Configuring Inbound Authentication for a Service Provider

The responsibility of the inbound authenticators component is to identify and parse all the incoming authentication requests and then build the corresponding response. A given inbound authenticator has two parts:

- Request Processor
- Response Builder

For each protocol supported by the WSO2 Identity Server, there should be an inbound authenticator. The Identity Server includes inbound authenticators for SAML 2.0, OpenID, OpenID Connect, OAuth 2.0, Kerberos KDC, WS-Trust STS and WS-Federation (passive). The responsibility of the SAML 2.0 request processor is to accept a SAML request from a service provider, validate the SAML request and then build a common object model understood by the authentication framework and handover the request to it. The responsibility of the SAML response builder is to accept a common object model from the authentication framework and build a SAML response out of it.

Both the request processors and the response builders are protocol aware, while the authentication framework is not coupled to any protocol. See Architecture for more information on the complete flow where inbound authenticators come into play.

You can configure the following for inbound authentication.

1. Expand the SAML2 Web SSO Configuration and click Configure.
2. Fill in the form that appears.

![New Service Provider Form]

3. Click **Register**.

The following points should be taken into consideration when filling the above **New Service Provider** form.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issuer</td>
<td><strong>Issuer.</strong> This is the <code>&lt;saml:Issuer&gt;</code> element that contains the unique identifier of the service provider. This is also the issuer value specified in the SAML Authentication Request issued by the service provider. When configuring single-sign-on across Carbon servers, ensure that this value is equal to the <strong>ServiceProviderID</strong> value mentioned in the <code>&lt;IS_HOME&gt;/repository/conf/security/authenticators.xml</code> file of the relying party Carbon server.</td>
<td>travelocity.com</td>
</tr>
<tr>
<td>Assertion Consumer URLs</td>
<td><strong>Assertion Consumer URLs.</strong> This is the URL to which the browser should be redirected after the authentication is successful. This is the Assertion Consumer Service (ACS) URL of the service provider. The identity provider redirects the SAML2 response to this ACS URL. However, if the SAML2 request is signed and SAML2 request contains the ACS URL, the Identity Server will honor the ACS URL of</td>
<td><a href="http://localhost:8080/travelocity.com">http://localhost:8080/travelocity.com</a></td>
</tr>
</tbody>
</table>
the SAML2 request. It should have this format: https://(host-name):(port)/acs. You can add multiple assertion consumer URLs for the service provider by entering the URL and clicking the Add button.

**Default Assertion Consumer URL** Since there can be multiple assertion consumer URLs, you must define a Default Assertion Consumer URL in case you are unable to retrieve it from the authentication request.

---

**Tip:** In a service provider initiated single sign-on setup, the following needs to be considered.

- If no ACS URL is given in the <AuthnRequest>, the Identity Server sends the response to the default ACS URL of the service provider (whether the request is signed or not).
- If the ACS URL in <AuthnRequest> matches with one of the registered URLs, the Identity Server sends the response to the matched one.
- If the ACS URL in <AuthnRequest> does not match any of the registered ACS URLs and if the request is signed, the Identity Server sends the response to the ACS URL in the request only if the signature is valid. Alternatively, the <AuthnRequest> is rejected.

In an identity provider initiated single sign-on setup, the following needs to be considered.

- If the “acs” query parameter is not present in the request, the Identity Server sends the response to default ACS URL of the service provider.
- If the “acs” parameter is present and the value of that parameter matches with any of the registered ACS URLs of the service provider, then the Identity Server sends the response to the matched one.

---

**NameID format** Specify the NameID format. This defines the name identifier formats supported by the identity provider. The service provider and identity provider usually communicate with each other regarding a specific subject. That subject should be identified through a Name-Identifier (NameID), which should be in some format so that it is easy for the other party to identify it based on the format. Name identifiers are used to provide information regarding a user.

---

**About NameID formats**

For SSO interactions, you can use the following types of NameID formats.

- urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
- urn:oasis:names:tc:SAML:2.0:nameid-format:transient
- urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
- urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified
- urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName
- urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName
- urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos
- urn:oasis:names:tc:SAML:2.0:nameid-format:entity

This specifies the name identifier format that the Identity Server wants to receive in the subject of an assertion from a particular identity provider. The following is the default format used by the identity provider.
<table>
<thead>
<tr>
<th>Certificate Alias</th>
<th>Select the <strong>Certificate Alias</strong> from the dropdown. This is used to validate the signature of SAML2 requests and is used to generate encryption. Basically the service provider's certificate must be selected here. Note that this can also be the Identity Server tenant's public certificate in a scenario where you are doing a tenant specific configuration.</th>
<th>wso2carbon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Signing Algorithm</td>
<td>Specifies the ‘SignatureMethod’ algorithm to be used in the ‘Signature’ element in POST binding. The default value can be configured in the <code>&lt;IS_HOME&gt;/repository/conf/identity.xml</code> file, in the <code>SSOService element with SAMLDefaultSignatureAlgorithmURI</code> tag. If it is not provided the default algorithm is RSA-SHA 1, at URI <code>http://www.w3.org/2000/09/xmldsig#rsa-sha1</code>.</td>
<td><a href="http://www.w3.org/2000/09/xmldsig#rsa-sha1">http://www.w3.org/2000/09/xmldsig#rsa-sha1</a></td>
</tr>
<tr>
<td>Response Digest Algorithm</td>
<td>Specifies the ‘DigestMethod’ algorithm to be used in the ‘Signature’ element in POST binding. The default value can be configured in the <code>&lt;IS_HOME&gt;/repository/conf/identity.xml</code> file, in the <code>SSOService element with SAMLDefaultSignatureAlgorithmURI</code> tag. If it is not provided the default algorithm is SHA 1, at URI <code>http://www.w3.org/2000/09/xmldsig#sha1</code>.</td>
<td><a href="http://www.w3.org/2000/09/xmldsig#sha1">http://www.w3.org/2000/09/xmldsig#sha1</a></td>
</tr>
<tr>
<td>Enable Response Signing</td>
<td>Select <strong>Enable Response Signing</strong> to sign the SAML2 Responses returned after the authentication process.</td>
<td>Selected</td>
</tr>
<tr>
<td>Enable Signature Validation in Authentication Requests and Logout Requests</td>
<td>Select <strong>Enable Signature Validation in Authentication Requests and Logout Requests</strong> if you need this functionality configured. This specifies whether the identity provider must validate the signature of the SAML2 authentication request and the SAML2 logout request that are sent by the service provider.</td>
<td>Selected</td>
</tr>
<tr>
<td>Enable Assertion Encryption</td>
<td>Enable <strong>Assertion Encryption</strong>, if you wish to encrypt the assertion.</td>
<td>Selected</td>
</tr>
<tr>
<td>Enable Single Logout</td>
<td>Select <strong>Enable Single Logout</strong> so that all sessions are terminated once the user signs out from one server. If single logout is enabled, the identity provider sends logout requests to all service providers. Basically, the identity provider acts according to the single logout profile. If the service provider supports a different URL for logout, you can enter a <strong>SLO Response URL</strong> and <strong>SLO Request URL</strong> for logging out. These URLs indicate where the request and response should go to. If you do not specify this URL, the identity provider uses the Assertion Consumer Service (ACS) URL.</td>
<td>Selected</td>
</tr>
<tr>
<td>Enable Attribute Profile</td>
<td>Select <strong>Enable Attribute Profile</strong> to enable this and add a claim by entering the claim link and clicking the <strong>Add Claim</strong> button. The Identity Server provides support for a basic attribute profile where the identity provider can include the user's attributes in the SAML Assertions as part of the attribute statement. Once you select the checkbox</td>
<td>Selected</td>
</tr>
</tbody>
</table>
Include Attributes in the Response Always, the identity provider always includes the attribute values related to the selected claims in the SAML attribute statement.

| Enable Audience Restriction | Select Enable Audience Restriction to restrict the audience. You may add audience members using the Audience text box and clicking the Add button. | Selected |
| Enable Recipient Validation | Select this if you require validation from the recipient of the response. | Selected |
| Enable IdP Initiated SSO | Select the Enable IdP Initiated SSO checkbox to enable this functionality. When this is enabled, the service provider is not required to send the SAML2 request. | Selected |
| Enable IdP Initiated SLO | Select the Enable IdP Initiated SLO checkbox to enable this functionality. You must specify the URL. | Selected |

### Additional configurations

- The following table provides the list of signature algorithms available and their respective URI.

