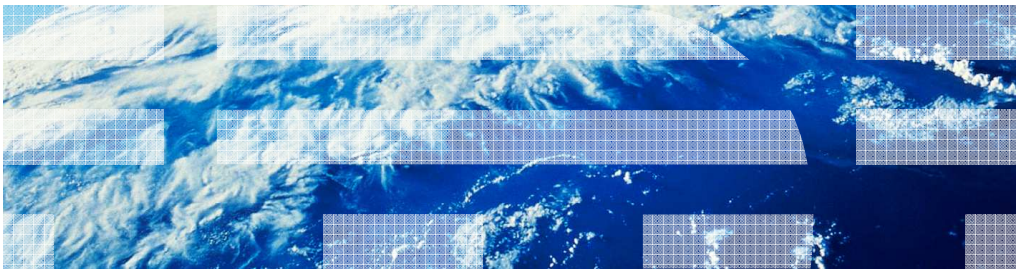
IBM WebSphere Application Server V8

Web Services Security SAML usage scenarios

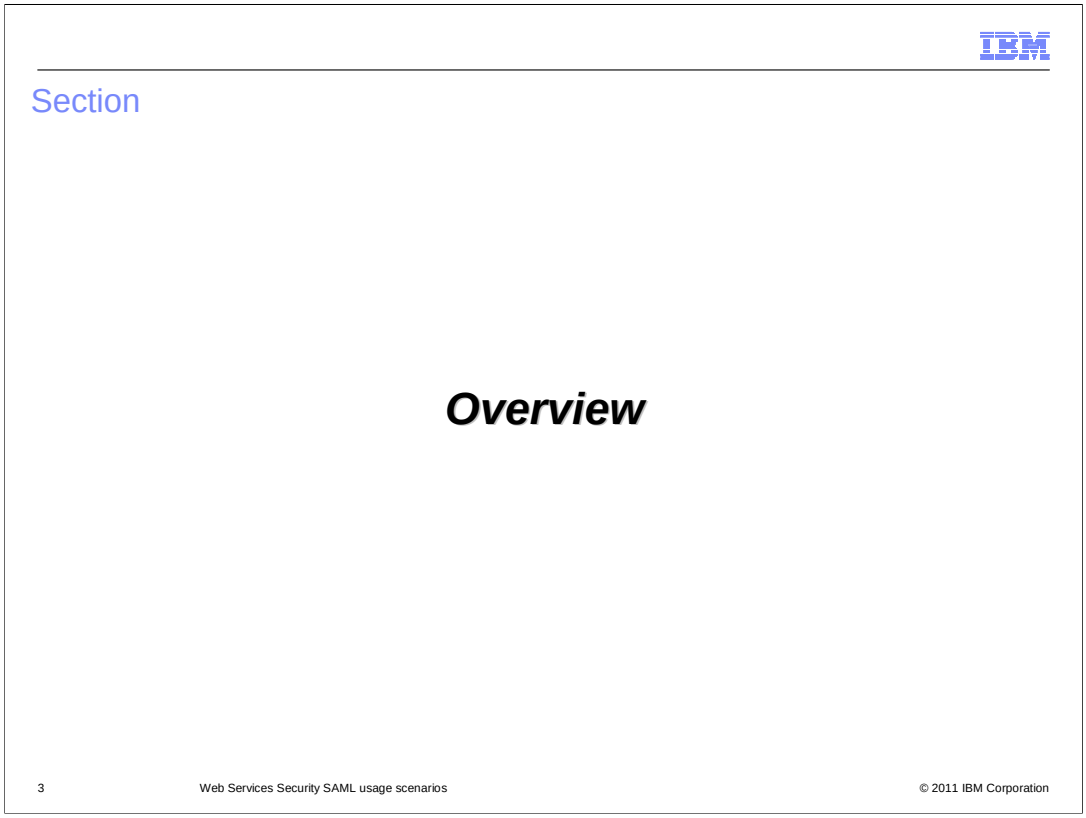


WebSphere software

This presentation describes support for web services Security SAML in IBM WebSphere Application Server V8.

Table of contents

This module covers the web Services Security SAML (Security Assertion Markup Language) feature and describes several typical usage scenarios.



In this section you will see an overview of the SAML (Security Assertion Markup Language) token support for web Services Security.

