The strategic direction for IBM is to provide product and solution based on standard specifications. This is no different in Web Services Security. WebSphere Application Server Version 5.0.2 supports the Web services security specification, “Web Services Security: SOAP Message Security” , proposed by IBM, Microsoft and Verisign in April 2002. The implementation is based on the IBM Web services engine.

Agenda

- WS-Security Overview
- WS-Security Architecture and Deployment Model
- WS-Security Authentication, Integrity and Confidentiality
- Authentication Flow – J2EE and WS-Security
- WS-Security Binding File Contents
- Tooling Support
- Scenarios
- Summary

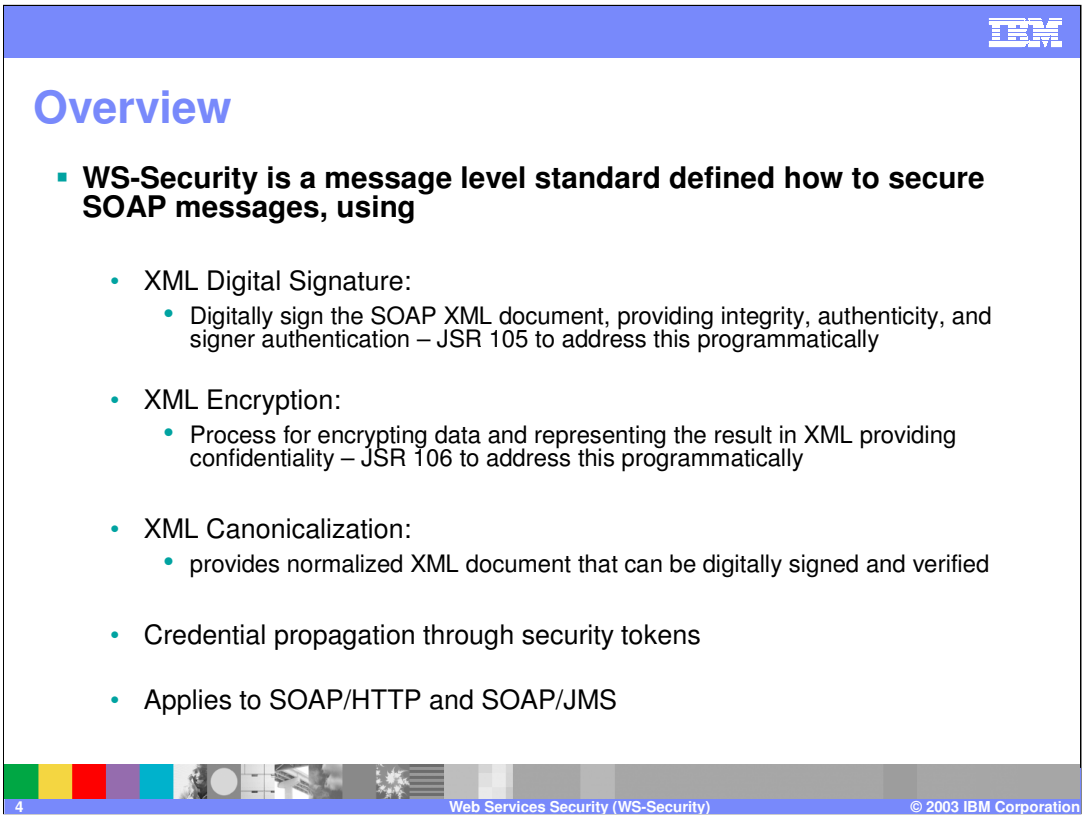
We will discuss the WS-Security overview, authentication, integrity and confidentiality.

We will discuss the authentication flow – since with WS-Security, the security information is flowing through the SOAP message, we have two authentication flows – the J2EE authentication flow and WS-Security authentication flow.

We will also talk about the WS-security deployment descriptors which is comprised of web services bindings and extension files which are used exclusively for specifying the ws-security constraints on the client and server side. We will also talk about the tooling support and take a high level look at some scenarios.

Section

WS-Security Overview

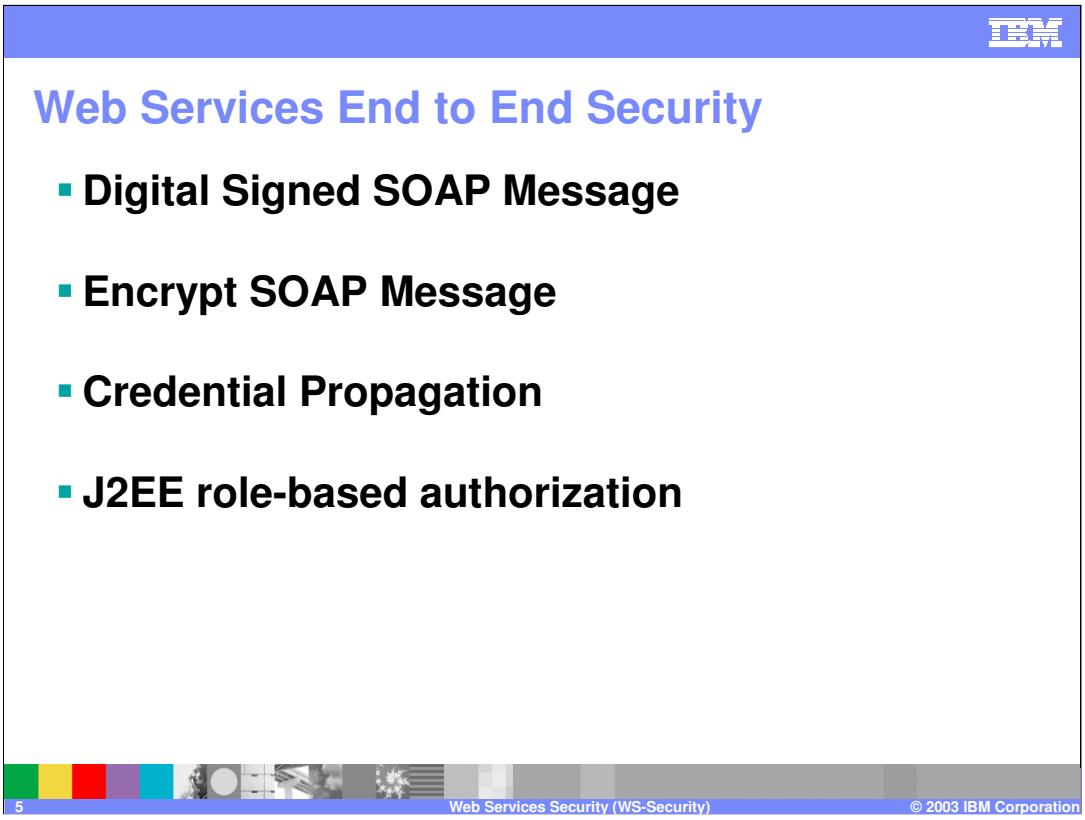


Web services security for WebSphere Application Server, Version 5.0.2 and above is based on standards included in the Web services security (WS-Security) specification. Web services security is a message-level standard, based on securing Simple Object Access Protocol (SOAP) messages through XML digital signature, confidentiality through XML encryption and credential propagation through security tokens.

WS-Security is a message level standard which means that security information is part of the SOAP message from the client. The client, based on the constraints in the web services binding and extension files will insert WS-Security information in the SOAP message. And the server, based on the constraints in the server-side web services binding and extension files will check for the security constraints in the incoming SOAP message's header. So, the security information is part of the message that is passed as part of the SOAP message that goes from the client to the server.

Web services security defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. The security constraints can specify XML Digital Signature (for Message Integrity), XML Encryption (for Message Confidentiality), the credential propagation. WS-Security actually applies to both SOAP/HTTP as well as SOAP/JMS. The structure of the SOAP message's security constraints contents is same in both cases.

Today, there is no way to specify the security constraints programmatically. The WS-security constraints have to be specified within the web services bindings and extensions. JSR 105 and JSR 106 allows a programmer to specify the digital signature and Encryption programmatically.



