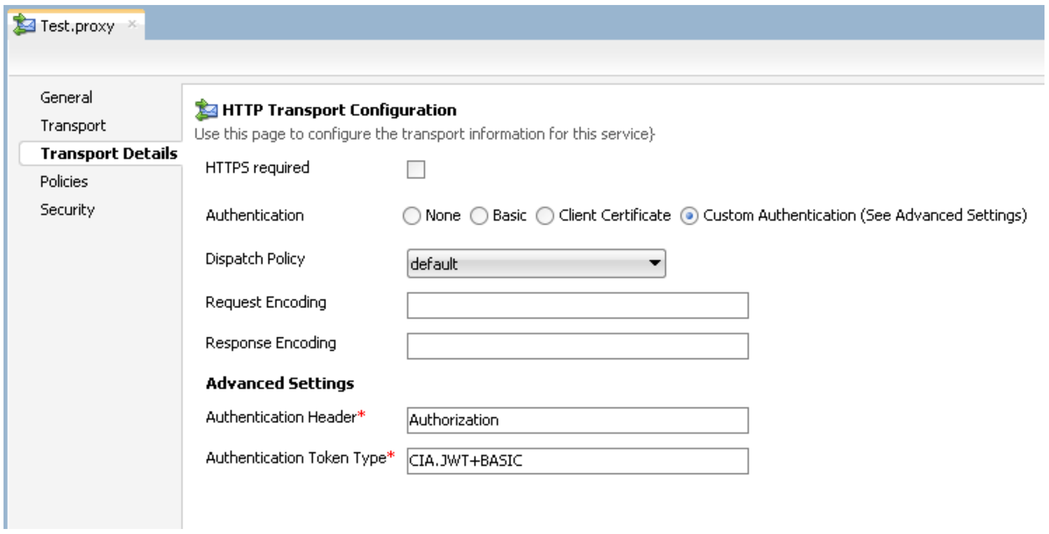
Disponibili:	Scelti:
☐ CIA.BASIC	☐ CIA.JWT+BASIC
☐ CIA.JWT	
☐ CIA.JWT#1	
☐ CIA.JWT#2	
☐ CIA.JWT+BASIC#1	
☐ CIA.JWT+BASIC#2	

How to configure Proxy Services

To enable the use of our provider on Proxy Services, you need to act from the JDeveloper IDE or from the Service Bus console on the configuration of the Proxy transport details as shown in the following screenshot.

In the "Authentication Header" field the http header must be specified, whose presence activates the use of the custom authentication provider. It can be any valid identifier, however if you want to support the BASIC authentication scheme together with the JWT scheme, the header must necessarily be the standard "Authorization".

In the "Authentication Token Type" field you need to select one of the token types selected as active in the Provider configuration.



HTTP Transport Configuration
Use this page to configure the transport information for this service}

HTTPS required	☐
Authentication	None Basic Client Certificate [✓ Custom Authentication (See Advanced Settings)]
Dispatch Policy	default
Request Encoding	
Response Encoding	

Advanced Settings

Authentication Header*	Authorization
Authentication Token Type*	CIA.JWT+BASIC

Provider Parameters

The provider is highly configurable and can be adapted to be used with different types of identity providers.
Below is a detailed description of each parameter.