SAML Features

- SAML, or Security Assertion Markup Language, is a set of OASIS standards that define the syntax and processing semantics of assertions made about a subject by a system entity
- The SAML Feature in V8 concerns the use of SAML Assertions as security tokens in web services SOAP security to exchange client identity and other security information across security domains

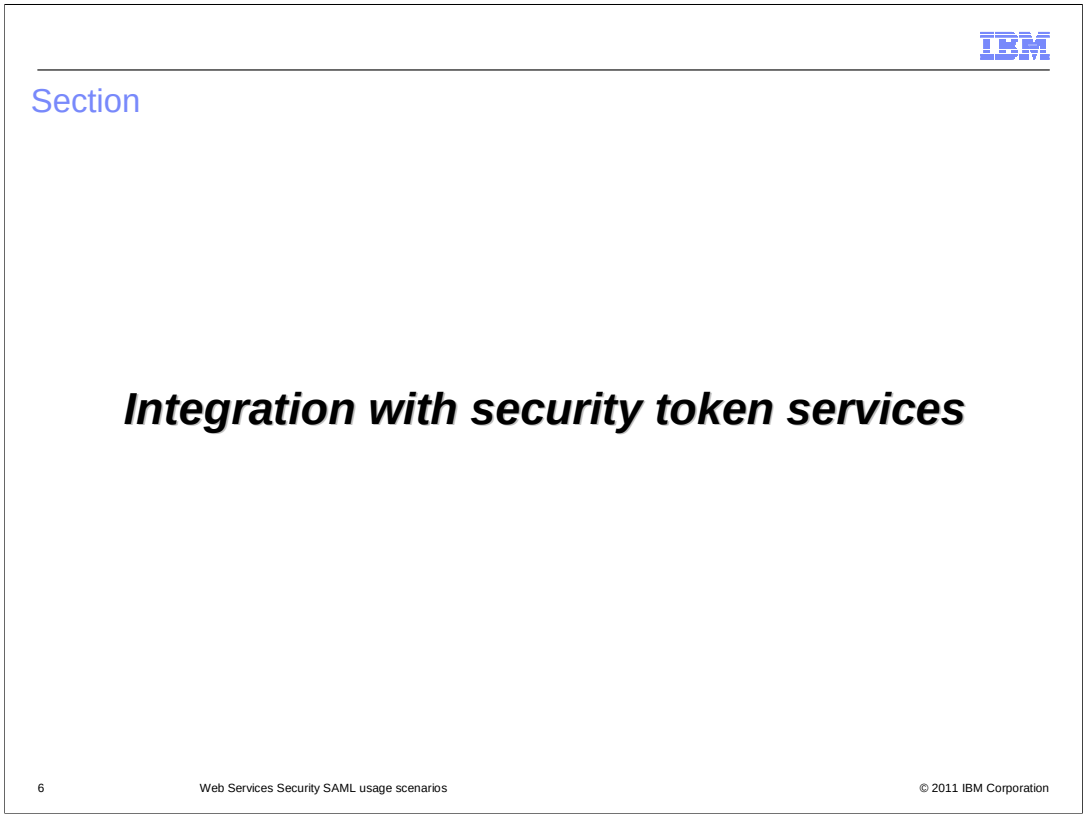
SAML (Security Assertion Markup Language) is a set of OASIS (Organization for the Advancement of Structured Information Standards) standards describing syntax and processing semantics for assertions made about a subject. A subject can be a user or some other entity. The SAML syntax is expressed in the form of XML (Extensible Markup Language).

The SAML feature in WebSphere Application Server V8 provides the ability for SAML Assertions to be propagated as security tokens in SOAP web services messages. This allows the client identity and other security attributes of the client to be transferred, within a security domain or across security domains.

Section

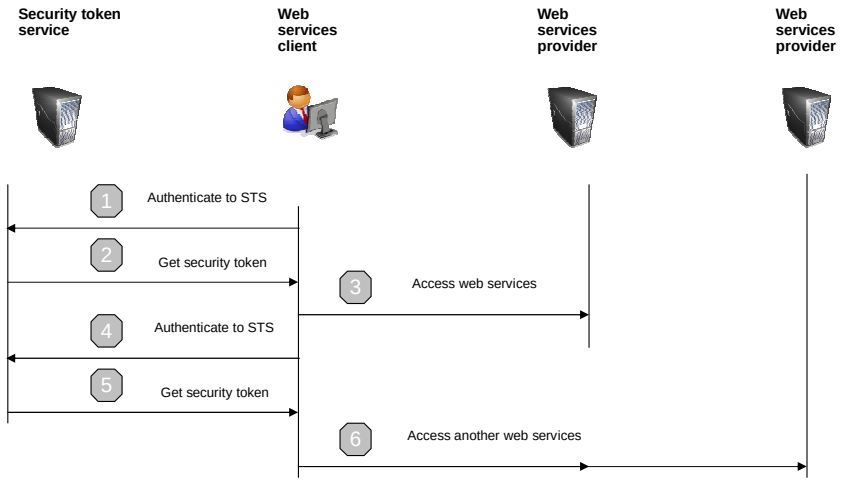
Usage scenarios

In this section you will see several usage scenarios for the SAML feature.



The first scenario shows integration with an STS (Security Token Service).

Security token service (STS) usage scenario



7 Web Services Security SAML usage scenarios

A Security Token Service (STS) is a service that can handle requests to issue or validate security tokens. It uses the standardized web Services Trust (WS-Trust) protocol to handle these requests. The IBM Tivoli Federated Identity Manager is one such product that provides an STS.