Web Services end-to-end security involves digital signed SOAP message, encrypting the SOAP message, credential propagation and J2EE role-based authorization.

To realize the benefits of Web services security, it is recommended that an implementation of the specification is integrated with underlying security mechanisms. This implementation is fully integrated with the WebSphere Application Server, Version 5.0.2 security infrastructure. Authorization, for example, is based on the J2EE security model. When a user ID and password are embedded in a request message, authentication is performed with the user ID and password. If successful, a user identity is established in the context of execution and further resource access is authorized based on that identity. The authentication process is similar to the process for HTTP basic authentication. Once the user ID and password are authenticated, authorization is performed by a J2EE container.

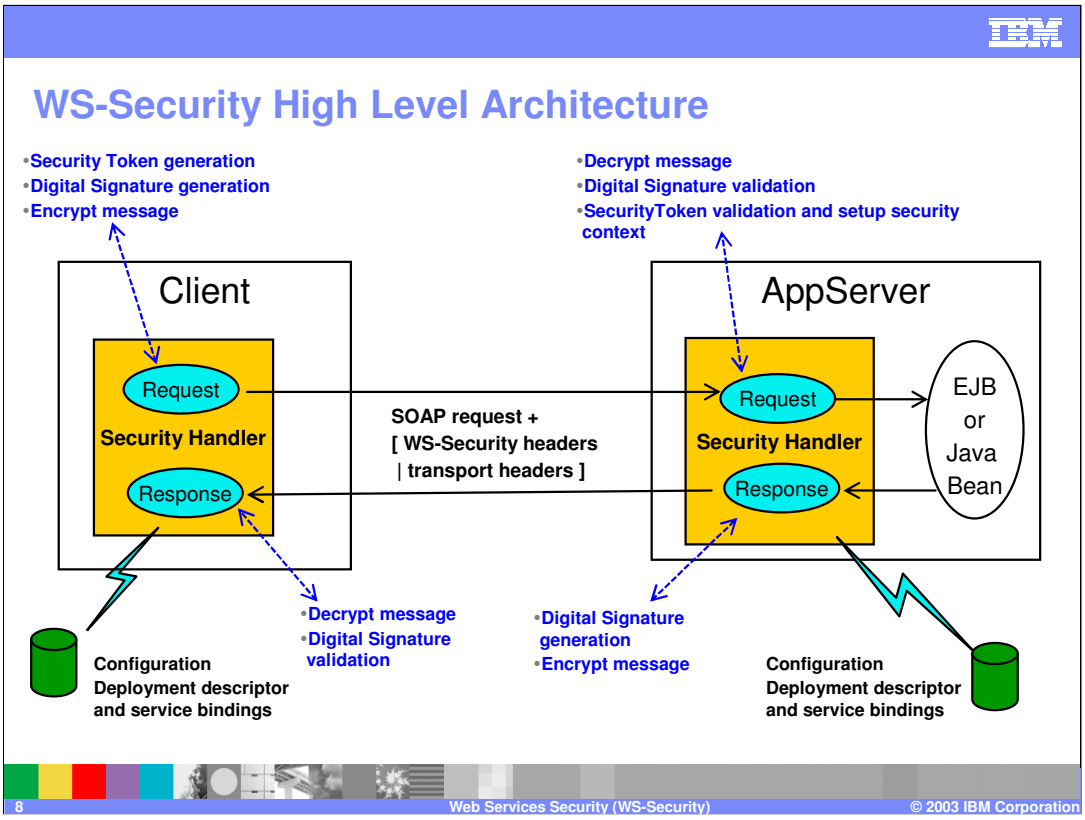
Section

WS-Security Architecture
and
Deployment Model

WS-Security Implementation in WebSphere

- **WS-Security is implemented as message level system handler and is registered to the Web Service runtime by the Application Server**
  - Henceforth, the handlers will be referred to as the Security Handlers
- At the Client: **Security handler is invoked to generate the required security headers in the SOAP message before the message is sent out to the wire**
- At the Provider (Server): **Security handler is called to enforce the declared security constraint in the deployment descriptor prior to dispatching the request to the Web Service Provider (EJB or Java Beans) implementation**

WS-Security Handler are similar to security interceptors for CSIv2 or SAS or Trust Association Interceptors (TAIs).



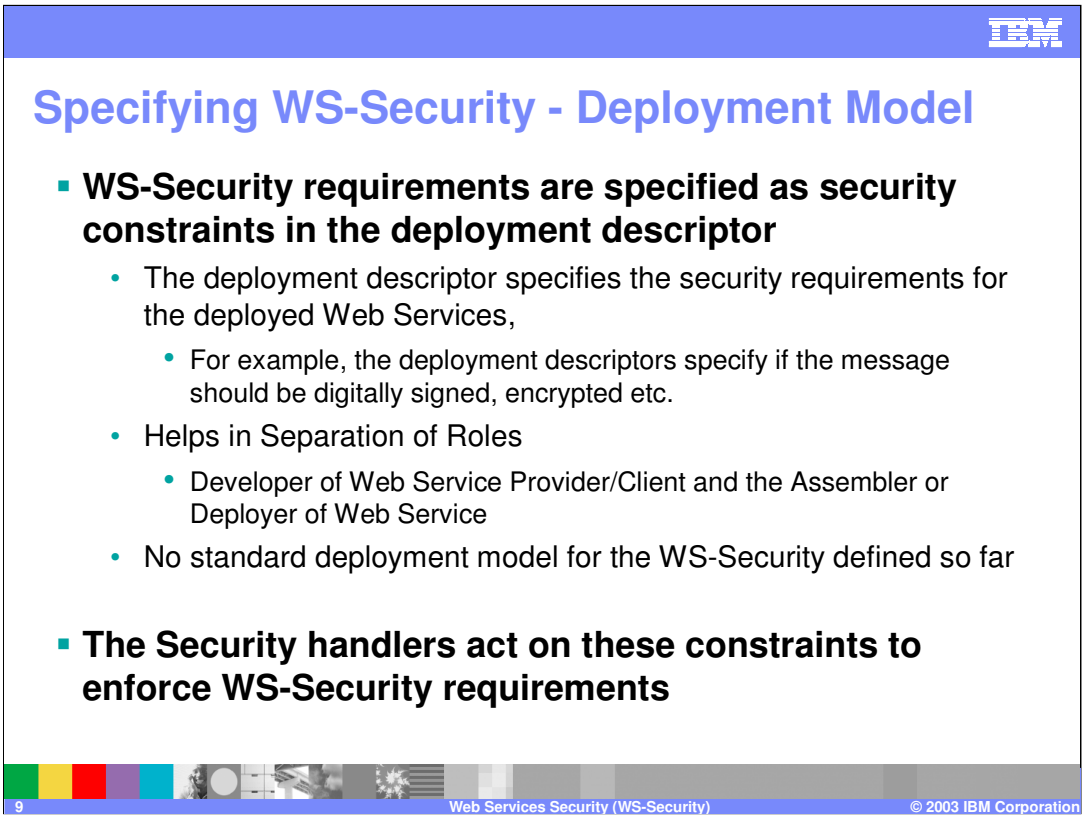
•WS-Security is designed and implemented as message level handlers of the Web Services engine, as a system handler. WS-Security handler is a “system handler” (called Security Handler in the foil) and is registered to the Web Service runtime

•On the client side, the WS-Security handler is invoked to generate the required security headers in the SOAP message before the message is send out to the wire. The security handler generates the security constraint defined in the deployment descriptor and package the security information (digital signature, encrypted data and security tokens) in the SOAP message

•On the server side, the WS-Security handler is called to enforce the declared security constraint in the deployment descriptor prior to dispatching the request to the Web Service EJB or Java Beans implementation.

•This is similar to security interceptors for CSIv2 or SAS.