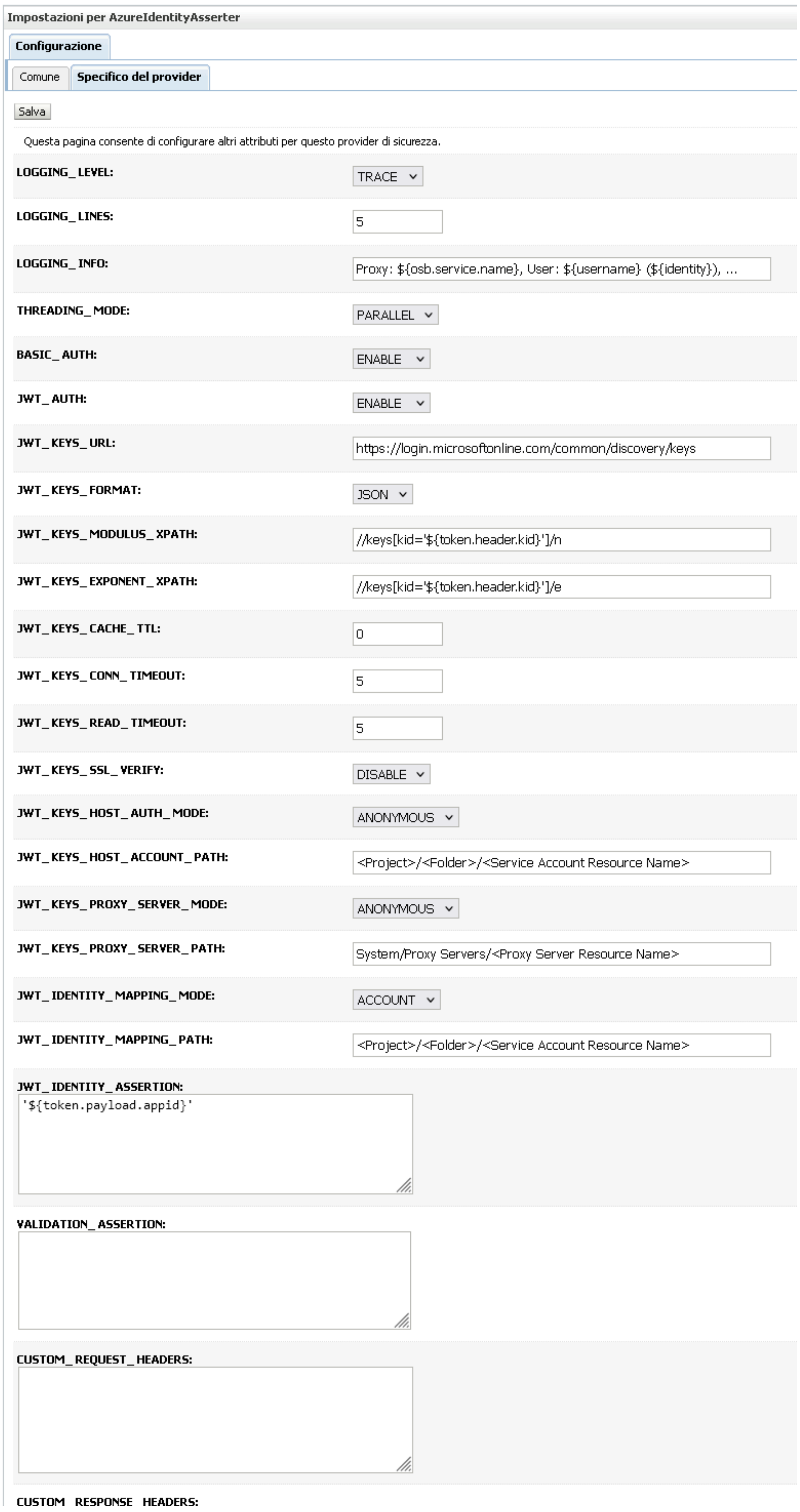
Parameter	Description
`LOGGING_LEVEL`	Minimum level of log messages printed.
`LOGGING_LINES`	Maximum number of stacktrace lines logged.
`LOGGING_INFO`	Allows you to specify the format of the logging line generated with the INFO level at the end of each assertion
`THREADING_MODE`	Multithreading strategy.
`BASIC_AUTH`	Allows to control Basic authentication if it is among those actives.
`JWT_AUTH`	Allows to control JWT authentication if it is among those actives.
`JWT_KEYS_URL`	Download URL for public keys used for JWT signature validation.
`JWT_KEYS_FORMAT`	Payload format for public keys. Could be JSON or XML.
`JWT_KEYS_MODULUS_XPATH`	XPath expression for public key modulus. If payload is in JSON format it's translated to XML before. The default value is suitable for standard JWKS (JSON Web Key Set).

Parameter	Description
`JWT_KEYS_EXPONENT_XPATH`	XPath expression for public key exponent. If payload is in JSON format it's translated to XML before. The default value is suitable for standard JWKS (JSON Web Key Set).
`JWT_KEYS_CACHE_TTL`	Public keys caching duration (Seconds).
`JWT_KEYS_CONN_TIMEOUT`	Public keys server connection timeout (Seconds).
`JWT_KEYS_READ_TIMEOUT`	Public keys server response timeout (Seconds).
`JWT_KEYS_SSL_VERIFY`	Public keys request SSL enforcement. Use only in non-production environement to test endpoints.
`JWT_KEYS_HOST_AUTH_MODE`	Public keys server require authentication. The following choices are supported : ANONYMOUS, BASIC, NTLM, KERBEROS (NEGOTIATE).
`JWT_KEYS_HOST_ACCOUNT_PATH`	OSB resource path (*) of the "Service Account" used to get public keys credentials.
`JWT_KEYS_PROXY_SERVER_MODE`	Public keys server require proxy mediation. The following choices are supported : DIRECT (no proxy), ANONYMOUS, BASIC, NTLM, KERBEROS (NEGOTIATE).
`JWT_KEYS_PROXY_SERVER_PATH`	OSB resource path (*) for the "Proxy Server" used to extract proxy host and credentials.
`JWT_IDENTITY_MAPPING_MODE`	Specifies whether the token identity is translated into the WebLogic realm username using an OSB "Service Account" resource.
`JWT_IDENTITY_MAPPING_PATH`	OSB resource path (*) of the "Service Account" used for mapping user identity to realm username.
`JWT_IDENTITY_ASSERTION`	Must contain a javascript text that returns the identity of the jwt token according to the specifications of the IDP used. It must return a String object.
`VALIDATION_ASSERTION`	May contain a javascript text that must validate or reject the authentication request according to arbitrary criteria defined by the user. If present, it must return a Boolean object.
`CUSTOM_REQUEST_HEADERS`	Allows you to inject one or more custom http request headers that will pass through the OSB context. Each line must follow the format <header>=<value>.
`CUSTOM_RESPONSE_HEADERS`	Allows you to inject one or more custom http response headers that will return to client. Each line must follow the format <header>=<value>.
`DEBUGGING_ASSERTION`	May contain a javascript text that is used to filter log messages with TRACE or DEBUG level according to arbitrary criteria defined by the user. This can be useful to reduce log messages and analyze specific requests. If present, it must return a Boolean object.
`DEBUGGING_PROPERTIES`	Allows you to send one or more string expressions to the log file. They are printed as log messages with DEBUG level. Any template variables are resolved allowing you to analyze the runtime context.
`KERBEROS_CONFIGURATION`	tbd

