Implementing SAML with identity federation for service aggregation in WebSphere Commerce

Skill Level: Intermediate

Michael D.J. Shields (mshields@ca.ibm.com)
Solution Architect
IBM

23 May 2007

This tutorial provides WebSphere® Commerce best practices on implementing Security Assertion Markup Language (SAML) with identity federation for service aggregation.

Section 1. Before you start

In this tutorial, you will review the WebSphere Commerce best practice approach for implementing SAML with identity federation between external applications. You will learn how SAML assertions can compliment a service-oriented architecture (SOA) initiative by providing significant flexibility for identity federation management. Federation refers to the establishment of some or all of user identifiers across security and policy domains to enable seamless cross-domain business interactions. In this case, each domain is isolated from the details of other domains' authentication and authorization infrastructure.

Section 2. Introduction

Emerging demands require solutions for employee single sign-on to enterprise partners, business-to-business single sign-on, and business-to-consumer account aggregation.
There are opportunities to streamline and improve how applications authenticate their customers and employees, while also providing “transparent and cohesive” access to internal and external network resources. This tutorial shows how WebSphere Commerce can leverage some best practices when implementing SAML with identity federation for service aggregation.

What is SAML?

SAML is an XML standard for exchanging authentication and authorization data between security domains; that is, between an identity provider and a service provider. The purpose of using SAML in WebSphere Commerce solutions is to allow centralized, decentralized, and federated systems to interoperate. Interoperability is an important factor in the success of solutions that are based on Web services and service-oriented architecture (SOA). SAML is a product of the OASIS Security Services Technical Committee, which went through many revisions to ensure a definitive standard. Thus, to keep your architecture clean, you can leverage the SAML standard to streamline your SOA goals with the most important problem that SAML is trying to solve. SAML tries to solve the Web identity federation problem where a specific proprietary token is not a dependency of decentralized deployments. The token, just like SAML itself, becomes standardized.

SAML architecture

The single sign on solution assumes the principal (often a user) has enrolled with at least one identity provider. This identity provider is expected to provide local authentication services to the principal. Hereafter, the principal can integrate with any number of given applications across any type of deployment through sharing a SAML assertion. The SAML assertion is proof that the principal did indeed authenticate with an identity provider.

Since all external applications can participate, each application agrees to accept a SAML assertion and defines the meaning of the statements contained within it. The identity provider first provides local authentication to the principal; however, from then on, the principal can travel to any other application in any environment without any proprietary restrictions. It is much the same as how a passport is handled. You first apply for a passport in your own country. Your home country has local regulations on how you receive that passport. Once you receive your passport, you are an authenticated citizen of your country. You can then travel to any other country in the world. Most countries do recognize more than just a passport for acceptance. Some countries that you visit accept your passport much like the basic SAML assertion. Other countries need more information in their acceptance, such as a passport and a Visa, which is a more sophisticated acceptance of a SAML assertion. There is a standardized approach to accepting the authentication of who you are that leads to the design of how the SAML assertion is represented.
Once you are accepted to another country, they do not need to manage your passport. You are still required to carry your passport with you. Much the same as a SAML assertion, it is an accepted number of user attributes or rights. However, each system manages that information in their own way and ensures loose coupling of identity federation. That is really how SAML provides interoperability. Each application your identity visits has their own flexibility to manage your session their own way.

Tutorial architecture

In this tutorial, you will review the WebSphere Commerce best practice approach for implementing SAML for usage with identity federation between external applications. You will learn how SAML assertions can compliment an SOA initiative by providing significant flexibility for federation in identity management. Federation refers to the establishment of some or all of user identifiers across security and policy domains to enable more seamless cross-domain business interactions. In this case, each domain is isolated from the details of the other domains' authentication and authorization infrastructure.

With technology such as a SiteMinder component used as the authentication mechanism, you can use WebSphere Commerce as the identity provider and service provider. Figure 1 illustrates identity federation for a service aggregation scenario - access to an external service with federated single sign-on.

Figure 1. Architecture: Identity federation for service aggregation
In this scenario, the user accesses the WebSphere Commerce application and the trust service authenticates the user, which generates a security token. A SAML assertion passes the user identity and the information required for external applications.

With SAML included in the architecture, the dependency on the traditional browser cookie is removed. That way, you can achieve cross-domain SSO. The SAML standard provides the means by which authentication and authorization assertions are exchanged between communicating parties. Identity management is simplified across organizational boundaries by allowing users to consolidate many local identities into a single federated identity.
Section 3. Overview

This tutorial uses the SAML 1.1 Browser/POST profile. The Browser/POST profile relies on a "push" operation, which passes a Single Sign On assertion by value. You do not need a back-channel communication as required in a Browser/Artifact Profile. The identity provider, WebSphere Commerce, "pushes" the assertion to the service provider.

Let's review the following four steps in context to handling, using best practices in WebSphere Commerce:

1. Request the Inter-Site Transfer Service
   The principal (user) requests the inter-site transfer service at the identity provider. Within WebSphere Commerce, the JSP can provide a target parameter that is passed along with the request to the Inter-Site Transfer Service controller command.