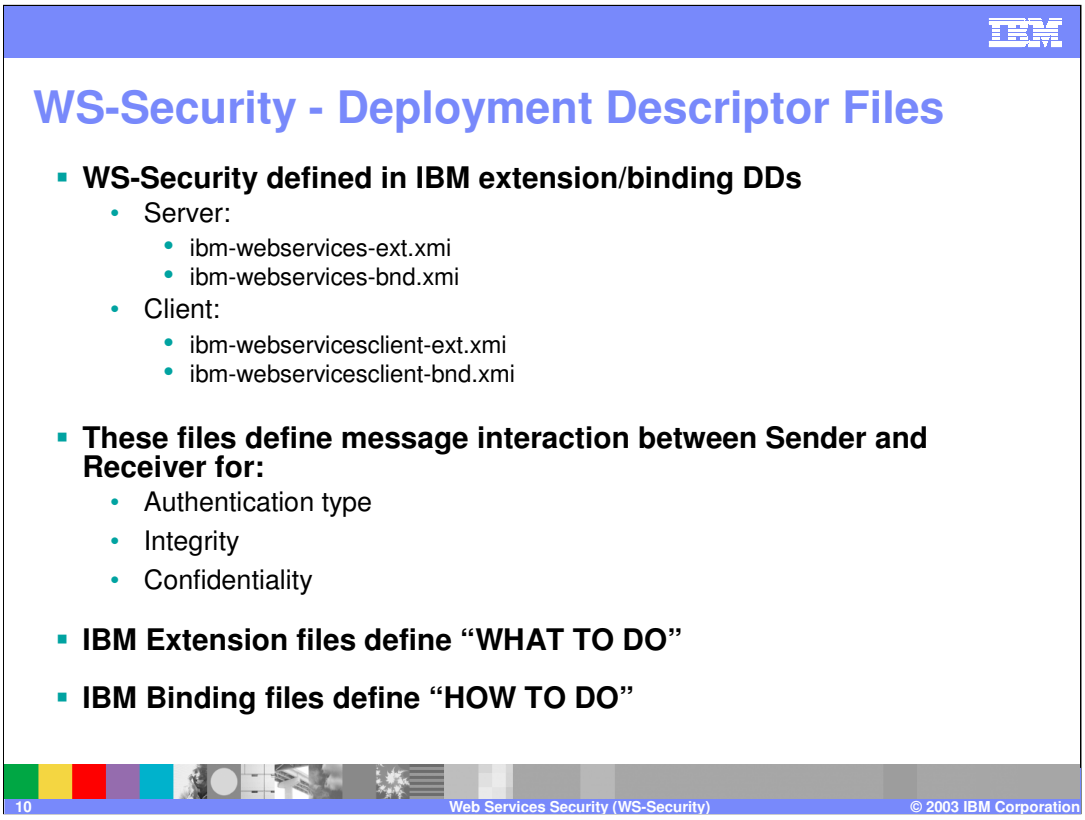
Note: The security constraints of request sender and request receiver must match. Also, the security constraints of the response sender and response receiver must match. For example, if you specify integrity as a constraint in the request receiver, then you must configure the request sender to have integrity applied to the SOAP message. Otherwise, the request is denied because the SOAP message does not include the integrity specified in the request constraint.



The Web services security model employed by WebSphere Application Server is the declarative model. There are no APIs in Version 5.0.2 for programmatically interacting with Web services security, but there are Server Provider Interfaces (SPIs) for extending some security run time behaviors. You can secure an application with Web Services Security by defining security constraints in the IBM extension deployment descriptors and IBM extension bindings. application with Web Services Security by defining security constraints in the IBM extension deployment descriptors and IBM extension bindings.

The development life cycle of a Web services security-enabled application is similar to the Java 2 Platform, Enterprise Edition (J2EE) model since it enabled separation of roles, whereby the component provider creates the J2EE module, then the assembler adds declarative security constraints to the J2EE module. The deployer then takes the web service enabled J2ee application with web services security and deploys it to the runtime environment.

The Web services security handler acts on the security constraints defined in the IBM extension deployment descriptor and enforces the security constraints accordingly.



The security constraints for Web services security are specified in IBM deployment descriptor extension for Web services. The Web services security run time acts on the constraints to enforce Web services security for the Simple Object Access Protocol (SOAP) message. The scope of the IBM deployment descriptor extension is at the module level (EJB or Web). There also are bindings associated with each of the following IBM deployment descriptor extensions:

Client (Might be either a stand-alone client or Web services acting as a client)

ibm-webservicesclient-ext.xmi

ibm-webservicesclient-bnd.xmi

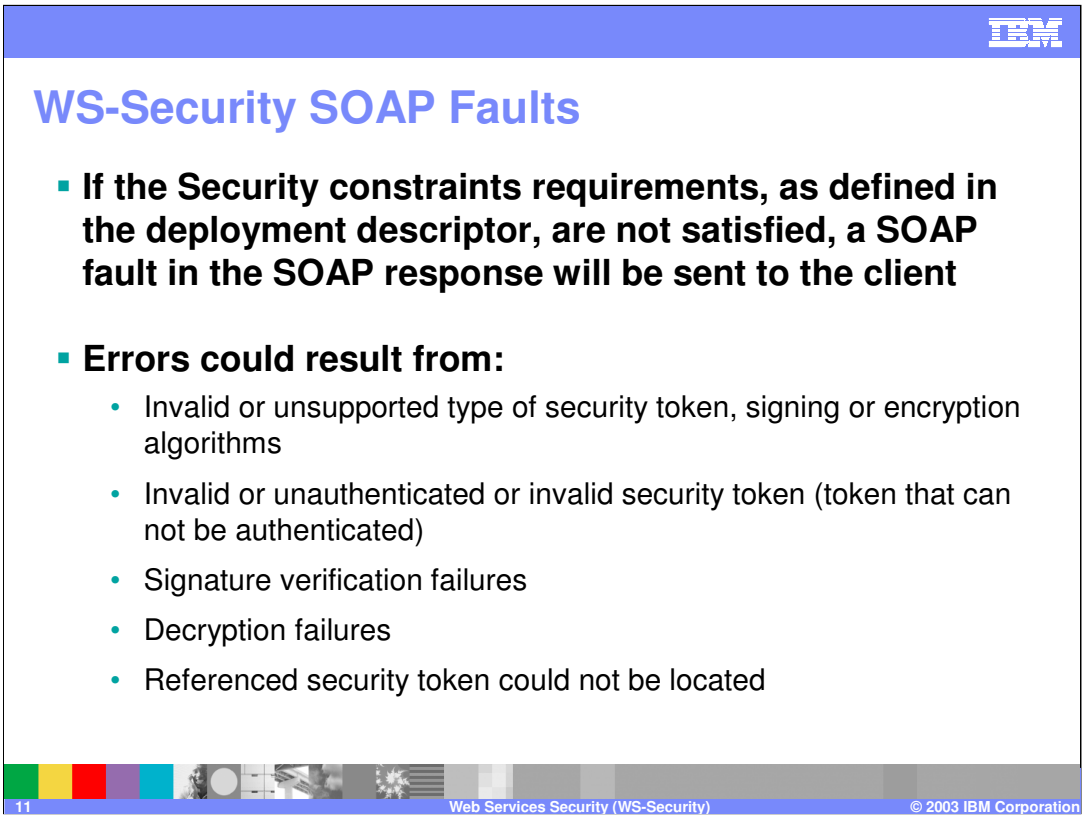
Server

ibm-webservices-ext.xmi

ibm-webservices-bnd.xmi

The IBM extension deployment descriptor and bindings are associated with each EJB module or Web module.

When configuring security for Web services security, the security extensions configuration specifies what security is to be performed while the security bindings configuration indicates how to perform what is specified in the security extensions configuration. You can use the defaults for some elements at the cell and server levels in the bindings configuration, including key locators, trust anchors, the collection certificate store, trusted ID evaluators, and login mappings and reference them from the WAR and JAR binding configurations.