<table>
<thead>
<tr>
<th>Signature algorithm name</th>
<th>Signature algorithm URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSA with SHA1</td>
<td><a href="http://www.w3.org/2000/09/xmldsig#dsasha1">http://www.w3.org/2000/09/xmldsig#dsasha1</a></td>
</tr>
<tr>
<td>ECDSA with SHA1</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#ecdsasha1">http://www.w3.org/2001/04/xmldsigmore#ecdsasha1</a></td>
</tr>
<tr>
<td>ECDSA with SHA256</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#ecdsasha256">http://www.w3.org/2001/04/xmldsigmore#ecdsasha256</a></td>
</tr>
<tr>
<td>ECDSA with SHA384</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#ecdsasha384">http://www.w3.org/2001/04/xmldsigmore#ecdsasha384</a></td>
</tr>
<tr>
<td>ECDSA with SHA512</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#ecdsasha512">http://www.w3.org/2001/04/xmldsigmore#ecdsasha512</a></td>
</tr>
<tr>
<td>RSA with MD5</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#rsamd5">http://www.w3.org/2001/04/xmldsigmore#rsamd5</a></td>
</tr>
<tr>
<td>RSA with RIPEMD160</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#rsaripemd160">http://www.w3.org/2001/04/xmldsigmore#rsaripemd160</a></td>
</tr>
<tr>
<td>RSA with SHA1</td>
<td><a href="http://www.w3.org/2000/09/xmldsig#rsasha1">http://www.w3.org/2000/09/xmldsig#rsasha1</a></td>
</tr>
<tr>
<td>RSA with SHA256</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#rsasha256">http://www.w3.org/2001/04/xmldsigmore#rsasha256</a></td>
</tr>
<tr>
<td>RSA with SHA384</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#rsasha384">http://www.w3.org/2001/04/xmldsigmore#rsasha384</a></td>
</tr>
<tr>
<td>RSA with SHA512</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#rsasha512">http://www.w3.org/2001/04/xmldsigmore#rsasha512</a></td>
</tr>
</tbody>
</table>

- The following table provides the list of digest algorithms available and their respective URI.

<table>
<thead>
<tr>
<th>Digest algorithm name</th>
<th>Digest algorithm URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td><a href="http://www.w3.org/2001/04/xmldsigmore#md5">http://www.w3.org/2001/04/xmldsigmore#md5</a></td>
</tr>
</tbody>
</table>
### Related Topics

See [SAML 2.0 Web SSO](#) for more information on SAML2 single-sign-on and see the following topics for samples of configuring single-sign-on using SAML2.

- Configuring Single Sign-On
- Logging in to WSO2 Products via Identity Server
- Configuring SAML2 Single-Sign-On Across Different WSO2 Products

See [Using the SAML2 Toolkit](#) for support on debugging issues with SAML2 configurations.

To enable OAuth support for your client application, you must first register your application by providing an application name. For code and implicit grant types, you have to provide a callback URL. Follow the instructions below to add a new application.

1. Expand the **OAuth/OpenID Connect Configuration** and click **Configure**.
2. Fill in the form that appears. For the **Allowed Grant Types** you can disable the ones you do not require or wish to block.

When filling out the **New Application** form, the following details should be taken into consideration.

<table>
<thead>
<tr>
<th>Field</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth Version</td>
<td>Selecting <strong>OAuth Version</strong> as <strong>1.0a</strong> removes all the configurable <strong>Allowed Grant Types</strong>. This is because this version of OAuth does not support grant types.</td>
</tr>
<tr>
<td>Callback Url</td>
<td>This is the exact location in the service provider's application where an access token would be sent. This is a required field (if the grant type is anything other than 'Code' or 'Implicit') and it is important to configure, as it is imperative that the service provider receives the access token. This is necessary for security purposes to ensure that the token is not compromised.</td>
</tr>
</tbody>
</table>

**Configure multiple callback URLs**

From IS 5.2.0 onwards, regex based consumer URLs are supported when defining the callback URL. This enables you to configure multiple callback URLs for one application by entering a regex pattern as the value for the callback URL field. For example, if you have two service providers that use the same application, you can now define a regex pattern which will work for both callback URLs instead of having to configure two different applications for the two service providers.