(*) OSB resources path are constructed as follows: `<project-name>/<root-folder>/.../<parent-folder>/<resource-name>` .
If a resource is located directly under a project, the path is constructed as follows: `<project-name>/<resource-name>` .
Please note that resources of type "Proxy Server" can only be created in the fixed path `System/Proxy Servers/<resource-name>` .

For more information on OSB resources follow this link

Below is a screenshot of the available parameters populated with sample values suitable for Azure Entra ID.



Impostazioni per AzureIdentityAsserter

Configurazione

Comune | Specifico del provider

Salva

Questa pagina consente di configurare altri attributi per questo provider di sicurezza.

LOGGING_LEVEL:	TRACE
LOGGING_LINES:	5
LOGGING_INFO:	Proxy: ${osb.service.name}, User: ${username} (${identity}), ...
THREADING_MODE:	PARALLEL
BASIC_AUTH:	ENABLE
JWT_AUTH:	ENABLE
JWT_KEYS_URL:	https://login.microsoftonline.com/common/discovery/keys
JWT_KEYS_FORMAT:	JSON
JWT_KEYS_MODULUS_XPATH:	//keys[kid='${token.header.kid}']/n
JWT_KEYS_EXPONENT_XPATH:	//keys[kid='${token.header.kid}']/e
JWT_KEYS_CACHE_TTL:	0
JWT_KEYS_CONN_TIMEOUT:	5
JWT_KEYS_READ_TIMEOUT:	5
JWT_KEYS_SSL_VERIFY:	DISABLE
JWT_KEYS_HOST_AUTH_MODE:	ANONYMOUS
JWT_KEYS_HOST_ACCOUNT_PATH:	<Project>/<Folder>/<Service Account Resource Name>
JWT_KEYS_PROXY_SERVER_MODE:	ANONYMOUS
JWT_KEYS_PROXY_SERVER_PATH:	System/Proxy Servers/<Proxy Server Resource Name>
JWT_IDENTITY_MAPPING_MODE:	ACCOUNT
JWT_IDENTITY_MAPPING_PATH:	<Project>/<Folder>/<Service Account Resource Name>

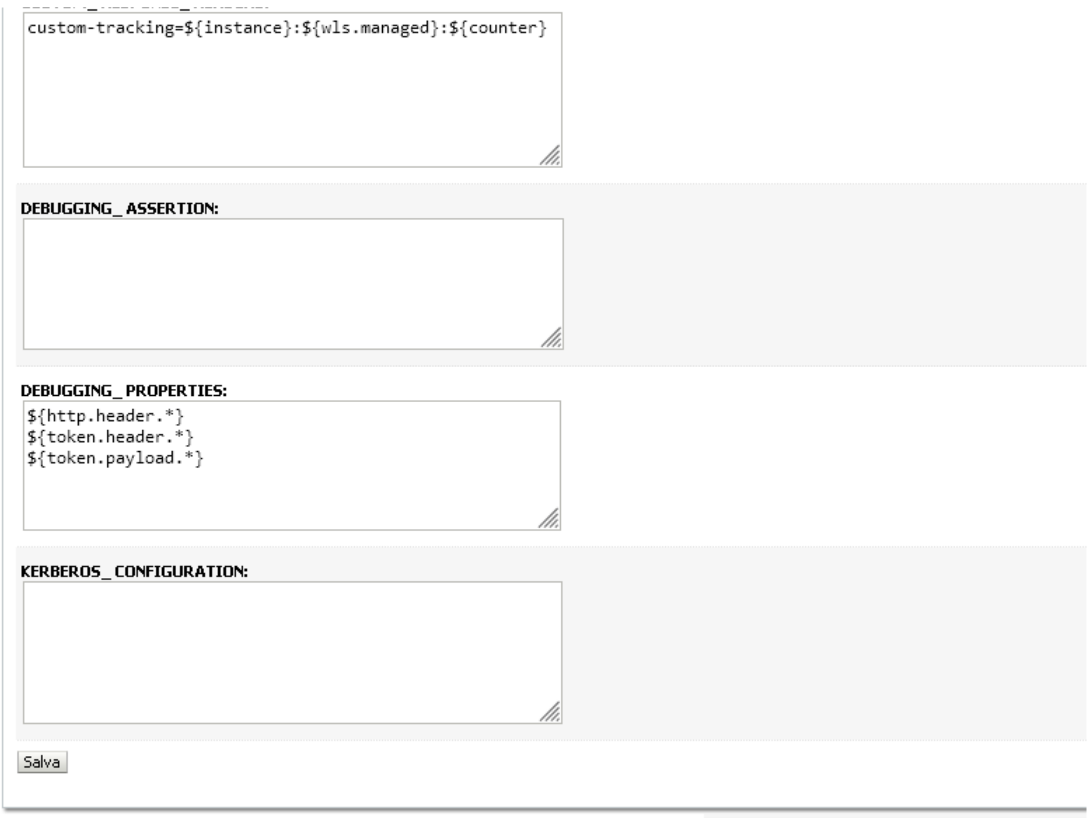
JWT_IDENTITY_ASSERTION:

```
'${token.payload.appid}'
```