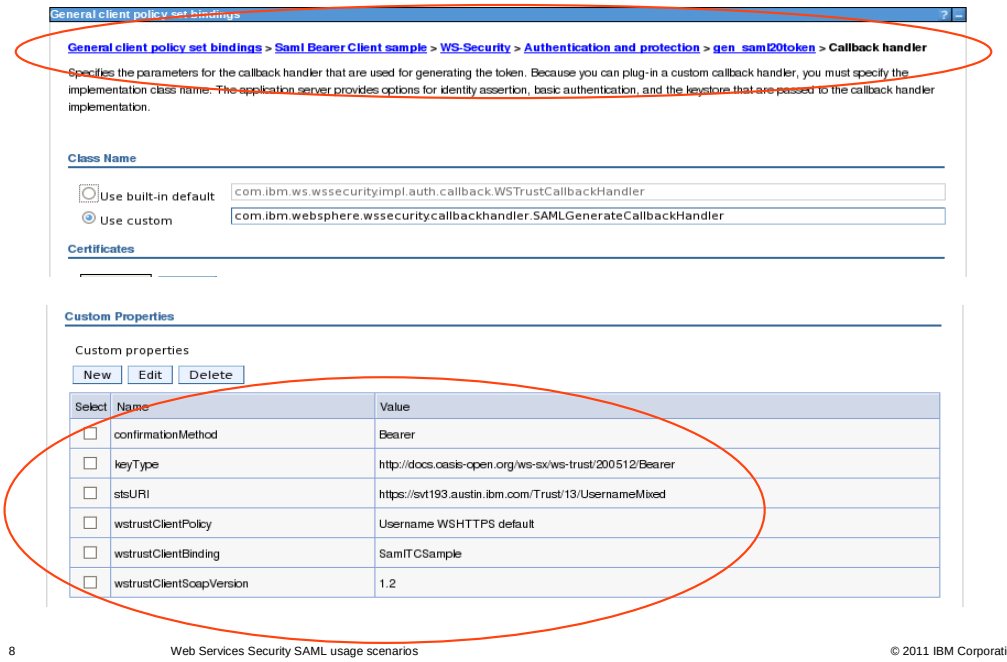
In this scenario, the client needs to access a web services provider, and this provider requires that a SAML token be sent in the SOAP message. In order to obtain the SAML token, the web services client will authenticate to the STS and request the token from the STS. This is shown in the flows labeled “1” and “2” in the diagram.

Once the SAML token has been obtained, the web services client can then send the SOAP message, including the SAML token, to the target web services provider. This is the flow labeled “3” in the diagram.

If the client then needs to access another web services provider, this flow can be repeated, as shown in the flows labeled “4”, “5” and “6”.

Using an STS has the advantage of a centralized entity to control the issuance of tokens. The management of the service, its policies for authentication requirements and access to a user repository are all centralized. This simplifies the management of your infrastructure.

STS client configuration



The behavior of web services security on the client is configured using “policy sets” and bindings. The policy will specify that a SAML token is to be sent. And the bindings have the details of how the SAML token is to be obtained.

This screen capture shows the console panel where the bindings for the generation of the SAML token are configured. As you can see, there are properties to configure the URI of the STS, the confirmation method of the SAML token and the SOAP version to use in the WS-Trust request to the STS. There are also properties to specify the “policy set” and bindings to use to protect the WS-Trust request to the STS.

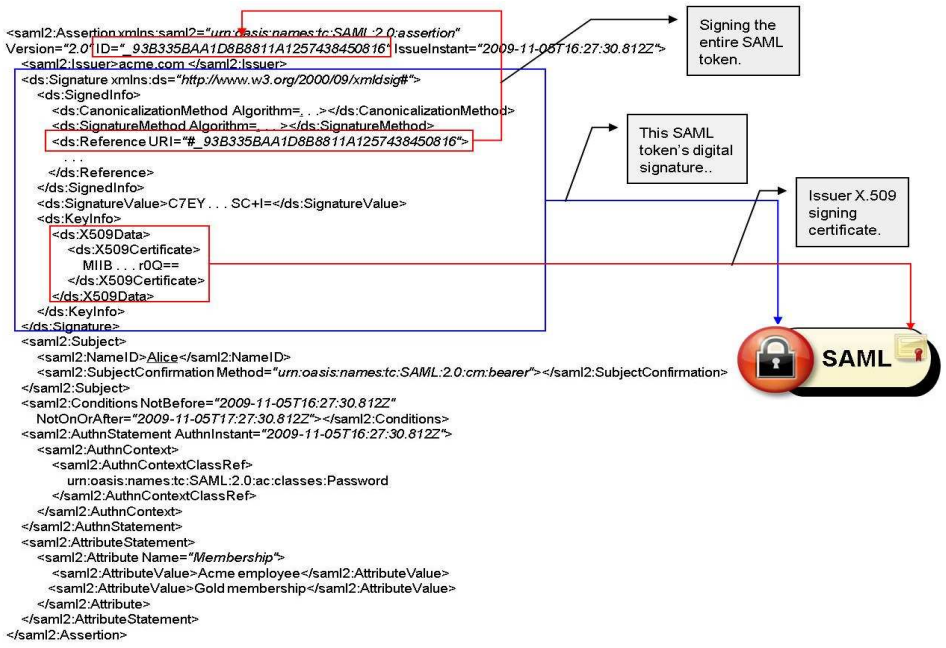
The “Bearer” confirmation method shown in the properties here is one of the three SAML confirmation methods supported.