There are many circumstances where an error can occur while processing security information. For example:

Invalid or unsupported type of security token, signing, or encryption

Invalid or unauthenticated or unauthenticatable security token

Invalid signature

Decryption failure

Referenced security token is unavailable.

When a failure occurs, then the failure is reported to the client (sender) using SOAP's Fault mechanism.

Section

WS-Security
Authentication, Integrity, Confidentiality

The following authentication methods are available:

Basic Auth:

Basic authentication includes both a user name and password in the security token. The information in the token is authenticated by the receiving server and used to create a credential.

Signature:

Used when the authentication method is signature where an X.509 certificate is sent as a security token. For Lightweight Directory Access Protocol (LDAP) registries, the distinguished name (DN) is mapped to a credential, which is based on the LDAP certificate filter settings. For Local OS registries, the first attribute of the certificate, usually the common name (CN) is mapped directly to a user ID in the registry. Kerberos tokens are not supported in V5.0.2.

IDAssertion:

The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. Do not attempt to configure identity assertion from a pure client as it will not work. Identity assertion works only when you configure on the client-side of a Web service acting as a client to a downstream Web service. Identity assertion maps a trusted identity (ID) to a WebSphere credential. This authentication method only includes a user name in the security token. An additional token is included in the message for trust purposes. Once the additional token is trusted, the IDAssertion token user name is mapped to a credential. This is used when the authentication method is IDAssertion.

LTPA:

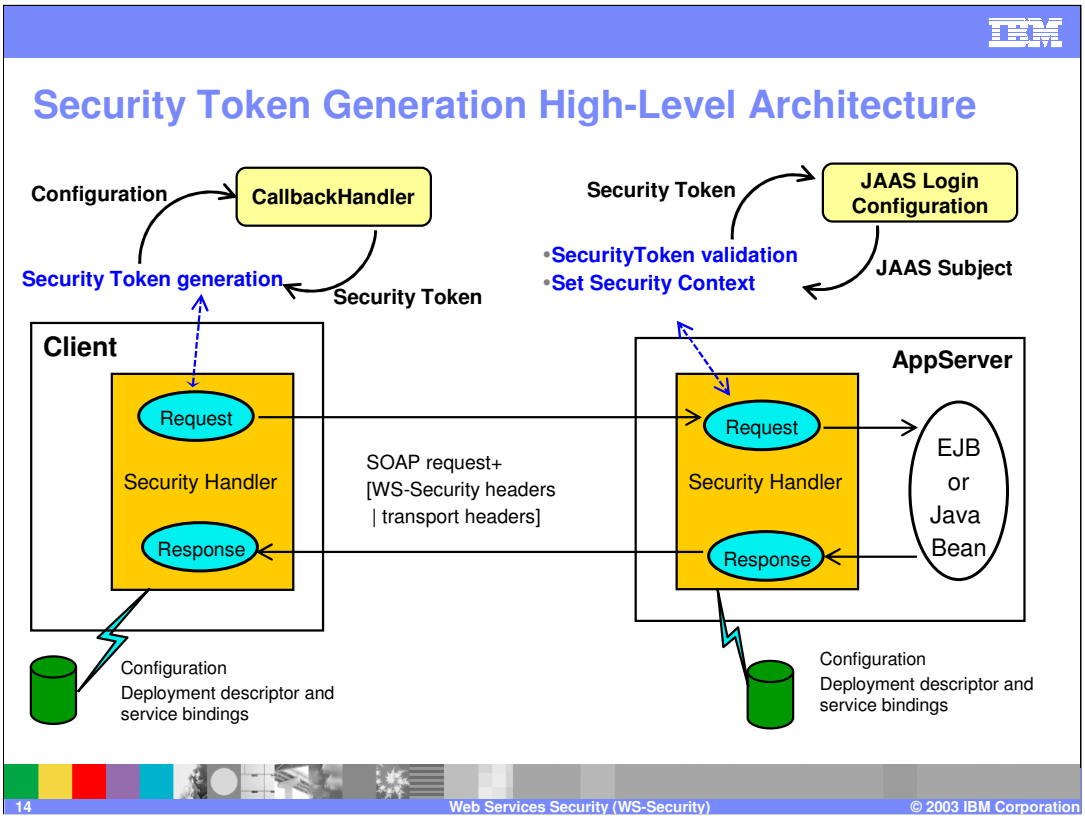
Light-weight Third Party Authentication validates an LTPA token.

Custom Token:

Custom token is pluggable and allows custom-defined tokens to be inserted into the SOAP message.

NOTE: The type of tokens that may be accepted by a message received may be defined at Assembly phase. A receiver may support one or more type of security tokens. The message sender may choose one of the token types that are supported by the receiver when sending a message. The token type to be used by the sending

side is defined in the client descriptor extension ibm-webservicesclient-ext.xmi while the token types that are supported by the receiver will be specified in the ibm-webservices-ext.xmi.



For Security token generation and validation, On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, a Java Authentication and Authorization Service (JAAS) login module is used to validate the security token. These two operations, token generation and token validation are described below:

On the client (sender) side:

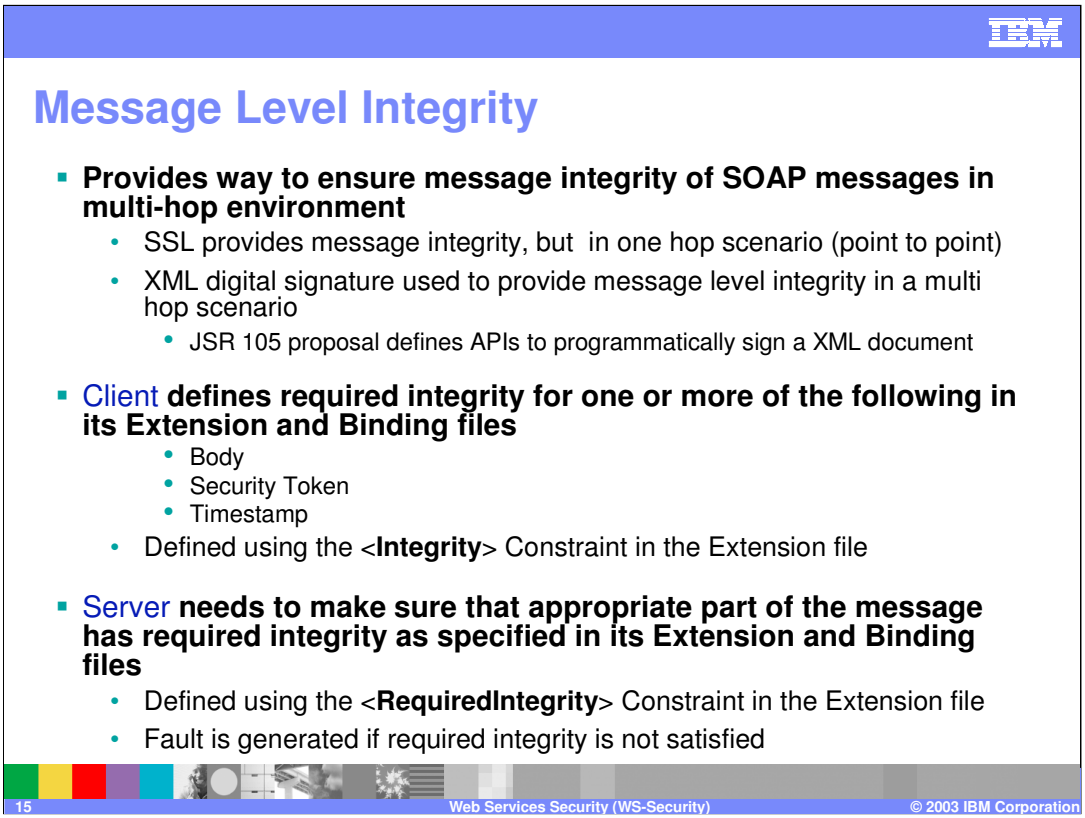
The client Security Handler generates a security token using a callback handler. The security token returned by the callback handler is inserted in the SOAP message. The callback handler that is used is specified in the <LoginBinding> element of the bindings file, ibm-webservicesclient-bnd.xmi. Some callback handler implementations are provided with WebSphere Application Server (as discussed in later foil). You can also add your own callback handlers that implement javax.security.auth.callback.CallbackHandler.