VALIDATION_ASSERTION:

CUSTOM_REQUEST_HEADERS:

CUSTOM_RESPONSE_HEADERS:


```
custom-tracking=${instance}:${wls.managed}:${counter}
```

```
${http.header.*}
${token.header.*}
${token.payload.*}
```

Template Variables

All string configuration parameters support the use of substitution variables to create configurations that can dynamically adapt to the runtime state. The following is a list of supported variables.

Variable	Replaced by
`${thread}`	Current thread id
`${provider}`	The user-assigned provider instance name
`${instance}`	Unique identifier of the instance. It is always "CIAx" where x is a counter of instances created for this provider.
`${authtype}`	Authentication detected (BASIC or JWT)
`${identity}`	Asserted JWT token identity.
`${username}`	Asserted UserName. In the case of Basic Auth it coincides with the authenticated user while in the case of JWT Auth, if mapping is not required it coincides with the ${identity} otherwise it is the user mapped by the OSB mapping service account. Must exist in the WebLogic realm.
`${request.counter}`	Request counter for this provider instance on current managed server
`${request.datetime}`	Request timestamp in the format yyyy-MM-dd HH:mm:ss.SSS
`${request.timestamp}`	Request milliseconds since unix epoch
`${current.datetime}`	Current timestamp in the format yyyy-MM-dd HH:mm:ss.SSS
`${current.timestamp}`	Current milliseconds since unix epoch
`${wls.realm}`	WebLogic Realm Name
`${wls.domain}`	WebLogic Domain Name
`${wls.managed}`	WebLogic Manager Server Name
`${osb.project}`	The name of the Osb Project that the endpoint that received the request is part of.
`${osb.service.name}`	The name of the Osb Proxy Service that took charge of the request.
`${osb.service.path}`	The full path the Osb Proxy Service that took charge of the request.

Variable	Replaced by
`${http.client.host}`	The remote/client hostname of the http request.
`${http.client.addr}`	The remote/client address of the http request.
`${http.server.host}`	The local/server hostname of the machine that took charge of the request.
`${http.server.addr}`	The local/server address of the machine that took charge of the request.
`${http.server.name}`	The hostname declared in the http request by the client.
`${http.content.type}`	The content mime/type declared in http request by the client.
`${http.content.body}`	The body sent in http request by the client.
`${http.content.length}`	The content length of body.
`${http.request.url}`	The original url of http request.
`${http.request.proto}`	The protocol version of http request.
`${http.request.scheme}`	The scheme of http request.
`${http.header.<name>}`	The value of the http header <name> in the http request.
`${http.header.*}`	Enumerate all http headers of the request, except "Authorization" to avoid disclosing password.
`${token.header.<attr>}`	The value of the header <attr> element in the JWT token. If token is not initialized return blank.
`${token.header.*}`	Enumerate all attributes of JWT token header. If token is not initialized return blank.
`${token.payload.<attr>}`	The value of the payload <attr> element in the JWT token. If token is not initialized return blank.
`${token.payload.*}`	Enumerate all attributes of JWT token payload. If token is not initialized return blank.
`${java.version}`	Java major version

Identity mapping strategies

It is possible to implement different strategies to establish the identities of the JWT token and eventually map this identity to the users of the weblogic realm. Let's see some possible scenarios below.

1. Direct identity

The simplest scenario is to use one of the attributes of the JWT token that uniquely defines its identity (typically a "client_id"), ensuring that in the weblogic realm there is a user with the same username as the identity. This scenario does not require any identity mapping and does not necessarily require the use of special JWT claims. The problem is that in the WebLogic realm we observe the proliferation of esoteric users that are not immediately attributable to consumers. Furthermore, if a consumer already authenticates with a Basic Auth on an existing weblogic user, it cannot be reused and the profiling of users must be reproduced from scratch.