Section

SAML token propagation in SOAP messages

The next scenario covers propagation of a SAML token in SOAP messages.

SAML assertions example



An advantage of using SAML tokens is that the format is standardized and allows a rich set of attributes to be associated with the subject. Using a standardized format promotes interoperability.

This slide shows an example SAML assertion. As you can see, it is in XML format and can be sent as a security token in the security header of a SOAP message.

In this example, the SAML token is signed. It is typical that the SAML token be signed by the issuing STS. Signing the token ensures the token is not altered after it is signed and allows verification that it was issued by a trusted STS. You can see the signature in the blue box on the slide.

The bottom half of the slide, below the blue box, shows the rest of the SAML token.

The “Subject” element specifies that the name of the subject represented by this token is Alice and that the confirmation method is “Bearer”.

The “Conditions” element specifies some expiration constraints for the token.

The attribute statement shows the group membership of the subject.

Also note the SAML icon on the right side of the slide. This icon is used to represent the SAML token in several of the subsequent slides in this presentation. The lock on the left of the icon indicates that the token is signed. And the certificate on the right of the icon represents the signing certificate of the token issuer.

SAML assertions as security token in SOAP messages

- Supports OASIS web Services Security SAML Token Profile 1.1 Usage Scenarios
  - Supports Bearer, Sender-vouches, and Holder-of-key Subject Confirmation Methods

Subject confirmation method	Use for	Requirement
Bearer	Identity and attributes propagation	None
Sender-vouches	Identity and attributes propagation	Sender must ensure integrity of message and SAML security token
Holder-of-key	Identity and attributes propagation, and message integrity protection	Sender must demonstrate knowledge of key

The OASIS “Web Services Security SAML Token Profile 1.1” defines how SAML tokens can be sent in the security header of a SOAP message and how they can be used to protect parts of the SOAP message. This profile supports the use of three confirmation methods for SAML tokens – Bearer, Sender-Vouches and Holder-of-Key.

All three of the confirmation methods allow the identity and attributes of the user ID to be propagated.

Additionally, Holder-of-Key allows protection of the SOAP message using key material contained in the token.

Each confirmation method has different requirements for how the token is protected and vouched for.

The Bearer token has no special requirements.

The Sender-Vouches token requires that the sender protect the integrity of the token and bind it to the SOAP message. This can be done by signing the token and key parts of the SOAP message, like the message “Body”.

The Holder-of-Key token requires that the sender demonstrate knowledge of the key referenced by the token. This is typically done by signing the message using the key referenced by the token.