On the Server (receiver) side:

The request receiver retrieves the security token from the SOAP message and validates it using a JAAS login module. If the validation is successful, the login module returns a JAAS Subject. This subject then is set as the identity of the thread of execution. If the validation fails, the request is rejected with a SOAP fault exception.

The JAAS login configuration is specified in the <LoginMapping> element of the bindings file. The configuration information consists of a CallbackHandlerFactory and a ConfigName. The CallbackHandlerFactory specifies the name of a class that is used for creating the JAAS CallbackHandler object. WebSphere Application Server provides the com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl CallbackHandlerFactory implementation. The ConfigName specifies a JAAS configuration name entry. WebSphere Application Server searches the security.xml file for a matching configuration name entry. If a match is not found, it searches the wsjaas.conf file for a match.

WebSphere Application Server provides the WSLogin default configuration entry, which is suitable for the BasicAuth authentication method. WebSphere Application Server similarly provides the system.wssecurity.IDAssertion default configuration entry, which is suitable for the identity assertion authentication method.



Integrity is the property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner.

Web services can be accessed by sending SOAP messages to service endpoints identified by URIs, requesting specific actions, and receiving SOAP message responses (including fault indications). Within this context, the broad goal of securing Web services breaks into the subsidiary goals of providing facilities for securing the integrity and confidentiality of the messages and for ensuring that the service acts only on requests in messages that express the claims required by policies.

The Secure Socket Layer (SSL) along with the de facto Transport Layer Security (TLS) can be used to provide transport level security for web services applications. SSL/TLS offers several security features including authentication, data integrity and data confidentiality. However SSL/TLS enables point-to-point secure sessions. (one hop scenario)

Today's Web service application topologies include a broad combination of mobile devices, gateways, proxies, load balancers, demilitarized zones (DMZs), outsourced data centers, and globally distributed, dynamically configured systems. All of these systems rely on the ability for message processing intermediaries to forward messages. Specifically, the SOAP message model operates on logical endpoints that abstract the physical network and application infrastructure and therefore frequently incorporates a multi-hop topology with intermediate actors.

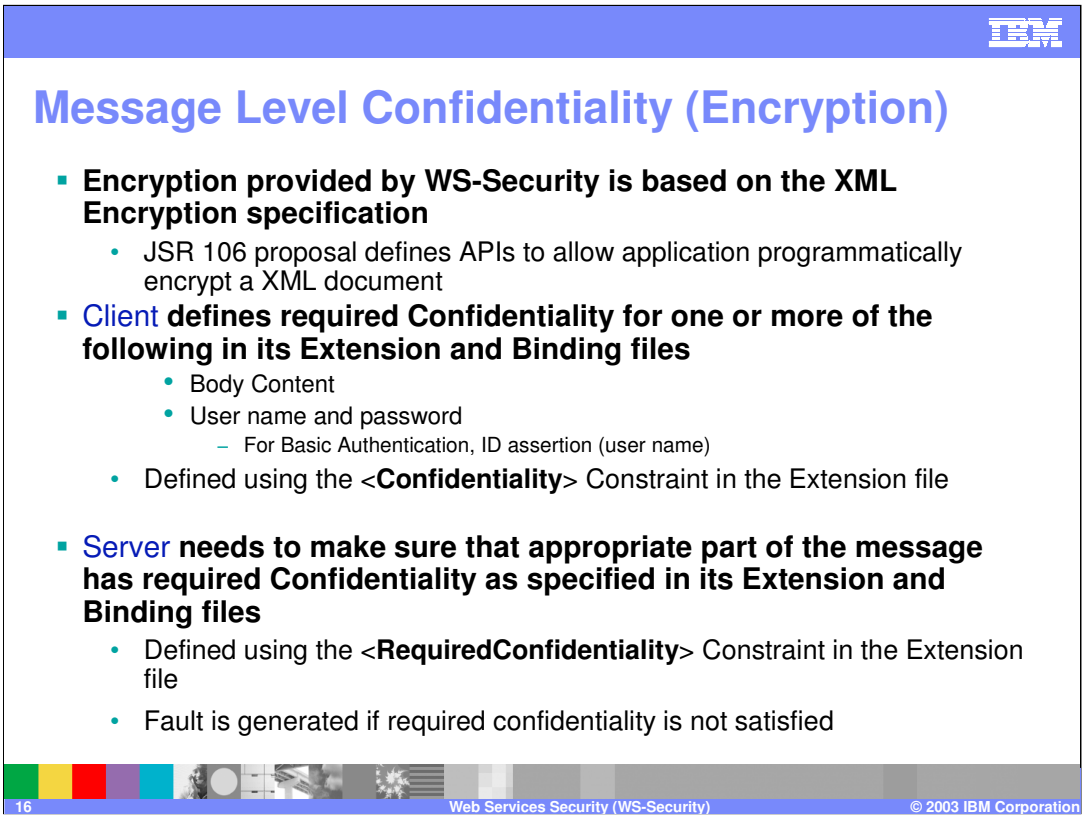
XML digital signature is used to provide integrity in a multi-hop scenario.

Signature is a value generated from the application of a private key to a message via a cryptographic algorithm such that it has the properties of integrity, message authentication and/or signer authentication.

You can select multiple parts of a message to be digitally signed. The different parts that can be digitally signed are Body, Security Token and Time Stamp. This is set in the Integrity constraint in the Extensions file for the client. The security handler on the client side will act on the constraint and digitally sign the message . On the server-side, the Integrity options have to be set in the Required Integrity constraint in the web services Extension file for the server. The signing information is specified in the binding files. The security handler on the server side of the SOAP message enforces these security specifications.

A SOAP Fault is generated, if required SOAP message elements are not properly signed, like:

- No signature could be found in the message
- The signature / signer certificate is invalid