2. Claim identity

If you can fully trust the identity provisioning process on the IDP and it is possible to request the configuration of the claims on the JWT token it is possible to implement a form of identity mapping by delegating it to the IDP. For example, you could use the standard claim "sub" (Subject) making sure that it is populated with one of the users of the weblogic realm that at this point can be created according to the format that you like, possibly reusing existing users for the Basic Auth and therefore without the need to re-profile them.

In this case the `JWT_IDENTITY_ASSERTION` parameter would be valued with: `'${token.payload.sub}'` .

3. Mapped identity

If you prefer not to delegate the identity mapping to the IDP, you can use any of the JWT token attributes to extrapolate its identity and then map it to a realm username using a OSB "Service Account" resource. For example, if you use the Azure Entra ID as IDP, you can typically use the "appid" attribute as the token identity, which typically contains the "client_id".

In this case the `JWT_IDENTITY_ASSERTION` parameter would be valued with: `'${token.payload.appid}'` .

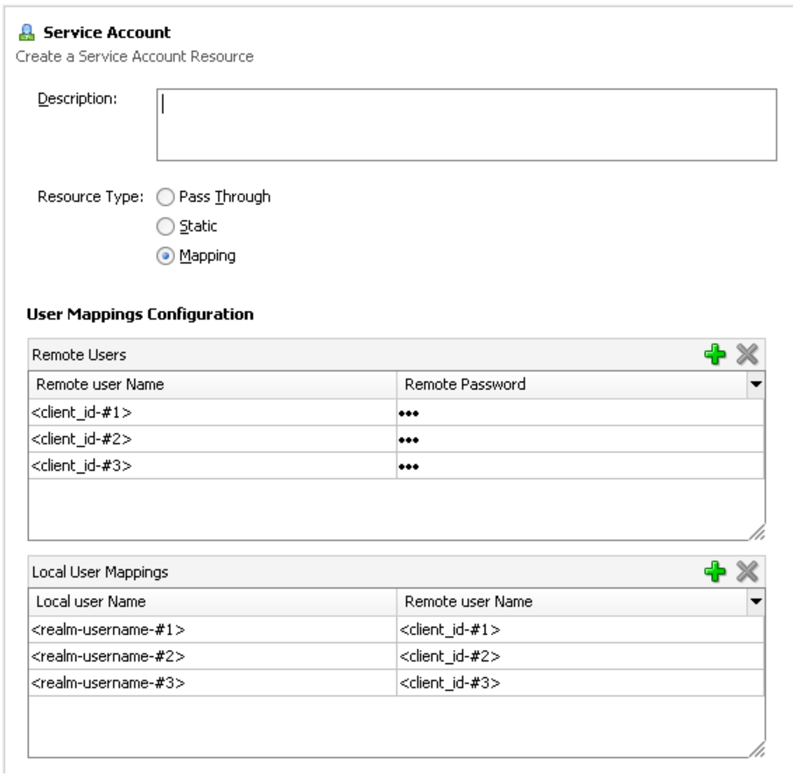
4. Combined identity

It is possible to combine scenarios 2 and 3 to strengthen security, maintaining the management of the mapping on the OSB and at the same time forcing the use of a claim by verifying that it corresponds to the mapped user. In this way, you also get the benefit of forcing OAUTH2 app-registrations dedicated to use with the OSB, avoiding that identities already used in other contexts are recycled. This scenario can be implemented by leveraging the VALIDATION_ASSERTION parameter to force this verification.

For example, you can configure the `VALIDATION_ASSERTION` parameter with a simple script like this: `'${token.payload.sub}'=='${username}'` .

How to configure Mapping Service Account

The OSB resource used for translating the JWT token identity into the weblogic realm username must be a "Service Account" of type "mapping" and must be handled as highlighted in the following screenshot. Please note that it is not necessary to fill in the "Remote Password" field, which can be filled with arbitrary text.



Service Account

Create a Service Account Resource

Description:

Resource Type: Pass Through [✓ Mapping] Static

User Mappings Configuration

Remote Users

Remote user Name	Remote Password
<client_id-#1>	•••
<client_id-#2>	•••
<client_id-#3>	•••

Local User Mappings

Local user Name	Remote user Name
<realm-username-#1>	<client_id-#1>
<realm-username-#2>	<client_id-#2>
<realm-username-#3>	<client_id-#3>

Build instructions

The sources can be compiled with any Java IDE with Ant support but you need to prepare the necessary dependencies for WebLogic and Oracle Service Bus libraries. You only need to modify "javaHomeDir" and "weblogicDir" in "Build.xml" file to suit your environment. The file supports multiple terget already prepared for WebLogic 12.1.3, 12.2.1 and 14.1.2 on a Windows operating system. Here is an excerpt of the section that needs to be customized.

```
<switch value="${targetConfig}">
  <case value="12.1.3">
    ...
    <property name="javaHomeDir" value="C:/Programmi/Java/jdk1.7"/>
    <property name="weblogicDir" value="C:/Oracle/Middleware/12.1.3"/>
    ...
  </case>
  <case value="12.2.1">
    ...
    <property name="javaHomeDir" value="C:/Programmi/Java/jdk1.8"/>
    <property name="weblogicDir" value="C:/Oracle/Middleware/12.2.1"/>
    ...
  </case>
  <case value="14.1.2">
    ...
    <property name="javaHomeDir" value="C:/Programmi/Java/jdk17"/>
    <property name="weblogicDir" value="C:/Oracle/Middleware/14.1.2"/>
    ...
  </case>
  <default>
    <fail message="Unsupported target: ${targetConfig}"/>
  </default>
</switch>
```