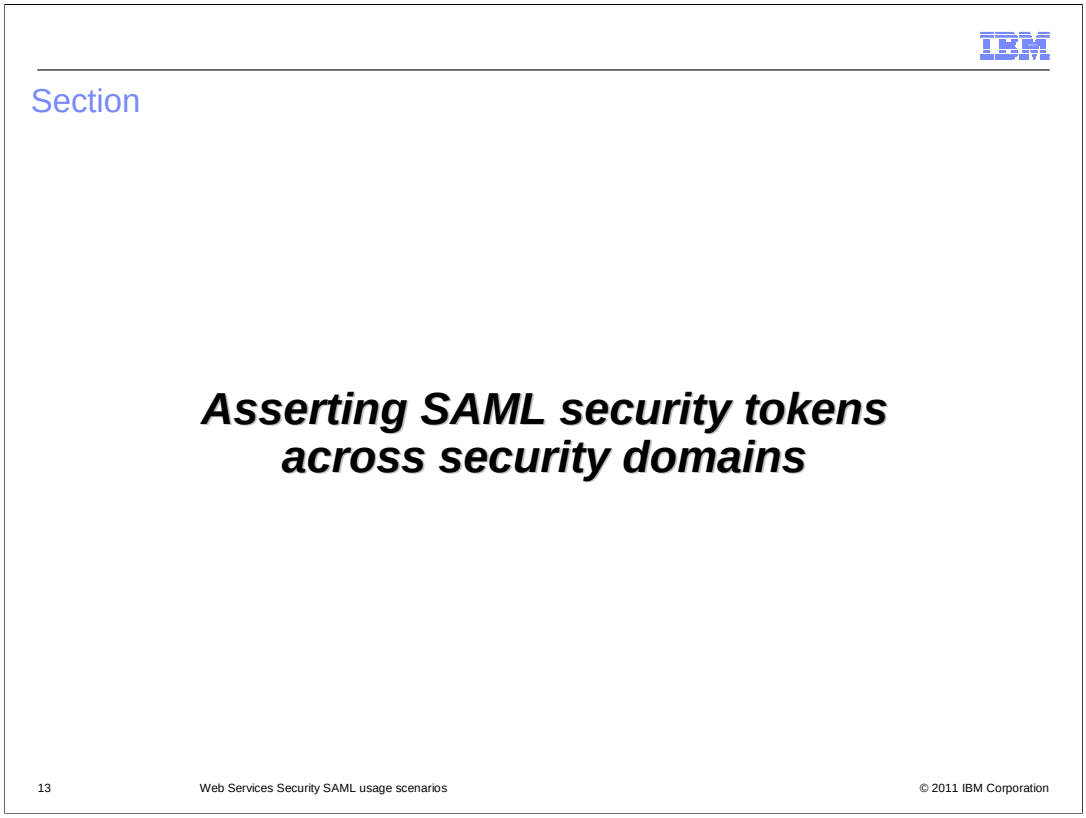
Default SAML policy sets

☐	SAML11 HoK Symmetric WSSecurity default	Not editable	Policies: WSSecurity, WSAddressing • Message integrity: Digitally sign body, timestamp and addressing headers • Message confidentiality: Encrypt body and signature • Message authentication: Using SAML 1.1 token contains holder-of-key confirmation method and a symmetric key. • Follows the OASIS SAML Token Profile and WS-Security specifications.
☑	SAML20 Bearer WSHTTPS default	Not editable	Policies: HTTPTransport, SSLTransport, WSAddressing, WSSecurity • Transport security: Using SSL for HTTP • Message authentication: Using SAML 2.0 Token with bearer confirmation method.
☐	SAML20 Bearer WSSecurity default	Not editable	Policies: WSSecurity, WSAddressing • Message integrity: Digitally sign body, timestamp and addressing headers • Message confidentiality: Encrypt body and signature • Message authentication: Using SAML 2.0 Token with bearer confirmation method. • Follows the OASIS SAML Token Profile and WS-Security specifications.

Eight predefined “policy sets” for use with SAML tokens are shipped with WebSphere Application Server V8. A selection of three of them are shown here. The shipped “policy sets” include both Bearer and Holder-of-Key confirmation methods in addition to SAML 1.1 and SAML 2.0 token versions.

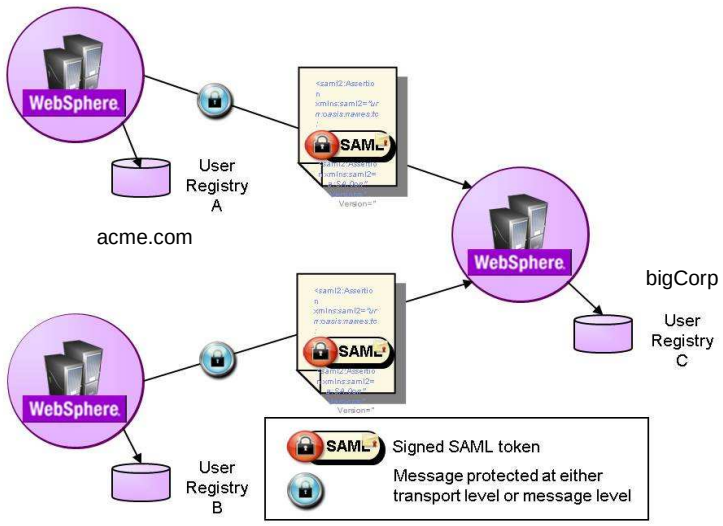
There are no predefined Sender-Vouches “policy sets” shipped. But there are articles in the Information Center describing how to copy and modify one of the existing predefined “policy sets” to use the Sender-Vouches confirmation method.

The predefined “policy sets” shipped for SAML should cover many common scenarios, either as-is or with minor modifications.



The next scenario covers asserting SAML tokens across different security domains.

Asserting SAML security tokens across security domains

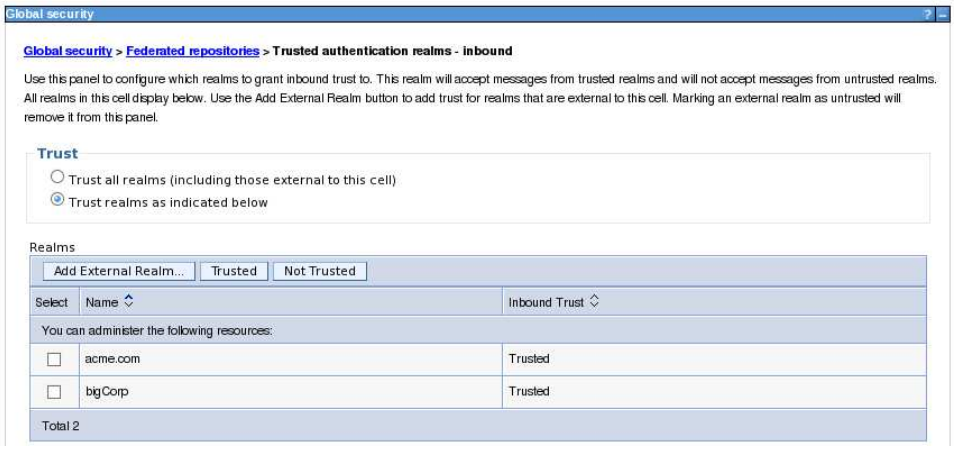


SAML tokens allow a user's identity to be propagated across security domains.