2. Respond with an HTML form
   The Inter-Site Transfer Service returns an HTML document containing a FORM element whose ACTION attribute is the URL of the assertion consumer service.

   Within WebSphere Commerce, the session is extracted by the controller command and sets the appropriate parameters of an Inter-Site Transfer Service Data Bean. The forwarding JSP extracts that data bean and sets the response into the form.

3. Request the assertion consumer service
   The client requests the assertion consumer service at the service provider. In this tutorial, the submission is automated using Javascript to forward the form action.

4. Respond to the Principal's Original Request
   The assertion consumer service inspects the SAML Response element, creates a security context on the service provider, and redirects the client to the target resource.

   The external application is responsible for handling this step.
Prerequisite

The tutorial steps will go through each of the four steps introduced above. Before starting this tutorial, identify the SAML code base that you will use. This tutorial uses the Verisign TSIK, Verisign XML Trust Services. Verisign XML Trust Services provide a JAR file to place into your classpath environment (for example, the lib directory) to be picked up by the code example in this tutorial.

The easiest way to include this JAR file into your environment is through publishing the SAR. This allows ease of development verification. From that point, you decide how to deploy the JAR into your production environment.

To include the JAR file into your environment, follow this tutorial as a guide: Leveraging the WebSphere Commerce store archive.

When you have extracted your SAR file into your file system, place the JAR into the WEB-INF/lib directory. If such a directory does not exist in your expanded SAR, then create it before packaging it up for republishing. The republishing steps are outlined in the guide provided above. Of course, you can also manually place the JAR file directly into your development or testing environment.

Section 4. Architecture

Figure 2 shows more details about the WebSphere Commerce runtime behavior.

Figure 2. Architecture: WebSphere Commerce SAML integration
Section 5. Steps

The user can be authenticated and authorized in WebSphere Commerce. If the user hits a URL that takes them outside of WebSphere Commerce to an external application, then you want them to go through the Inter-Site Transfer Service first.

1. Request the Inter-Site Transfer Service
   a. **Initial Controller Command**
      When making a request to hit a JSP page, you should follow best
practices and go through a controller command. The controller command will pull out the user information and send the user's credentials into session. When the controller command is finished, the user is then forwarded to the JSP page.

In your controller command, which is forwarding to a JSP page that provides links to an external application, add the following code:

Listing 1. Initial controller command

```
// BEGINNING OF SAML SESSION
String SAML_ATTRIBUTE_USER_CREDIT_ID = "user_credit";
String SAML_ATTRIBUTE_USER_DEBIT_ID = "user_debit";
String AUTH_USER_ID = "AUTH_USER_ID";
String AUTH_SUBJECT_IP = "AUTH_SUBJECT_IP";
String SAML_ATTRIBUTE_USER_NAME = "SAML_ATTRIBUTE_USER_NAME";
String SAML_ATTRIBUTE_USER_TYPE = "SAML_ATTRIBUTE_USER_TYPE";
javax.servlet.http.HttpSession session =
    ((com.ibm.commerce.webcontroller.HttpControllerRequestObject)
     (getCommandContext().getRequest()).getHttpRequest().getSession());
session.setAttribute(AUTH_SUBJECT_IP,getCommandContext().getRemoteHost()+"");
session.setAttribute(AUTH_USER_ID,logonId);
session.setAttribute(SAML_ATTRIBUTE_USER_CREDIT_ID,creditValue);
session.setAttribute(SAML_ATTRIBUTE_USER_DEBIT_ID,debitValue);
session.setAttribute(SAML_ATTRIBUTE_USER_NAME,userName);
session.setAttribute(SAML_ATTRIBUTE_USER_TYPE,userType);
```

b. View to provide URL

The controller command has forwarded the user to a JSP page. This page contains the URLs, which send the user to an external application within the same browser.

Within the JSP to which this controller command forwards, set the following URL parameter and hypertext link:

Listing 2. View to provide URL

```
<c:url var="SAMLURL" value="SwanInterSiteTransferService">
    <c:param name="langId" value="${langId}" />
    <c:param name="storeId" value="${storeId}" />
    <c:param name="catalogId" value="${catalogId}" />
    <c:param name="productId" value="${itemList.itemId}" />
    <c:param name="TARGET" value="${itemList.url}${fileName}" />
</c:url>

<a target="_blank" href="<c:out value="${SAMLURL}"/>">
    External App
</a>
```
c. When the user hits the URL, they are sent to the Inter Site Transfer Service mechanism. Because this follows WebSphere Commerce best practice, the controller command sets session parameters for confidential user information, whereas the JSP page is the view to handle the user interface to invoke the Inter Site Transfer Service request.

2. **Respond with an HTML form**

When responding with an HTML form, set the SAML assertion response as one of the form values. The Inter Site Transfer Service controller command is responsible for setting the FORM values, whereas the declaration of the FORM is handled in the JSP. This again is an example of allocation of responsibilities that exemplifies WebSphere Commerce best practices.

a. **Inter Site Transfer Service data bean**

The Inter Site Transfer Service data bean contains setters and getters particular for SAML assertion data. This bean is populated by the Inter Site Transfer Services controller command and forwarded to the Inter Site Transfer Service View.