Assume the two callback URLs for your two service providers are as follows:

- https://myapp.com/callback
- https://testapp:8000/callback

To configure the callback URL to work for **both** of these URLs, set it using a regex pattern as follows:

```plaintext
```

You must have the prefix `regexp=` before your regex pattern. To define a normal URL, you can specify the callback URL without this prefix.

You can also configure a regex pattern that contains dynamic values as seen below.

```plaintext
regexp=https://mchcon.clance.local\?id=(.*)
```
### Allowed Grant Types - The following are the grant types that are used to get the access token:

<table>
<thead>
<tr>
<th>Grant Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Entering the username and password required at the service provider will result in a code being generated. This code can be used to obtain the access token.</td>
</tr>
<tr>
<td>Implicit</td>
<td>This is similar to the code grant type, but instead of generating a code, this directly provides the access token.</td>
</tr>
<tr>
<td>Password</td>
<td>This authenticates the user using the password provided and the access token is provided.</td>
</tr>
<tr>
<td>Client Credential</td>
<td>This is the grant type for the client key and client secret. If these two items are provided correctly by the service provider, the access token is sent.</td>
</tr>
<tr>
<td>Refresh Token</td>
<td>This will enable the user to obtain an access token by using the refresh token once the originally provided access token is used up.</td>
</tr>
<tr>
<td>SAML</td>
<td>This uses SAML as the grant type to obtain the access token.</td>
</tr>
<tr>
<td>IWA-NTLM</td>
<td>This is similar to the password grant type, but it is specific to Microsoft Windows users.</td>
</tr>
<tr>
<td>PKCE Mandatory</td>
<td>Check this if you are using the Code grant type. PKCE is a recommended security measure used to mitigate a code interception attack. See Mitigating Authorization Code Interception Attacks for more information.</td>
</tr>
<tr>
<td>Support PKCE</td>
<td>Check this if you are using PKCE.</td>
</tr>
</tbody>
</table>

3. Click **Add**. The following information is added for your service provider.

- **OAuth Client Key** - This is the client key of the service provider, which will be checked for authentication by the Identity Server before providing the access token.
- **OAuth Client Secret** - This is the client secret of the service provider, which will be checked for authentication by the Identity Server before providing the access token. Click the **Show** button to view the exact value of this.

**Tip:** The OAuth client key and client secret are stored in plain text. To encrypt the client secret, access token and refresh token, do the following:

Open the `identity.xml` file found in the `<IS_HOME>/repository/conf/identity` directory and change the `<TokenPersistenceProcessor>` property as follows:

```xml
<TokenPersistenceProcessor>org.wso2.carbon.identity.oauth.tokenprocessor.EncryptionDecryptionPersistenceProcessor</TokenPersistenceProcessor>
```
### Related Topics

See Configuring OpenID Connect Single Logout to configure single logout or session management with OpenID Connect.

See Delegated Access Control for more information on working with OAuth2/OpenIDConnect. See the following topics for samples of configuring delegated access control:

- OAuth 2.0 with WSO2 Playground
- SAML2 Bearer Assertion Profile for OAuth 2.0 with WSO2 Travelocity

### Deprecated Feature!

OpenID 2.0 has been deprecated in this release (WSO2 Identity Server 5.2.0) as it is now an obsolete specification and has been superseded by OpenID Connect. It will be removed from the base product in the next release (WSO2 Identity Server 5.3.0) and will be moved to the store. Alternatively, we recommend that you use OpenID Connect instead.

1. Expand the **OpenID Configuration**.
2. Enter the identifier for the **OpenID Realm** as specified in the form below. This identifier need to be specified as identification when the service provider reaches out to the Identity Server.

[Image of OpenID Configuration]

WSO2 Identity Server's passive security token service (Passive STS) is used as the **WS-Federation** implementation. The Passive STS is capable of issuing SAML 1.1 and 2.0 security tokens.