The repository contains three projects already prepared for JDeveloper 12.1.3, 12.2.1.4 and 14.1.2 installation on Windows operating system. You could install JDeveloper with respective versions of Oracle SOA Suite Quick Start for Developers (see references). Ant compilation can be triggered from JDeveloper by right-clicking on the "Build.xml" file and selecting the "all" target or from the command line by running the "Build-xxx.cmd" Windows batch. Note that cross-compilation is fully supported, meaning that you can compile the provider for a different version target than JDeveloper, provided that at least the dependency libraries are accessible and configured correctly in the Ant build targets.

In both cases, at the end of the compilation, one or two jar archives are produced (depending on the Ant option for merging libraries) which are automatically copied into the `<WEBLOGIC_HOME>/wlserver/server/lib/mbeantypes` folder from which WebLogic loads the security providers at startup. At the end of the compilation, you can directly launch the WebLogic environment integrated into JDeveloper to test the provider's operation.

Log Management

The log messages generated by the provider follow the following format: `<timestamp> <module> <sequence> <level> <message>` .
Let's see below the format of each token :

Token	Format
`<timestamp>`	Timestamp with milliseconds resolution in the format 'yyyy-MM-dd HH:mm:ss.SSS'.
`<module>`	Identifies log messages generated by the provider and is always equal to 'CIAx' where 'x' is a counter incremented for each running instance of the provider.
`<sequence>`	Numeric sequence incremented at each request handled by the provider.
`<level>`	Log message severity level (TRACE,DEBUG,INFO,WARN,ERROR).
`<message>`	Message text

The provider generates logs in its own format, with different levels of attention depending on the type of information. The logging level to filter messages is defined by the LOGGING_LEVEL parameter. Below is an almost exhaustive example of the various types of log messages generated by the provider, both during initialization (weblogic boot) and at each request.

1. Initialization Phase

```
... <INFO>  ##########################################################################################
... <INFO>  INITIALIZE
... <INFO>  ##########################################################################################
... <INFO>  ==========================================================================================
... <INFO>  CONTEXT
... <INFO>  ==========================================================================================
... <INFO>  Realm Name ...........: myrealm
... <INFO>  Domain Name ..........: DefaultDomain
... <INFO>  Managed Name .........: DefaultServer
... <INFO>  Provider Name ........: AzureIdentityAsserter
... <INFO>  Instance Name ........: CIA0
... <INFO>  ------------------------------------------------------------------------------------------
... <INFO>  JWT Provider .........: org.falpi.utils.jwt.JWTProviderNimbusShadedImpl
... <INFO>  Kerberos Config ......: C:\Temp\krb5-4823829319811767945.conf
... <INFO>  Scripting Engine .....: Oracle Nashorn (1.8.0_391)
... <INFO>  Scripting Language ...: ECMAScript (ECMA - 262 Edition 5.1)
... <INFO>  ##########################################################################################
```