**Listing 3. Inter Site Transfer Service data bean**

```java
public class SwanInterSiteDetailBean
extends SmartDataBeanImpl {

    public static final String COPYRIGHT =
            com.ibm.commerce.copyright.IBMCopyright.SHORT_COPYRIGHT;
    private String AUTH_USER_ID = null ;
    private String AUTH_SUBJECT_IP = null ;
    private String SAML_ATTRIBUTE_USER_CREDIT_ID = null ;
    private String SAML_ATTRIBUTE_USER_DEBIT_ID = null ;
    private String SAML_ATTRIBUTE_USER_NAME = null ;
    private String SAML_ATTRIBUTE_USER_TYPE = null;

    /**
     * @return
     */
    public String getAUTH_SUBJECT_IP() {
        return AUTH_SUBJECT_IP;
    }
    /**
     * @return
     */
    public String getAUTH_USER_ID() {
        return AUTH_USER_ID;
    }
    /**
     * @return
     */
    public String getSAML_ATTRIBUTE_USER_TYPE() {
        return SAML_ATTRIBUTE_USER_TYPE;
    }

    /**
     * @return
     */
    public String getSAML_ATTRIBUTE_USER_NAME() {
        return SAMLATTRIBUTE_USER_NAME;
    }
    /**
     * @return
     */
    public String getSAML_ATTRIBUTE_USER_CREDIT_ID() {
        return SAML_ATTRIBUTE_USER_CREDIT_ID;
    }
    /**
     * @return
     */
    public String getSAML_ATTRIBUTE_USER_DEBIT_ID() {
        return SAML_ATTRIBUTE_USER_DEBIT_ID;
    }
}
```
b. **Inter Site Transfer Service Controller Command**
The controller command below pulls the user information from the session and populates the Inter Site Transfer Service data bean. The data bean is then forwarded to the Inter Site Transfer Service View.

**Listing 4. Inter Site Transfer Service controller command**
public class SwanInterSiteTransferServiceCmdImpl
extends ControllerCommandImpl
implements SwanInterSiteTransferServiceCmd {

public static final String className =
"com.scs.commerce.manageemployees.commands.SwanInterSiteTransferServiceCmdImpl";

private String AUTH_USER_ID = null;
private String AUTH_SUBJECT_IP = null;
private String SAML_ATTRIBUTE_USER_CREDIT_ID = null;
private String SAML_ATTRIBUTE_USER_DEBIT_ID = null;
private String SAML_ATTRIBUTE_USER_NAME = null;
private String SAML_ATTRIBUTE_USER_TYPE = null;

public void performExecute() throws ECException {

String METHODNAME = "performExecute";
ECTrace.entry(ECTraceIdentifiers.COMPONENT_EXTERN,
this.getClass().getName(), "performExecute");

SwanInterSiteDetailBean bean = new SwanInterSiteDetailBean();
try {
    DataBeanManager.activate(bean, getCommandContext(),
(HttpServletResponse) getCommandContext().getResponse());
} catch(ECException e) {
    e.printStackTrace();
}
bean.setAUTH_USER_ID(AUTH_USER_ID);
bean.setAUTH_SUBJECT_IP(AUTH_SUBJECT_IP);
bean.setSAML_ATTRIBUTE_EXTERNAL_TAXPAYER_ID(SAML_ATTRIBUTE_USER_CREDIT_ID);
bean.setSAML_ATTRIBUTE_INTERNAL_TAXPAYER_ID(SAML_ATTRIBUTE_USER_DEBIT_ID);
bean.setSAML_ATTRIBUTE_USER_NAME(SAML_ATTRIBUTE_USER_NAME);
bean.setSAML_ATTRIBUTE_EXTERNAL_TAXPAYER_TYPE(SAML_ATTRIBUTE_USER_TYPE);

// set response property
responseProperties = new TypedProperty();
responseProperties.put("empireBean", bean);
responseProperties.put(ECConstants.EC_VIEWTASKNAME,
SWANConstants.VIEW_INTERSITE_TRANSFER_SERVICE);

ECTrace.exit(ECTraceIdentifiers.COMPONENT_EXTERN,
this.getClass().getName(), "performExecute");
}

public void setRequestProperties(TypedProperty reqParms)
throws ECApplicationException {

final String methodName = "setRequestProperties";
requestProperties = reqParms;
HttpSession session =
(((HttpControllerRequestObject)(commandContext.getRequest())).
getHttpRequest()).getSession();