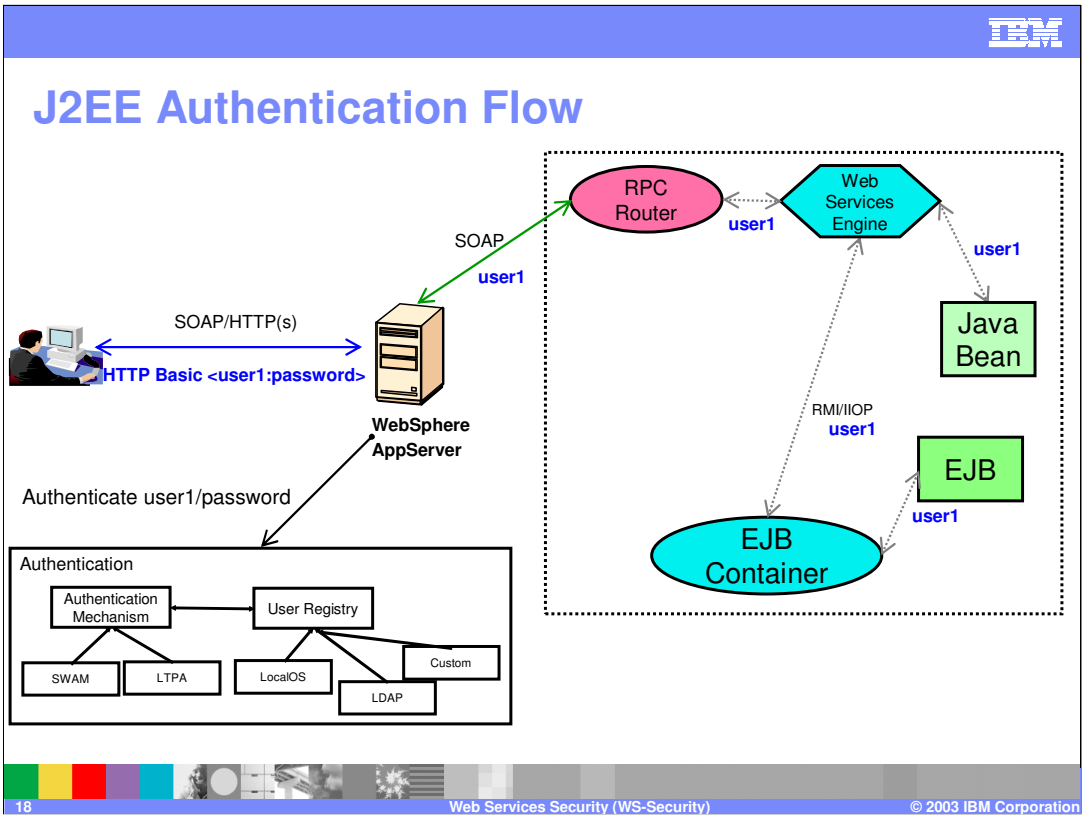


Confidentiality is the process by which data is protected such that only authorized actors or security token owners can view the data. Message Confidentiality is provided by leveraging XML Encryption in conjunction with security tokens to keep portions of SOAP messages confidential. In WebSphere Application Server V5.0.2, there is no way to specify integrity and confidentiality constraints programmatically. This support is coming down the road with support of JSR 105 and JSR 106. Again, Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone being able to understand the message flowing across the Internet. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the correct target.

In WebSphere Application Server V5.0.2, confidentiality has to be specified declaratively as constraints within the client and server web services extension files. In the client's web services extension file, indicate which part of the message needs to be encrypted. Body content is the user data portion of the message. Username token is the basic authentication information. The encryption algorithm and other encryption information has to be specified within the binding file. On the server side, server needs to make sure that the message is encrypted properly and should generate a SOAP fault if the required confidentiality constraints in the deployment descriptor are not satisfied. The server(receiver) side extensions file will specify which part of the message is to be decrypted and the bindings file will specify how to do the decryption.

Section

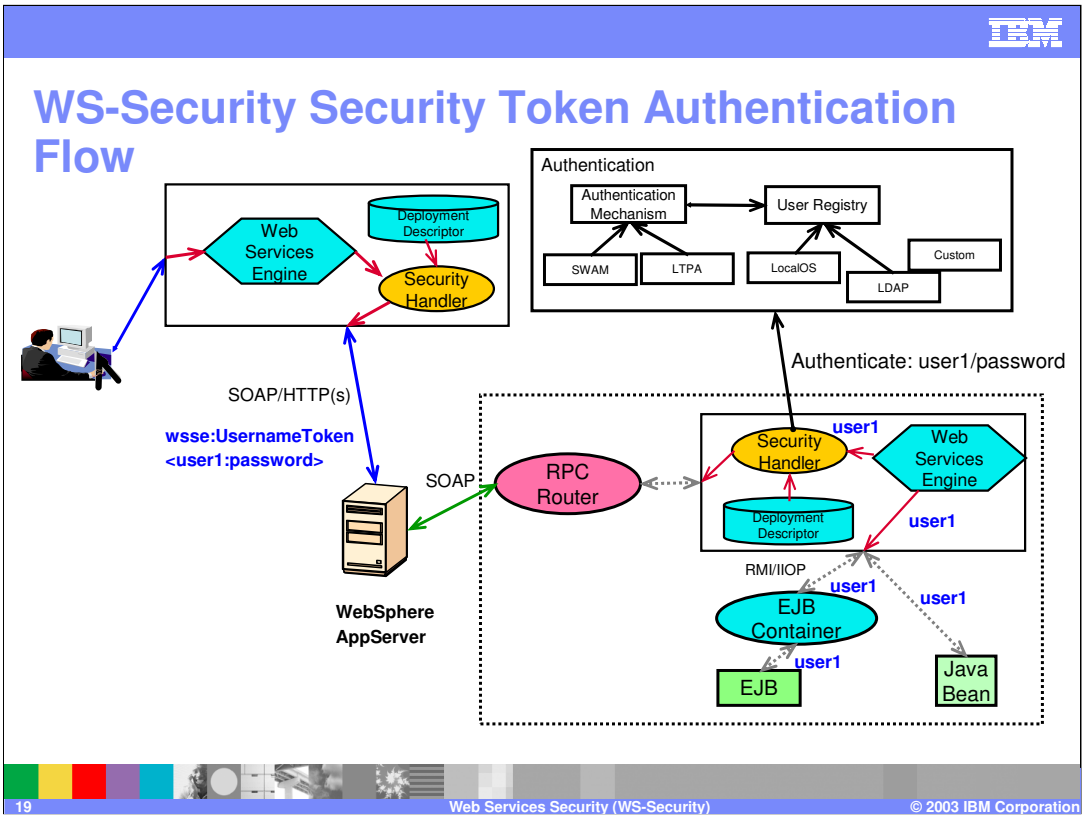
Authentication Flow of Security Tokens



You can secure Web services using the existing security infrastructure of WebSphere Application Server, J2EE role-based security, and SSL transport level security.

The Web services endpoint can be secured using J2EE role-based security. The Web services sender can be used to send the basic authentication data in the HTTP header. SSL (https) can used to secure the transport level. When the WebSphere Application Server receives the SOAP message, the WebContainer authenticates the user (in this example, is "user1") and sets the security context for the call. After this is complete, the SOAP router servlet sends the request to the implementation of the Web services (the implementation can be Java Beans or Enterprise Java Beans). For Enterprise Java Beans implementations, the EJB container performs an authorization check against the identity "user1".

The Web services endpoint can also be secured using the J2EE role. In this case the authorization check is performed before the request is sent. This might be the only way to get to "coarse grain authorization" for Java Bean Web services implementation.



You can also secure Web services using Web services security at the message level. In this case, you can digitally sign or encrypt a certain part of the message. Web services security also supports security token propagation within the SOAP message. This scenario assumes that the Web services endpoint is not secured with J2EE role-based security and that the Enterprise Java Bean is secured with J2EE role-based security.

In this case the Web services endpoint is not secured with J2EE role-based security. The Web services engine processes the SOAP message before the client sends the message to the Web services endpoint. The Web services security runtime acts on the security constraints, such as digitally signing, encrypting, or generating (and inserting) a security token in the SOAP header. In this case <wsse:UsernameToken> is generated using "user1" and "password". On the server-side (receiving), the Web service processes the incoming message and Web services security enforces security constraints. This includes making sure messages are properly signed, properly encrypted, and decrypted, authenticating the security token, and setting up the security context with the authenticated identity. (In this case, "user1" is the authenticated identity.) Finally, the SOAP message is sent to Web service (if the implementation is Enterprise Java Beans, the EJB container performs an authorization check against user1).

As you can see, Web services security can complement J2EE role-based security. For example, SSL can be enabled at the transport level to provide a secure channel, and if the Web services implementation is Enterprise Java Bean you can leverage the EJB authorization by performing authorization checks. The Web services security runtime leverages the security infrastructure in order to set the authenticated identity in the security context. The authenticated identity can be used in the downstream call to J2EE resources (or other resource types).

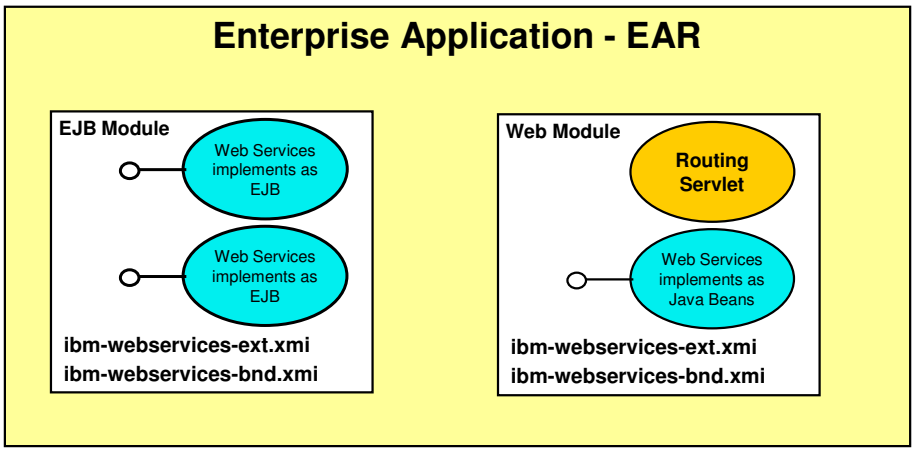
Note that if you use SOAP over JMS, Web services security is the only way to propagate security token from sender to the receiver.