This allows you to provide web services that can be accessed by your business partners without having to share a common user registry. And similarly you can access the web services of your business partners.

In this diagram, “Big Corp” is providing a web service to multiple business partners, for example acme.com. “Big Corp” and acme.com each have their own user registry. This feature provides the ability to establish a trust relationship between “Big Corp” and acme.com. Acme.com users can then access the “Big Corp” service without having to share or federate their user registries.

Trusted authentication realm



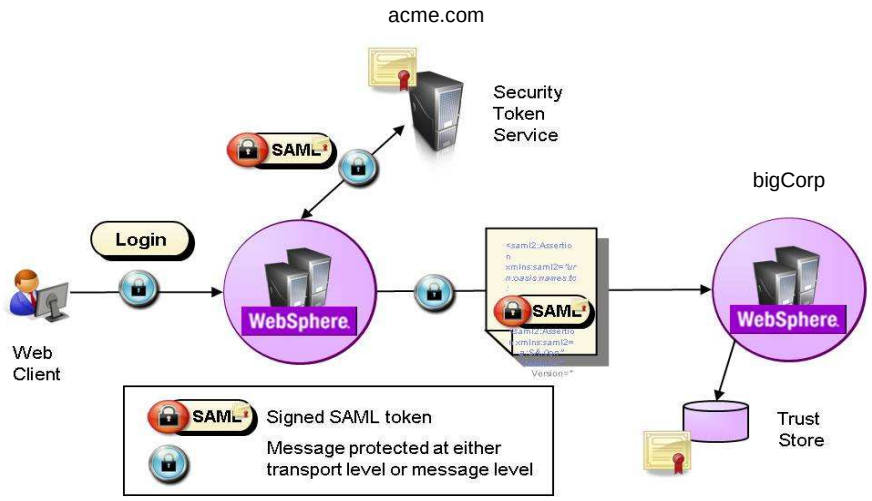
15 Web Services Security SAML usage scenarios

To establish this trust relationship, you use the “Trusted Authentication Realm” feature of the WebSphere Application Server.

This slide shows the panel in the administrative console used to configure the trusted realms.

In the configuration shown, “Big Corp” will allow access to authorized users from the acme.com realm as long as the SAML token is issued by acme.com. And “Big Corp” will also allow access to authorized users from the “Big Corp” realm – its own local realm.

Asserting SAML security tokens across security domains



16 Web Services Security SAML usage scenarios

Putting together the previous ideas, this slide shows the flow of this scenario.

The web services client at acme.com will use the login credentials of the user to request a SAML token from the acme.com STS. The SAML token is signed by the issuer , namely the acme.com STS. The web services client can then put the SAML token in the web services message and send it to the web service hosted at “Big Corp”.

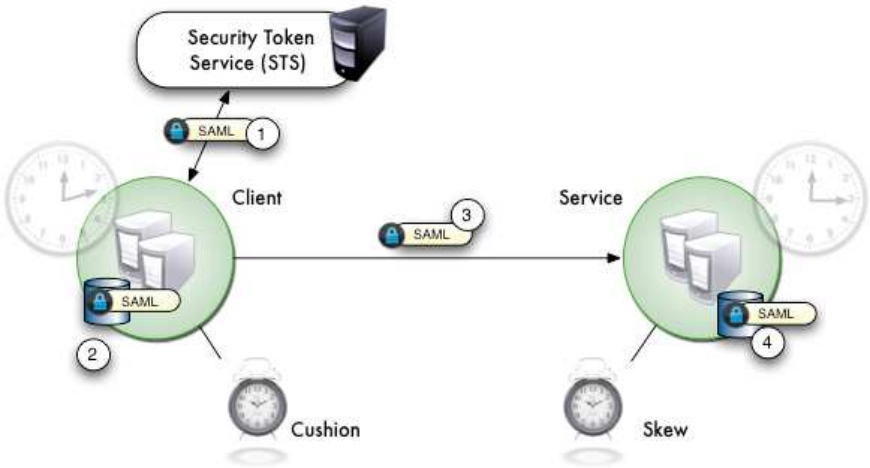
“Big Corp” will receive the message, verify the signature of the SAML token and confirm that the signer matches the acme.com certificate contained in its trust store. And if acme.com is one of the realms configured as a trusted authentication realm, “Big Corp” can allow access by authorized users from that realm.

Section

SAML token lifetime management

The next scenario covers SAML tokens lifetime management.