AUTH_USER_ID = (String)session.getAttribute(sBean.AUTH_USER_ID);
AUTH_SUBJECT_IP = (String)session.getAttribute(sBean.AUTH_SUBJECT_IP);
SAML_ATTRIBUTE_USER_CREDIT_ID =
(String)session.getAttribute(sBean.SAML_ATTRIBUTE_USER_CREDIT_ID);
SAML_ATTRIBUTE_USER_DEBIT_ID =
(String)session.getAttribute(sBean.SAML_ATTRIBUTE_USER_DEBIT_ID);
SAML_ATTRIBUTE_USER_NAME =
(String)session.getAttribute(sBean.SAML_ATTRIBUTE_USER_NAME);
SAML_ATTRIBUTE_USER_TYPE =
(String)session.getAttribute(sBean.SAML_ATTRIBUTE_USER_TYPE);
}
public void validateParameters() throws ECException {
    final String methodName = "validateParameters";

    if (AUTH_USER_ID == null || AUTH_USER_ID.length()<1
        || AUTH_SUBJECT_IP == null
        || SAML_ATTRIBUTE_USER_CREDIT_ID == null
        || SAML_ATTRIBUTE_USER_DEBIT_ID == null
        || SAML_ATTRIBUTE_USER_NAME == null
        || SAML_ATTRIBUTE_USER_TYPE == null ) {
        TypedProperty hshNVPs = new TypedProperty();
        hshNVPs.put(ECConstants.EC_ERROR_CODE, "2132");
        throw new ECApplicationException(
            ECMessage._ERR_CMD_INVALID_PARAM,
            className, methodName,
            ECMessageHelper.generateMsgParms(
                "InterSiteTransfer: "),
            Constants.GENERIC_APPLICATION_ERROR,
            hshNVPs);
    }
}
}

The controller command above uses the generic methods contained within the following:

**setRequestProperties()**
This method pulls the SAML attributes from the session and sets the class parameters.

**validateParameters()**
This method ensures that all of the SAML attributes are available before forwarding to the Inter Site Transfer Service FORM.

**performExecute()**
This method populates the Inter Site Transfer Service data bean and forwards this bean as a typed property to the view.

c. **Inter Site Transfer Services view**
The Inter Site Transfer Service view is where the declaration of the FORM data takes place. This form data is posted to the external application. This methodology is using the SAML 1.1 Browser/POST profile. That way, the external application receives the FORM data and handles that request in any manner it so chooses. The following sections comprise of one single JSP page. This page is split into sections so that each is described in detail for better understanding.

i. **Special page imports**
The page imports are required to access the Verisign JAR file that is placed into the store level lib directory. In this tutorial, we referenced the Verisign JAR file from the JSP view to ease skills transfer and development. However, when deploying into production, you actually move this code into the controller command and simply populate the FORM with data contained in the Inter Site Transfer Service data bean.

**Listing 5. Special page imports**

```<%@ page import="java.util.Date"%>
<%@ page import="java.util.Calendar"%>
<%@ page import="com.ibm.ws.security.util.Base64Coder"%>
<%@ page import="com.verisign.saml.authorities.IDGenerator"%>
<%@ page import="com.verisign.saml.assertions.NameIdentifier"%>
<%@ page import="com.verisign.saml.assertions.SubjectConfirmation"%>
<%@ page import="com.verisign.saml.assertions.Subject"%>
<%@ page import="com.verisign.saml.XMLResponseGeneratorFactory"%>
<%@ page import="com.verisign.saml.authorities.ResponseGenerator"%>
<%@ page import="com.verisign.saml.protocol.Response"%>
<%@ page import="com.verisign.saml.protocol.Status"%>
<%@ page import="com.verisign.saml.protocol.StatusCode"%>
<%@ page import="com.verisign.saml.assertions.Assertion"%>
<%@ page import="com.verisign.saml.XMLAssertionGeneratorFactory"%>
<%@ page import="com.verisign.saml.authorities.AssertionGenerator"%>
<%@ page import="com.verisign.saml.assertions.Conditions"%>
<%@ page import="com.verisign.saml.assertions.Statement"%>
<%@ page import="com.verisign.saml.assertions.SubjectLocality"%>
<%@ page import="com.verisign.saml.assertions.AuthNStatement"%>
<%@ page import="com.verisign.saml.assertions.AttributeStatement"%>
<%@ page import="com.verisign.saml.assertions.Attribute"%>
<%@ page import="java.io.InputStream"%>
<%@ page import="java.io.FileInputStream"%>
<%@ page import="java.security.KeyStore"%>
<%@ page import="java.security.PrivateKey"%>
<%@ page import="java.security.cert.Certificate"%>
<%@ page import="java.security.cert.X509Certificate"%>
<%@ page import="com.verisign.xmlsig.RSASigningKey"%>
<%@ page import="com.verisign.xmlsig.RSAVerifyingKey"%>```

ii. **Initialize variables**

The excerpt below instantiates various variables to be used by the SAML assertion. You populate these variables from a properties file in a later step.

**Listing 6. Initialize variables**

```<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="flow.tld" prefix="flow" %>
<%@ include file="../include/JSTLEnvironmentSetup.jspf" %>
<% boolean debug = false;```
String usersId=null;
String authSubject=null;
String userCreditId=null;
String userName=null;
String userType=null;
String userDebitId=null;