Section

WS-Security
IBM Extension and Binding Files

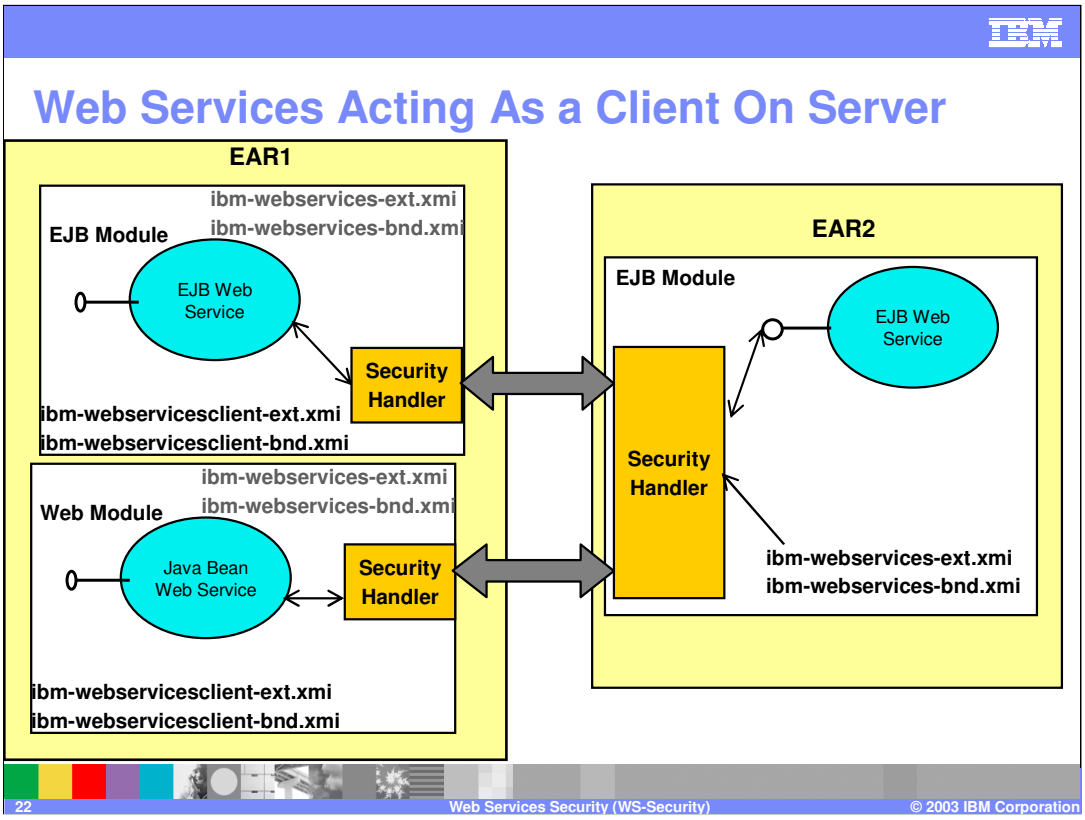
(Defines WS-Security Policies for Client and Provider)

WS-Security Deployment Descriptor - Provider



Example of WS-Security IBM Deployment Descriptors for Web Services provider

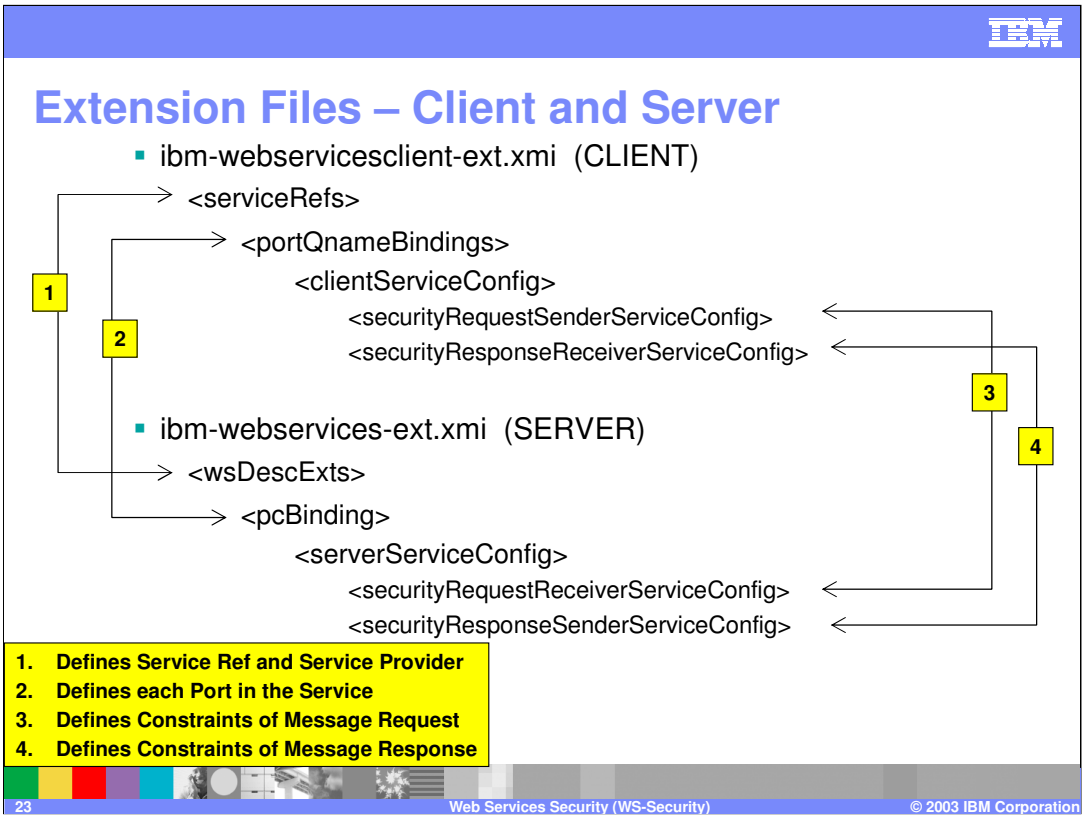
As explained earlier, the security constraints for Web services security are specified in IBM deployment descriptor extensions and bindings for Web services. The Web services security run time acts on the constraints to enforce Web services security for the Simple Object Access Protocol (SOAP) message. The scope of the IBM deployment descriptor extensions and bindings is at the module level (EJB or Web). The IBM extension deployment descriptor and bindings are associated with each EJB module or Web module.