2. Identity Assertion Phase

Below is an example of the logs generated by a JWT authenticated request against Azure Entra ID provider, with an example of using the "DEBUGGING_PROPERTIES" to inspect all http headers and JWT token attributes.

```
... <DEBUG> ##########################################################################################
... <DEBUG> ASSERT IDENTITY
... <DEBUG> ##########################################################################################
... <DEBUG> ==========================================================================================
... <DEBUG> CONFIGURATION
... <DEBUG> ==========================================================================================
... <DEBUG> LOGGING_LEVEL ................: TRACE
... <DEBUG> LOGGING_LINES ................: 5
... <DEBUG> LOGGING_INFO .................: Proxy: ${osb.service.name}, User: ${username} (${identity}), Client:  ${http.client.host}
... <DEBUG> THREADING_MODE ...............: PARALLEL
... <DEBUG> BASIC_AUTH ...................: ENABLE
... <DEBUG> JWT_AUTH .....................: ENABLE
... <DEBUG> JWT_KEYS_URL .................: https://login.microsoftonline.com/common/discovery/keys
... <DEBUG> JWT_KEYS_FORMAT ..............: JSON
... <DEBUG> JWT_KEYS_MODULUS_XPATH .......: //keys[kid='${token.header.kid}']/n
... <DEBUG> JWT_KEYS_EXPONENT_XPATH ......: //keys[kid='${token.header.kid}']/e
... <DEBUG> JWT_KEYS_CACHE_TTL ...........: 3600
... <DEBUG> JWT_KEYS_CONN_TIMEOUT ........: 5
... <DEBUG> JWT_KEYS_READ_TIMEOUT ........: 5
... <DEBUG> JWT_KEYS_SSL_VERIFY ..........: DISABLE
... <DEBUG> JWT_KEYS_HOST_AUTH_MODE ......: ANONYMOUS
... <DEBUG> JWT_KEYS_HOST_ACCOUNT_PATH ...:
... <DEBUG> JWT_KEYS_PROXY_SERVER_MODE ...: ANONYMOUS
... <DEBUG> JWT_KEYS_PROXY_SERVER_PATH ...: System/Proxy Servers/PROXY_Default
... <DEBUG> JWT_IDENTITY_MAPPING_MODE ....: ACCOUNT
... <DEBUG> JWT_IDENTITY_MAPPING_PATH ....: TEST/Mapper
... <DEBUG> JWT_IDENTITY_ASSERTION .......: '${token.payload.appid}'
... <DEBUG> VALIDATION_ASSERTION .........:
... <DEBUG> CUSTOM_REQUEST_HEADERS .......:
... <DEBUG> CUSTOM_RESPONSE_HEADERS ......: custom-tracking=${instance}:${wls.managed}:${counter}
... <DEBUG> DEBUGGING_ASSERTION ..........:
... <DEBUG> DEBUGGING_PROPERTIES .........: ${http.header.*},${token.header.*},${token.payload.*}
... <TRACE> ------------------------------------------------------------------------------------------
... <TRACE> Checking Parameter JWT_KEYS_URL: class 'String'
... <TRACE> Checking Parameter JWT_KEYS_MODULUS_XPATH: class 'String'
... <TRACE> Checking Parameter JWT_KEYS_EXPONENT_XPATH: class 'String'
... <TRACE> Checking Parameter JWT_KEYS_CACHE_TTL: class 'Integer'
... <TRACE> Checking Parameter JWT_KEYS_CONN_TIMEOUT: class 'Integer'
... <TRACE> Checking Parameter JWT_KEYS_READ_TIMEOUT: class 'Integer'
... <TRACE> Checking Parameter JWT_KEYS_PROXY_SERVER_MODE: class 'String'
... <TRACE> Checking Parameter JWT_IDENTITY_MAPPING_MODE: class 'String'
... <TRACE> Checking Parameter JWT_IDENTITY_ASSERTION: class 'String[]'
... <DEBUG> ==========================================================================================
... <DEBUG> CONTEXT
... <DEBUG> ==========================================================================================
... <DEBUG> Managed Name .....: DefaultServer
... <DEBUG> Project Name .....: TEST
... <DEBUG> Service Name .....: PS_TEST_Test_1.0
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> Server Host ......: sisnet-svil.local
... <DEBUG> Server Addr ......: 127.0.0.1
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> Client Host ......: sisnet-svil.local
... <DEBUG> Client Addr ......: 127.0.0.1
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> Request URL ......: http://127.0.0.1:7101/TEST/Test
... <DEBUG> Content Type .....: application/xml
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> Selected Token ...: CIA.JWT+BASIC
... <DEBUG> Detected Auth ....: JWT
... <DEBUG> ==========================================================================================
... <DEBUG> JWT AUTH
... <DEBUG> ==========================================================================================
... DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> KEYS RETRIEVE
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> Proxy Server Resource .........: System/Proxy Servers/PROXY_Default
... <DEBUG> Proxy Server Host .............: 127.0.0.1
... <DEBUG> Proxy Server Port .............: 3128
... <TRACE> Payload (JSON) ................: {"keys":[{"kty":"RSA","cloud_instance_name":...
... <TRACE> Payload (XML) .................: <root><keys><kty>RSA</kty><cloud_instance_name>...
... <DEBUG> Modulus XPath Parsed ..........: //keys[kid='CNv0OI3RwqlHFEVnaoMAshCH2XE']/n
... <DEBUG> Exponent XPath Parsed .........: //keys[kid='CNv0OI3RwqlHFEVnaoMAshCH2XE']/e
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> TOKEN VALIDATION
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> Key ID .........: CNv0OI3RwqlHFEVnaoMAshCH2XE
... <TRACE> Key Modulus ....: hz6fUSCSAuiyQz6L1nQj4za8kItevJzxhVbecMigTIl9pXZSHZa3gzMgtapnb1....
... <TRACE> Key Exponent ...: AQAB
... <DEBUG> Validation .....: true
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> IDENTITY ASSERTION
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> Token Identity ....: <client_id>
... <DEBUG> Mapping Account ...: TEST/Mapper
... <DEBUG> Realm UserName ....: falpi
... <DEBUG> ==========================================================================================
... <DEBUG> CUSTOM HEADERS
... <DEBUG> ==========================================================================================
... <DEBUG> Response: custom-tracking=CIA0:DefaultServer:0000003
... <DEBUG> ==========================================================================================
... <DEBUG> DEBUGGING PROPERTIES
... <DEBUG> ==========================================================================================
... <DEBUG> ${http.header.*} =>
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> Content-Type ......: application/xml
... <DEBUG> User-Agent ........: PostmanRuntime/7.30.0
... <DEBUG> Accept ............: */*
... <DEBUG> Host ..............: 127.0.0.1:7101
... <DEBUG> Accept-Encoding ...: gzip, deflate, br
... <DEBUG> Connection ........: keep-alive
... <DEBUG> Content-Length ....: 177
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> ${token.header.*} =>
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> x5t ...: "CNv0OI3RwqlHFEVnaoMAshCH2XE"
... <DEBUG> kid ...: "CNv0OI3RwqlHFEVnaoMAshCH2XE"
... <DEBUG> typ ...: "JWT"
... <DEBUG> alg ...: "RS256"
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> ${token.payload.*} =>
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> aud ...................: "00000002-0000-0000-c000-000000000000"
... <DEBUG> iss ...................: "https://sts.windows.net/<tenant_id>/"
... <DEBUG> iat ...................: 1743975300
... <DEBUG> nbf ...................: 1743975300
... <DEBUG> exp ...................: 1743979200
... <DEBUG> aio ...................: "<omissis>"
... <DEBUG> appid .................: "<client_id>"
... <DEBUG> appidacr ..............: "1"
... <DEBUG> idp ...................: "https://sts.windows.net/<tenant_id>/"
... <DEBUG> idtyp .................: "app"
... <DEBUG> oid ...................: "<omissis>"
... <DEBUG> rh ....................: "<omissis>"
... <DEBUG> sub ...................: "<omissis>"
... <DEBUG> tenant_region_scope ...: "EU"
... <DEBUG> tid ...................: "<omissis>"
... <DEBUG> uti ...................: "<omissis>"
... <DEBUG> ver ...................: "1.0"
... <DEBUG> xms_acd ...............: 1728384169
... <DEBUG> xms_ftd ...............: "<omissis>"
... <DEBUG> xms_idrel .............: "7 18"
... <DEBUG> xms_tdbr ..............: "EU"
... <DEBUG> ------------------------------------------------------------------------------------------
... <DEBUG> ##########################################################################################
... <INFO>  Inbound (JWT) => Proxy: PS_TEST_Test_1.0, User: falpi (<client_id>), Client:  <hostname> (127.0.0.1)
```