String SAML_TARGET = null;
String AUTH_USER_ID = null;
String AUTH_SUBJECT_IP = null;
String SAML_ATTRIBUTE_USER_CREDIT_ID = null;
String SAML_ATTRIBUTE_USER_NAME = null;
String SAML_ATTRIBUTE_USER_TYPE = null;
String SAML_ATTRIBUTE_USER_ID = null;
String SAML_NAMESPACE = null;
String SAML_ASSERTION = null;

String signerAliasResponse = null;
String storeAndKeyPasswordResponse = null;
String keyStoreFileNameResponse = null;

String signerAliasAssertion = null;
String storeAndKeyPasswordAssertion = null;
String keyStoreFileNameAssertion = null;

String samlAssertionEncoded = "nothing done yet...";
String samlAssertionUnencoded = "nothing done yet...";
String samlTarget = "error.jsp";


### iii. Redirecting section

To redirect the FORM to the external application, Javascript is used to redirect the page once the view is loaded in the client browser. This allows for a seamless transition between the view that invoked the URL request to reach the external application.

Listing 7. Redirecting section

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Inter-Site Transfer Service</title>
<script language="javascript" type="text/javascript">
function submitForm()
{
    document.formFromCommerce.action =
        document.formFromCommerce.%<%= SAML_TARGET %>.value;
    document.formFromCommerce.submit();
}
</script>
</head>
<body<%if (debug==false){ %> onload="submitForm()"%> %>
```
iv. **Extract data from the bean**

The excerpt below extracts data from the Inter Site Transfer Service data bean that was populated by the Inter Site Transfer Service controller command.

**Listing 8. Extract data from the bean**

```xml
<c:set var="userId" value="\$\{empireBean.AUTH_USER_ID\}/">
<c:set var="authSubject" value="\$\{empireBean.AUTH_SUBJECT_IP\}/">
<c:set var="externalTaxpayerid" value="\$\{empireBean.SAML_ATTRIBUTE_USER_CREDIT_ID\}/">
<c:set var="username" value="\$\{empireBean.SAML_ATTRIBUTE_USER_NAME\}/">
<c:set var="taxpayerType" value="\$\{empireBean.SAML_ATTRIBUTE_USER_TYPE\}/">
<c:set var="userDebitId" value="\$\{empireBean.SAML_ATTRIBUTE_USER_DEBIT_ID\}/">
```

v. **Extract values from the properties file**

The excerpt below extracts values and constants of the SAML assertion from your properties file to set the parameters to be used in the JSP form.

**Listing 9. Extract values from the properties file**