SAML token lifetime



18 Web Services Security SAML usage scenarios

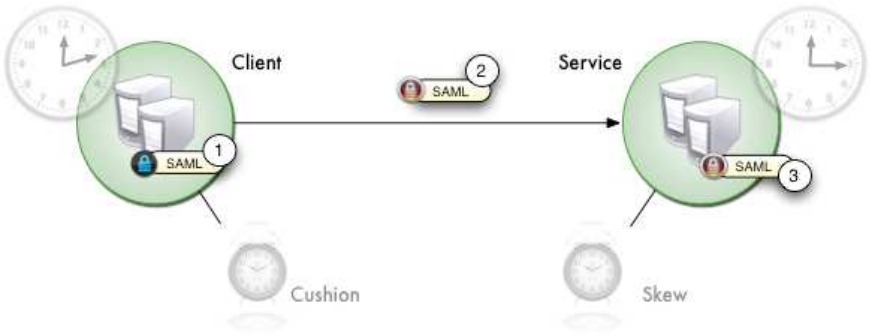
As seen in a previous slide, a SAML assertion contains token lifetime information in the form of “Not Before” and “Not On Or After” attributes. These attributes define the time period which the SAML token can be successfully validated. This time is typically kept relatively short to reduce the chances of a replay attack where the SAML token is captured and used again at a later time.

This feature provides three mechanisms for dealing with token lifetime management issues.

The first is a client side cache for the SAML tokens. If the client is making repeated calls to the same service and needs to send a SAML token each time, it can request a new SAML token each time from the STS. The client side cache allows the client to skip the call to the STS if it has a cached token that still has a reasonable lifetime. This reduces the overhead of making some of the calls to the STS, improving performance.

The second involves accounting for clock skew between client and the service. There is a configurable clock skew parameter to specify how much clock skew is tolerable in your environment. For example, assume the clock skew is set to one minute. The token arrives at the service and according to the clock at the service, the token expired 30 seconds ago. Since 30 seconds is within the allowed clock skew period of one minute, the service will not reject the token.

SAML token propagation



The third lifetime management mechanism involves token propagation to a downstream service.

Consider a situation where an application requires a long time to execute. For example, the application might perform some processing steps that take a long time and then need to make a downstream call to another service. If the original SAML token obtained before the start of the processing steps is sent in the downstream call, it may have reached the “Not On Or After” time and be rejected by the downstream service.

To solve this problem, the WebSphere Application Server runtime can generate a new SAML token based on the original SAML token. It will have all the attributes of the original, but with a fresh lifetime. This new SAML token is sent to the downstream service.

Section

SAML token and WS-Trust client API

The next scenario covers the use of the SAML token and WS-Trust client Application Programming Interfaces (APIs).

Create SAML tokens sample code

```
SAMLToken samlToken = null;
  try{
   SAMLTokenFactory samlTokenFactory = SAMLTokenFactory.getInstance(SAMLTokenFactory.WssSamlV11Token11);

   RequesterConfig reqData = samlTokenFactory.newBearerTokenGenerateConfig();
    reqData.setAuthenticationMethod("password");
    reqData.setAssertionSignatureRequired(signing);
    reqData.getRSTTProperties().put(RSTT.APPLIESTO_ADDRESS, "http://acme.com:9080/saml/demo");

    CredentialConfig cred = samlTokenFactory.newCredentialConfig();
   String companyName = "IBM";
    String userName = "cyc";
   String alias = "oliver stone"
    cred.setRequesterNameID(userName);
    ArrayList<SAMLAttribute> userAttrs = new ArrayList<SAMLAttribute>();
   SAMLAttribute sAttribute1 = new SAMLAttribute("CompanyInfo", ,new String [] {companyName},  null, "WebSphre Namespace",
                                                 null,  "Company Name : " + companyName);
    SAMLAttribute sAttribute2 = new SAMLAttribute("UserAlias",  new String [] {alias}, null, "WebSphre Namespace",
                                                 null,  "User Alias : " + alias);
   userAttrs.add(sAttribute1);
     userAttrs.add(sAttribute2);
   cred.setSAMLAttributes(userAttrs);

     ProviderConfig samlProviderCfg = samlTokenFactory.newDefaultProviderConfig(null);
     samlToken = samlTokenFactory.newSAMLToken(cred, reqData, samlProviderCfg);
  } catch (Exception e) {
     // Error handling
    . . .
  }
```

RequesterConfig object specifies SAML Token parameters

CredentialConfig object specifies Subject related parameters

ProviderConfig object specifies Issuer related parameters. The newDefaultProviderConfig() method reads initial values from a properties file

The SAML support described so far allows the propagation of SAML tokens in SOAP web services messages. And this is achieved using the standard “policy set” and bindings configuration model. No custom code is required.

But if you have some special requirements or need to be able to generate SAML tokens from within your application code, a set of APIs is provided for that.

This slide shows some sample code using the APIs to create a SAML token.

The general flow is as follows.

First, get an instance of the SAML token factory class.

Next, set up the “RequesterConfig” object to specify parameters of the SAML token. These include parameters like the confirmation method, authentication method and whether the token should be signed.

Next, set up the “CredentialConfig” object to specify parameters related to the subject, or user. These include parameters like the user name and company name.