Threading Mode

The provider code base was designed and tested to be thread-safe because Identity Asserters in WebLogic can be called in parallel and this is their normal behavior. If multiple requests arrive at the same time the server allocates a different thread for each assertion. Load tests were done with the excellent SoapUI tool reaching up to 1000 threads for parallel requests and the provider code was found to be solid and without memory leaks.

However there may be situations where it is useful to force serialization of requests and this is the purpose of the "THREADING_MODE" configuration parameter. When "SERIAL" mode is selected a "synchronized" version of the assertion method is used and this causes multiple parallel requests to be queued serially, without overlapping.

This could be useful for example for analyzing debug logs of a specific service in the presence of a large number of requests. In "PARALLEL" mode the log lines of each request/thread would be mixed with those of others. In "SERIAL" mode instead each assertion execution completes atomically with a consistent footprint of its logs.

Known Issues

1. Invalid signature with Azure Entra ID

```
com.nimbusds.jose.proc.BadJWSException: Signed JWT rejected: Invalid signature
```

The provider uses the excellent Nimbus library for verifying JWT tokens. Please note that using Azure Entra ID as IDP there is a known issue with Microsoft Token API v2. In this case there are some prerequisites to follow when requesting an access token. See the following articles:

- https://stackoverflow.com/questions/71306470/cant-verify-access-token-signature-from-azure-using-nimbus
- https://stackoverflow.com/questions/75535497/azure-oauth2-cant-validate-access-token

Credits

- JSON-java (https://github.com/stleary/JSON-java)
- Mozilla Rhino (https://github.com/mozilla/rhino)
- Apache HttpClient (https://hc.apache.org/httpcomponents-client-4.5.x/index.html)
- Nimbus JOSE + JWT (https://connect2id.com/products/nimbus-jose-jwt)

References

- Installing Oracle SOA Suite Quick Start for Developers 12.1.3: https://docs.oracle.com/middleware/1213/soasuite/index.html
- WebLogic 12.1.3 Identity Assertion Providers: https://docs.oracle.com/middleware/1213/wls/DEVSP/ia.htm
- WebLogic 12.1.3 Security Providers Developer Guide: https://docs.oracle.com/middleware/1213/wls/DEVSP/DEVSP.pdf
- Generate an MBean Type Using the WebLogic MBeanMaker: https://docs.oracle.com/middleware/1213/wls/DEVSP/generate_mbeantype.htm
- Sample Custom Identity Asserter for Weblogic Server: https://weblogic-wonders.com/simple-sample-custom-identity-asserter-weblogic-server-12c/
- Sample Custom SSO using Weblogic IdentityAsserter: https://virtual7.de/en/blog/custom-sso-using-weblogic-identityasserter