```xml
<fmt:message var="signerAliasResponse" key="signerAliasResponse" bundle="\$\{tooltechtext\}/">
<fmt:message var="storeAndKeyPasswordResponse" key="storeAndKeyPasswordResponse" bundle="\$\{tooltechtext\}/">
<fmt:message var="keyStoreFileNameResponse" key="keyStoreFileNameResponse" bundle="\$\{tooltechtext\}/">
<fmt:message var="signerAliasAssertion" key="signerAliasAssertion" bundle="\$\{tooltechtext\}/">
<fmt:message var="storeAndKeyPasswordAssertion" key="storeAndKeyPasswordAssertion" bundle="\$\{tooltechtext\}/">
<fmt:message var="keyStoreFileNameAssertion" key="keyStoreFileNameAssertion" bundle="\$\{tooltechtext\}/">
<fmt:message var="SAML_ASSERTION" key="SAML_ASSERTION" bundle="\$\{tooltechtext\}/">
<fmt:message var="SAML_TARGET" key="SAML_TARGET" bundle="\$\{tooltechtext\}/">
<fmt:message var="SAML_ATTRIBUTE_USER_CREDIT_ID" key="SAML_ATTRIBUTE_USER_CREDIT_ID" bundle="\$\{tooltechtext\}/">
<fmt:message var="SAML_ATTRIBUTE_USER_NAME" key="SAML_ATTRIBUTE_USER_NAME" bundle="\$\{tooltechtext\}/">
<fmt:message var="AUTH_ATTRIBUTE_VALUE_PREFIX" key="AUTH_ATTRIBUTE_VALUE_PREFIX" bundle="\$\{tooltechtext\}/">
<fmt:message var="AUTH_ATTRIBUTE_NAME_PREFIX" key="AUTH_ATTRIBUTE_NAME_PREFIX" bundle="\$\{tooltechtext\}/">
<fmt:message var="SAML_NAMESPACE" key="SAML_NAMESPACE" bundle="\$\{tooltechtext\}/">
<fmt:message var="SAML_ATTRIBUTE_USER_TYPE" key="SAML_ATTRIBUTE_USER_TYPE" bundle="\$\{tooltechtext\}/">
<fmt:message var="SAML_ATTRIBUTE_USER_DEBIT_ID" key="SAML_ATTRIBUTE_USER_DEBIT_ID" bundle="\$\{tooltechtext\}/">
```
<fmt:message var="AUTH_SUBJECT_IP" key="AUTH_SUBJECT_IP" bundle="${tooltechtext}"/>
<fmt:message var="AUTH_USER_ID" key="AUTH_USER_ID" bundle="${tooltechtext}"/>

try { SAML_NAMESPACE = (String)pageContext.getAttribute("SAML_NAMESPACE"); } catch (Exception e) {} 
try { SAML_ATTRIBUTE_USER_TYPE = (String)pageContext.getAttribute("SAML_ATTRIBUTE_USER_TYPE"); } catch (Exception e) {} 
try { SAML_ATTRIBUTE_USER_CREDIT_ID = (String)pageContext.getAttribute("SAML_ATTRIBUTE_USER_CREDIT_ID"); } catch (Exception e) {} 
try { AUTH_SUBJECT_IP = (String)pageContext.getAttribute("AUTH_SUBJECT_IP"); } catch (Exception e) {} 
try { AUTH_USER_ID = (String)pageContext.getAttribute("AUTH_USER_ID"); } catch (Exception e) {} 
try { signerAliasResponse = (String)pageContext.getAttribute("signerAliasResponse"); } catch (Exception e) {} 
try { storeAndKeyPasswordResponse = (String)pageContext.getAttribute("storeAndKeyPasswordResponse"); } catch (Exception e) {} 
try { keyStoreFileNameResponse = (String)pageContext.getAttribute("keyStoreFileNameResponse"); } catch (Exception e) {} 
try { signerAliasAssertion = (String)pageContext.getAttribute("signerAliasAssertion"); } catch (Exception e) {} 
try { storeAndKeyPasswordAssertion = (String)pageContext.getAttribute("storeAndKeyPasswordAssertion"); } catch (Exception e) {} 
try { keyStoreFileNameAssertion = (String)pageContext.getAttribute("keyStoreFileNameAssertion"); } catch (Exception e) {} 
try { SAML_ASSERTION = (String)pageContext.getAttribute("SAML_ASSERTION"); } catch (Exception e) {} 
try { SAML_TARGET = (String)pageContext.getAttribute("SAML_TARGET"); } catch (Exception e) {} 
try { SAML_ATTRIBUTE_INTERNAL_TAXPAYER_ID = (String)pageContext.getAttribute("SAML_ATTRIBUTE_INTERNAL_TAXPAYER_ID"); } catch (Exception e) {} 
try { SAML_ATTRIBUTE_TAXPAYER_NAME = (String)pageContext.getAttribute("SAML_ATTRIBUTE_TAXPAYER_NAME"); } catch (Exception e) {} 
try { SAML_ATTRIBUTE_USER_NAME = (String)pageContext.getAttribute("SAML_ATTRIBUTE_USER_NAME"); } catch (Exception e) {} 
try { SAML_ATTRIBUTE_USER_ORG_INFO = (String)pageContext.getAttribute("SAML_ATTRIBUTE_USER_ORG_INFO"); } catch (Exception e) {} 
try { SAML_ATTRIBUTE_USER_EMP_ID = (String)pageContext.getAttribute("SAML_ATTRIBUTE_USER_EMP_ID"); } catch (Exception e) {} 
try { AUTH_ATTRIBUTE_VALUE_PREFIX = (String)pageContext.getAttribute("AUTH_ATTRIBUTE_VALUE_PREFIX"); } catch (Exception e) {} 
try { AUTH_ATTRIBUTE_NAME_PREFIX = (String)pageContext.getAttribute("AUTH_ATTRIBUTE_NAME_PREFIX"); } catch (Exception e) {} 
try { usersId = (String)pageContext.getAttribute("usersId"); } catch (Exception e) {}
vi. **Use WebSphere Commerce to pull the key file**

The excerpt below pulls the key file from the file system using WebSphere Commerce out of the box reference, relative to the WebSphere Commerce installation environment variable. The key file is used to provide credibility for the SAML assertion when sent to the external system. An X509 Certificate is produced. This allows the SAML assertion itself to be the authentication when sending to the external applications.

Listing 10. Use WebSphere Commerce to pull the key file
vii. **Populate the attributes of the SAML assertion**

For each SAML assertion attribute, an attribute object is populated and appended into an array representing all SAML attributes. These SAML attributes are extracted by the external application for their own purposes. The attributes used in this tutorial are confidential information, such as the user's credit, debit, and identifier. These values are confidential for the user and best represented by passing through a SAML assertion.
Listing 11. Populate attributes of the SAML assertion