To request a SAML 2.0 security token, the Request Security Token (RST) should be sent to the passive STS endpoint with the TokenType 'SAMLV2.0' when sending the token request. If there is no RST specified, the WSO2 Identity Server will issue a SAML 1.1 token by default.

### Configuring passive STS

1. See here for details on adding a service provider.
2. Expand the **Inbound Authentication Configuration** followed by the **WS-Federation (Passive)** Configuration section and provide the following values. See Configuring WS-Federation (Passive) for more information.
   - **Passive STS Realm** - Provide the same realm name given to the web app you are configuring WS-Federation for.
   - **Passive STS WReply URL** - Provide the URL of the web app you are configuring WS-Federation for. This endpoint URL will handle the token response.

[Tip]

If you want to configure an expiration time for the security token, you need to add the following configuration in the `<IS_HOME>/repository/conf/carbon.xml` file, under the `<Server>` element:

```xml
<Server>
  ...
  <ServerSecurity>
    ...
    <SecurityTokenService>
      ...
      <TTL>1800</TTL>
      ...
    </SecurityTokenService>
    ...
  </ServerSecurity>
  ...
</Server>
```
<STSTimeToLive>1800000</STSTimeToLive>

Here, the expiration time should be specified in milliseconds.

3. Expand the Claim Configuration section and map the relevant claims. See Configuring Claims for a Service Provider for more information.
4. Click Update to save changes.

Currently the signing algorithm used for passive STS by default is rsa-sha1 and the digest algorithm used is sha1. To change the default algorithms, add the following configuration under the <security> tag in the carbon.xml file found in the <IS_HOME>/repository/conf directory. The example given below sets the signing algorithm to rsa-sha256 and the digest algorithm to sha256.

```xml
<STSSignatureAlgorithm>http://www.w3.org/2001/04/xmldsig-more#rsa-sha256</STSSignatureAlgorithm>
<STSDigestAlgorithm>http://www.w3.org/2001/04/xmlenc#sha256</STSDigestAlgorithm>
```

To configure this, apply the 4665 WUM update to WSO2 IS 5.6.0 using the WSO2 Update Manager (WUM). To deploy a WUM update into production, you need to have a paid subscription. If you do not have a paid subscription, you can use this feature with the next version of WSO2 Identity Server when it is released. For more information on updating WSO2 Identity Server using WUM, see Getting Started with WUM in the WSO2 Administration Guide.

Related Topics

- To test out WSO2 Identity Server's passive security token service using a sample, see Testing Identity Server's Passive STS.
WSO2 Identity Server's security token service (STS) is used as the WS-Trust implementation. The STS is capable of issuing SAML 1.1 and 2.0 security tokens and has a SOAP/XML API for token issuance. This API can be secured with the UserNameToken or with any other WS-Security mechanism.

Configuring STS

STS is configured under the Resident Identity Provider section of the Identity Server Management Console. Use the following step to do the configurations.

1. Configure the Resident Identity Provider. See here for more detailed information on how to do this.
2. In the Resident Identity Provider page, expand the Inbound Authentication Configuration section along with the Security Token Service Configuration section.
3. Click Apply Security Policy.


5. Click Next.
6. In the resulting page, select the **admin** checkbox and click **Finish**.

![Activate Security](image)

7. Click **Ok** on the confirmation dialog window that appears and click **Update** to complete the process.

Now STS is configured and secured with a username and password. Only users with the Admin role can consume the service.

The next step is to add a service provider to consume the STS.

**Adding a service provider for the STS client**

1. See [here](#) for details on adding a service provider.
2. Expand the **Inbound Authentication Configuration** section and the **WS-Trust Security Token Service Configuration** section. Click **Configure**.

3. In the resulting screen, enter the **Endpoint Address**. This must be used as the service URL and the token is delivered by the STS client.

4. Click **Update** to save the changes made to the service provider.

**Related Topics**

- After configuring the service provider, you need to run the STS client. See **Running an STS Client** to try out a sample STS client.
Related Topics

See Single Sign-On for details on configuring single sign-on for service provider using inbound authentication. See the following topics for samples of configuring single sign-on:

- Configuring Single Sign-On
- Logging in to WSO2 Products via Identity Server