Next, set up the “ProviderConfig” object to specify parameters related to the issuer. The default values for this object are read from a properties file.

Finally, the token is generated using the “new SAML Token” method of the SAML token factory class. The three objects you previously set up are passed to this method as parameters.

Parse SAML assertions and create JAAS subject

```
try {
   SAMLTokenFactory samlFactory =
   SAMLTokenFactory.getInstance(SAMLTokenFactory.WssSamlV11Token11);

   ConsumerConfig samlConsumerCfg =  samlFactory.newConsumerConfig();
   samlConsumerCfg.setAssertionSignatureRequired(false);

   //create SAMLToken from inputStream
   SAMLToken samlToken = samlFactory.newSAMLToken(samlConsumerCfg, in);

   //create Subject that contains SAML NameIdentifier as principal, and SAMLToken in private credentials
   Subject subject = samlFactory.newSubject(samlToken);

   Context context = getContext();
   Subject oldSubject = context.getRunAsSubject();
   context.setRunAsSubject(subject);
     context.setContext();
     // do something with the new security context
    . . .
     //restore original security context
     context.setRunAsSubject(oldSubject);
     context.setContext();

  } catch (Exception e){
    // Error handling
  }
```


This example shows how to read, or parse, a SAML token and then set the token into the JAAS (Java Authentication and Authorization Service) subject.

The general flow is as follows.

Create a SAML token factory instance.

Create a “ConsumerConfig” object.

Create the SAML token object by reading the XML representation of the token from an input stream.

Create the JAAS subject.

Set the subject onto the current security context.

Request issuing SAML tokens sample code

```
try {
    ProviderConfig providerConfig = WSSTrustClient.newProviderConfig(Namespace.WST13,
                        https://acme.com/Trust/13/UsernameMixed );

    providerConfig.setPolicySetName("Username WSHTTPS default");
    providerConfig.setBindingName("SamlTCSample");
    providerConfig.setBindingScope("domain");

    RequesterConfig requesterConfig = WSSTrustClient.newRequesterConfig(Namespace.WST13);
    requesterConfig.put(RequesterConfiguration.RSTT.APPLIESTO_ADDRESS,
                        "https://acme.com:9443/WSSampleSei/EchoService12");
    requesterConfig.put(RequesterConfiguration.RSTT.TOKENTYPE, TokenType.SAML11);
    requesterConfig.put(RequesterConfiguration.RSTT.KEYTYPE, WST13.KEYTYPE_BEARER);
    requesterConfig.setSOAPNamespace(Namespace.SOAP12);

    WSSTrustClient client = WSSTrustClient.getInstance(providerConfig);
    List<SecurityToken> securityTokens = client.issue(providerConfig, requesterConfig);

} catch (SoapSecurityException ex) {
  System.out.println("Caught exception: " + ex.getMessage());
  ex.printStackTrace();
}
```


This example shows how to use the WS-Trust client APIs to request a SAML token from an STS.

The general flow is as follows.

Create a “ProviderConfig” object to specify the STS URI and the name of the “policy set” and bindings used to protect the STS request.

Create a “RequesterConfig” object to specify the SAML token version, confirmation method, SOAP version, and so on.

And finally, issue the request to the STS and retrieve the returned SAML token.

Note that the WS-Trust client APIs can be used to request other types of tokens besides SAML from the STS.

This section provides a summary of what you have learned in this presentation.

Summary

- Overview
- Usage scenarios
  - Integration with security token service
  - SAML token propagation
  - Asserting SAML tokens
  - Lifetime management
  - SAML token and WS-Trust client API

In this presentation you have seen an overview of the web Services Security SAML feature and some typical SAML usage scenarios.

The usage scenarios covered integration with an STS, SAML token propagation, asserting SAML tokens and token lifetime management. You have also seen examples of how to use the SAML token and WS-Trust client APIs.

References

- Feature Focus Week: OASIS web Services Security SAML Usage Scenarios

http://www.ibm.com/developerworks/forums/thread.jspa?threadID=346381&tstart=0

- IBM WebSphere Application Server V8 Alpha Forum

http://www.ibm.com/developerworks/forums/forum.jspa?forumID=2180&cat=9

- Introduction to SAML and support in WebSphere Application Server V7 FixPack 7

https://www.ibm.com/developerworks/wikis/download/attachments/116424904/Introduction+to+SAML+and+support+in+7.0.0.7.pdf

- YouTube Demo

http://www.youtube.com/watch?v=bgYxDsMwbjA

- SAML Assertion Across WebSphere Application Server Security Domains

http://www.ibm.com/developerworks/websphere/techjournal/1004_chao/1004_chao.html

- Secure web services Using SAML

http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/twbs_secureassertions.html

- SAML API

http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/rwbs_libraryapisaml.html

- WS-Trust Client API

http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/rwbs_samltrustclientapi.html

This slide contains links to useful information.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASV8_SAML.ppt

This module is also available in PDF format at: ../WASV8_SAML.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Java, and all Java-based trademarks and logos are trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.