```java
// retrieve attributes from the request and create attribute array
int attributeCount = 0;
int attributeNumber = 1;
if (request.getParameter(
    AUTH_ATTRIBUTE_NAME_PREFIX + attributeNumber) != null)
{
    do
    {
        attributeCount++;
        attributeNumber++;
    } while (request.getParameter(
        AUTH_ATTRIBUTE_NAME_PREFIX
        + attributeNumber) != null);
}
Attribute attributeArray[] = new Attribute[attributeCount+5];
attributeCount = 0;
attributeNumber = 1;
Attribute attribute;
Object objArray[];
if (request.getParameter(
    AUTH_ATTRIBUTE_NAME_PREFIX + attributeNumber) != null)
{
    do
    {  
        if (request.getParameter(
            AUTH_ATTRIBUTE_NAME_PREFIX + attributeNumber) != null &&
            request.getParameter(
                AUTH_ATTRIBUTE_VALUE_PREFIX + attributeNumber) != null)
        {
            objArray = new Object[1];
            objArray[0] = request.getParameter(
                AUTH_ATTRIBUTE_VALUE_PREFIX + attributeNumber).trim();
            attribute = new Attribute(  
                SAML_NAMESPACE,  
                request.getParameter(  
                    AUTH_ATTRIBUTE_NAME_PREFIX + attributeNumber),  
                objArray);  
            attributeArray[attributeCount] = attribute;
            attributeCount++;
        }
        attributeNumber++;
    } while (request.getParameter(
        AUTH_ATTRIBUTE_NAME_PREFIX + attributeNumber) != null);
}
// set attributes from session
if (userCreditId == null ||
    userCreditId.trim().equals(""))
{
    throw new Exception("userCreditId from session cannot be null");
}
else
{
    objArray = new Object[1];
    objArray[0] = userCreditId.trim();
    attribute = new Attribute(  
        SAML_NAMESPACE,  
        SAML_ATTRIBUTE_USER_CREDIT_ID,  
        objArray);  
    attributeArray[attributeCount] = attribute;
    attributeCount++;
}
```
if (userName == null ||
    userName.trim().equals"
"
) {
    throw new Exception("userName from session cannot be null");
} else {
    objArray = new Object[1];
    objArray[0] = userName.trim();
    attribute = new Attribute(  
        'SAML_NAMESPACE',  
        'SAML_ATTRIBUTE_USER_NAME',  
        objArray);  
    attributeArray[attributeCount] = attribute;  
    attributeCount++;
}

else {
    objArray = new Object[1];
    objArray[0] = userType.trim();
    attribute = new Attribute(  
        'SAML_NAMESPACE',  
        'SAML_ATTRIBUTE_USER_TYPE',  
        objArray);  
    attributeArray[attributeCount] = attribute;  
    attributeCount++;
}

if (userDebitId == null ||
    userDebitId.trim().equals"
"
) {
    throw new Exception("userDebitId from session cannot be null");
} else {
    objArray = new Object[1];
    objArray[0] = userDebitId.trim();
    attribute = new Attribute(  
        'SAML_NAMESPACE',  
        'SAML_ATTRIBUTE_INTERNAL_TAXPAYER_ID',  
        objArray);  
    attributeArray[attributeCount] = attribute;  
    attributeCount++;
}

viii. **Build the SAML assertion FORM field**
From the attribute array and various information, set up the SAML assertion in both encoded and readable format. The readable version of the SAML assertion is used to allow you to debug your requests to the external application and to verify the attributes are correct.

**Listing 12. Build the SAML assertion FORM field**
String CONF_METHOD_BEARER = "urn:oasis:names:tc:SAML:1.0:cm:bearer";
String AUTHN_METHOD_PASSWORD = "urn:oasis:names:tc:SAML:1.0:am:password";
Calendar cal = Calendar.getInstance();
Date now = cal.getTime();
cal.add(Calendar.DATE, 1);
Date tomorrow = cal.getTime();

NameIdentifier samlNameIdentifier = new NameIdentifier(userId.trim());
String[] confMethods = { CONF_METHOD_BEARER };
SubjectConfirmation sconf = new SubjectConfirmation(confMethods);
Subject samlSubject = new Subject(samlNameIdentifier, sconf);

AuthNStatement authNStatement = new AuthNStatement(
    samlSubject,
    null,
    AUTHN_METHOD_PASSWORD,
    now,
    new SubjectLocality(subjectIp,null),
    null);

AttributeStatement attributeStatement = new AttributeStatement(
    samlSubject,
    null,
    attributeArray);

Statement statementArray[] = new Statement[2];
statementArray[0] = authNStatement;
statementArray[1] = attributeStatement;

Conditions conditions = new Conditions(now,tomorrow,null,null);

XMLAssertionGeneratorFactory xaf = new XMLAssertionGeneratorFactory();
xf.setSigningKey(new RSASigningKey(keyAssertion));
xf.setVerifyingKey(new RSAVerifyingKey(certsAssertion));
AssertionGenerator ag = xaf.newAssertionGenerator();
Assertion assertion = ag.createAssertion(
    new Date(),
    IDGenerator.getInstance().getNewAssertionID(),
    SAML_NAMESPACE,
    statementArray,
    conditions,
    null);

Assertion assertionArray[] = new Assertion[1];
assertionArray[0] = assertion;

XMLResponseGeneratorFactory xgf = new XMLResponseGeneratorFactory();
xgf.setSigningKey(new RSASigningKey(keyResponse));
xgf.setVerifyingKey(new RSAVerifyingKey(certsResponse));
ResponseGenerator rg = xgf.newResponseGenerator();
Status samlResponseStatus = new Status(StatusCode.Success);
Response samlResponse = rg.createResponse(samlResponseStatus,assertionArray);
samlAssertionUnencoded = rg.responseToString(samlResponse);
samlAssertionEncoded = Base64Coder.base64Encode(samlAssertionUnencoded);

} catch (Exception e)
{
    samlAssertionEncoded = "Exception : " + e;
ix. **Build the HTML FORM**

The excerpt below shows the FORM being generated based on the SAML assertion. There is a debug mode for testing purposes, which is a flag that you can set in the JSP. Since there is a debug option available, it is best to develop your SAML assertion in the JSP view as done in this tutorial. Then once the SAML assertion is verified as working, move the Java excerpts in this JSP back over to the Inter Site Transfer Service controller command and populate the Inter Site Transfer Service data bean.

**Listing 13. Build the HTML FORM**

```html
<form name="formFromCommerce" method="post" action="<%= samlTarget %>">
  <% if (debug) { %>
    <div>This field contains the SAML assertion (before encoding)</div>
    <div><textarea name="unencoded_SAML_assertion" cols="84" rows="8"><%= samlAssertionUnencoded %></textarea>
    </div>
    <div>This (should be hidden) field contains the encoded SAML assertion (SAML 1.1 requirement)</div>
    <div><input <%if (debug){ %>type="text"<%}else{ %>type="hidden"<%} %>
      name="<%= SAML_ASSERTION %>
      value="<%= samlAssertionEncoded %>
      size="71" />
    </div>
    <% if (debug) { %>
      <div>This (should be hidden) field contains the TARGET (SAML 1.1 requirement)</div>
      <div><input <%if (debug){ %>type="text"<%}else{ %>type="hidden"<%} %>
        value="<%= samlTarget %>
        name="<%= SAML_TARGET %>
        size="38" />
      </div>
      <% if (debug) { %>
        <div>Click on the Submit button below when ready to launch the application with the encoded SAML assertion</div>
        <div><input onclick="submitForm();" type="button" value="Submit" /></>"
      </div>
  <% } %>
</form>
</body>
</html>
```
Inter Site Transfer Services properties files

The excerpt below is what you append into your store’s properties file. That way, there are no hard-coded references in your JSP file. You can reference this information from the Inter Site Transfer Service controller command as well.

Listing 14. Inter Site Transfer Services properties files

```java
SAML_ASSERTION =SAMLResponse
SAML_NAMESPACE =http://externalappdomain.com
SAML_TARGET =/externalapp/external/app
SAML_ATTRIBUTE_USER_CREDIT_ID =UserCreditId
SAML_ATTRIBUTE_USER_DEBIT_ID =UserDebitId
SAML_TARGET =TARGET
AUTH_USER_ID =AUTH_USER_ID
AUTH_SUBJECT_IP =AUTH_SUBJECT_IP
AUTH_ATTRIBUTE_NAME_PREFIX =AUTH_ATTRIBUTE_NAME_
AUTH_ATTRIBUTE_VALUE_PREFIX =AUTH_ATTRIBUTE_VALUE_
SAML_ATTRIBUTE_USER_NAME =UserName
SAML_ATTRIBUTE_USER_TYPE =UserType
signerAliasResponse =demo saml
storeAndKeyPasswordResponse =De5MOex3!
keyStoreFileNameResponse =DEMO_SAML_Keystore.jks
signerAliasAssertion =demo saml
storeAndKeyPasswordAssertion =De5MOex3!
keyStoreFileNameAssertion =DEMO_SAML_Keystore.jks
SAML_ATTRIBUTE_USER_ID =UserId
```

3. **Request to assertion consumer service**

The Inter Site Transfer Service View has a redirect contained within the JSP file. This redirect is written in Javascript that allows the client side browser to handle the request automatically.

4. **Respond to the principal's original request**

This step is handled by the external application. The external application accepts this request and handles it accordingly. This information will be provided in a future article. The key to this tutorial is how to setup the SAML assertion into WebSphere Commerce.

From the steps provided above, you can reuse and place these code excerpts directly into your WebSphere Commerce flow. For more information, see the WebSphere Information Center topic, *Customizing existing controller commands*.
Section 6. Conclusion

The tutorial showed how to implement identity federation solutions along with using your WebSphere Commerce application as the principal. Now that you have a better understanding of using WebSphere Commerce best practices when integrating SAML, your external applications can start sharing your aggregated services whether they are in centralized, decentralized, or federated deployments.
Resources

- XML Security: Ensure portable trust with SAML
- Debunking SAML myths and misunderstandings
- Understanding SOA Security Design and Implementation
- Tutorial: Leveraging the WebSphere Commerce store archive

About the author

Michael D.J. Shields

Michael Shields is a Solution Architect on the IBM Software Services team at IBM Toronto, Canada. He has several years of experience with WebSphere Commerce, consisting of product development and consulting work. He works with clients to perform requirements gathering, fit-gap analysis, design, implementation, and deployment of WebSphere Commerce solutions. Michael has published over 12 tutorials for the WebSphere Commerce Information Center. Along with over 12 technical certifications, he also holds designation in project management (PMP).