The IBM extension deployment descriptor and bindings are associated with each EJB module or Web module. If Web services is acting as a client, then it has the client IBM extension deployment descriptors and bindings in the

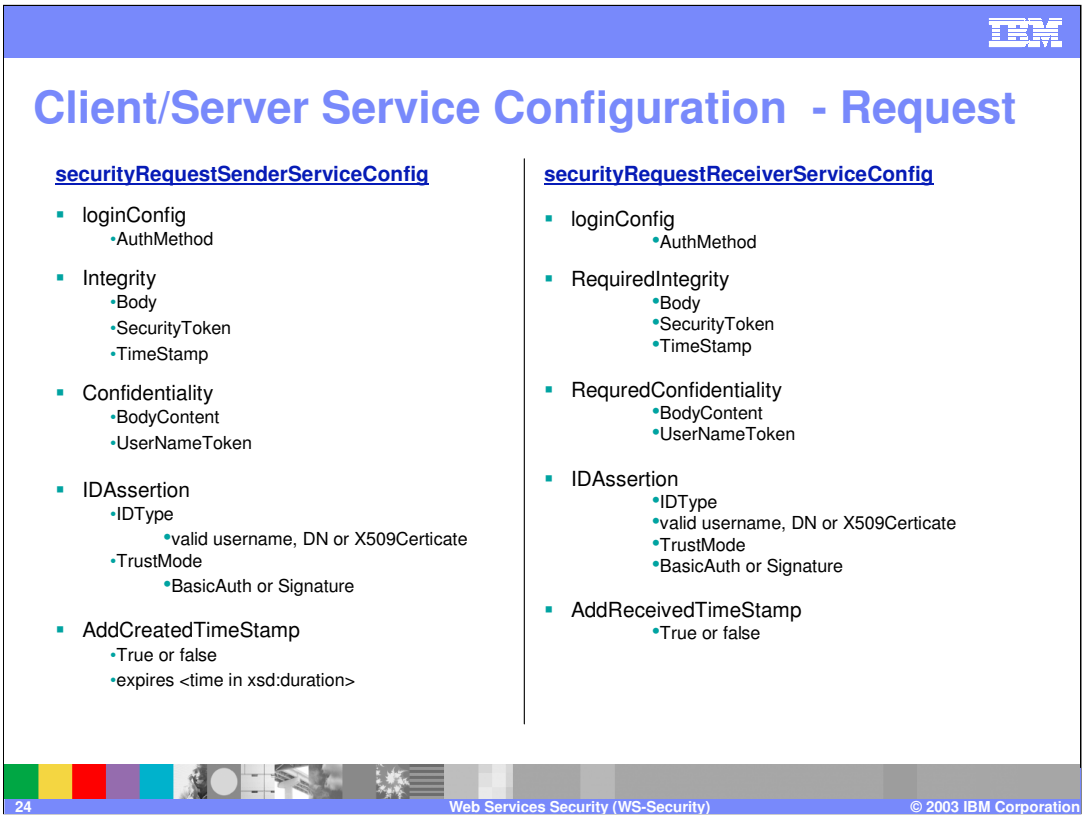
EJB module or Web module as well. In this example, EAR1 has an EJB web service and a Java Bean web service. Hence, the EJB module and the web module will have the server-side binding and extension files (ibm-webservices-bnd.xmi and ibm-webservices-ext.xmi). However these web services also act as clients to an EJB web service on EAR2. Hence, the EJB and Web module will also have client-side binding and extension files.

The Web services security handler acts on the security constraints defined in the IBM extension deployment descriptor and enforces the security constraints accordingly. There are outbound and inbound configurations in both the client and server security constraints.



The Extension Files define what to do. On the top is the format of the web services client extension file and on the bottom is the web services server extension file. The arrows point out the matching pieces of information in these two files. On the client side, the service reference and the service provider. For each port in the service, you specify the client service configuration. There are two bodies of information in the client service configuration namely the Request Sender configuration and the Response Receiver Configuration. All the security constraints that needs to be done before sending the message to the server has to be specified in the securityRequestSenderServiceConfig part. The matching part for this in the server extension file is the securityRequestReceiverServiceConfig. Hence, all the message request constraints (Integrity, Confidentiality, Login Configuration…) will be specified here.

For the message response from the server, the parts securityResponseReceiverServiceConfig on the client side and the securityResponseSenderServiceConfig on the server side will specify the security constraints.



Let's look at the contents of the extensions files in more detail. On the left is the Request Sender configuration which is a part of the ibm-webservicesclient-ext.xmi file. These constraints apply both to J2EE application clients or when Web services is acting as a client. You can specify the following security requirements for the **request sender** and apply them to the SOAP message:

loginConfig: You can specify the security token that you want to insert into the message here. Various security token options are available here such as Basic authentication, Identity assertion, X.509 binary security token, LTPA binary security token, Custom token.

Integrity: You can select multiple parts of a message to be digitally signed. The options are Body, Security Token and Time Stamp.

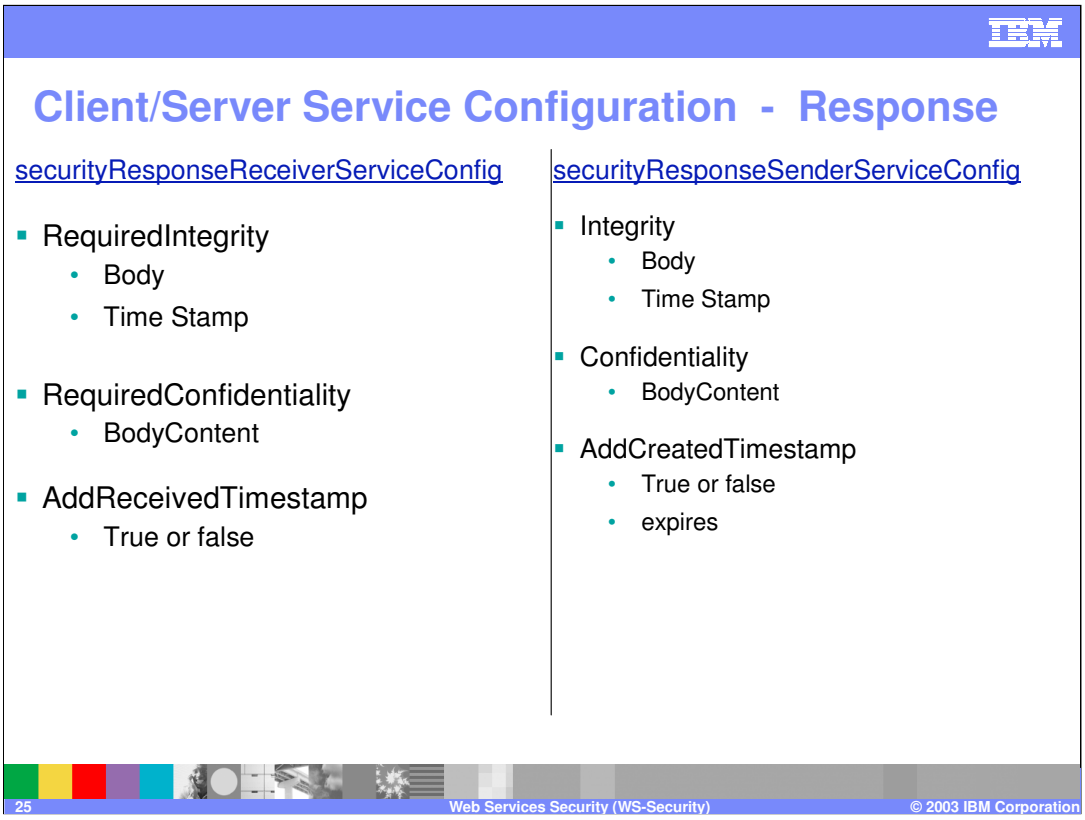
Confidentiality: You can select multiple parts of a message to be encrypted. The options are BodyContent and UserNameToken

IDAssertion (use only if you selected Identity Assertion as the Auth Method in the LoginConfig section): The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. In order for the downstream Web service to accept the identity of the originating client (just the user name), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on the downstream Web service configuration.

AddCreatedTimeStamp: You can have a time stamp for indicating the freshness of the message.

On the right is the Request Receiver configuration which is a part of the ibm-webservices-ext.xmi file. The request receiver configuration defines the security requirement of the SOAP message. If the incoming SOAP message does not meet all the security requirements defined, then the request is rejected with the appropriate fault code returned to the message. For security tokens, the token is validated using Java Authentication and Authorization Service (JAAS) login configuration and authenticated identity is set as the identity for the downstream invocation.

Note: The security constraint for request sender must match the security requirement of the request receiver for the request to be accepted by the server.

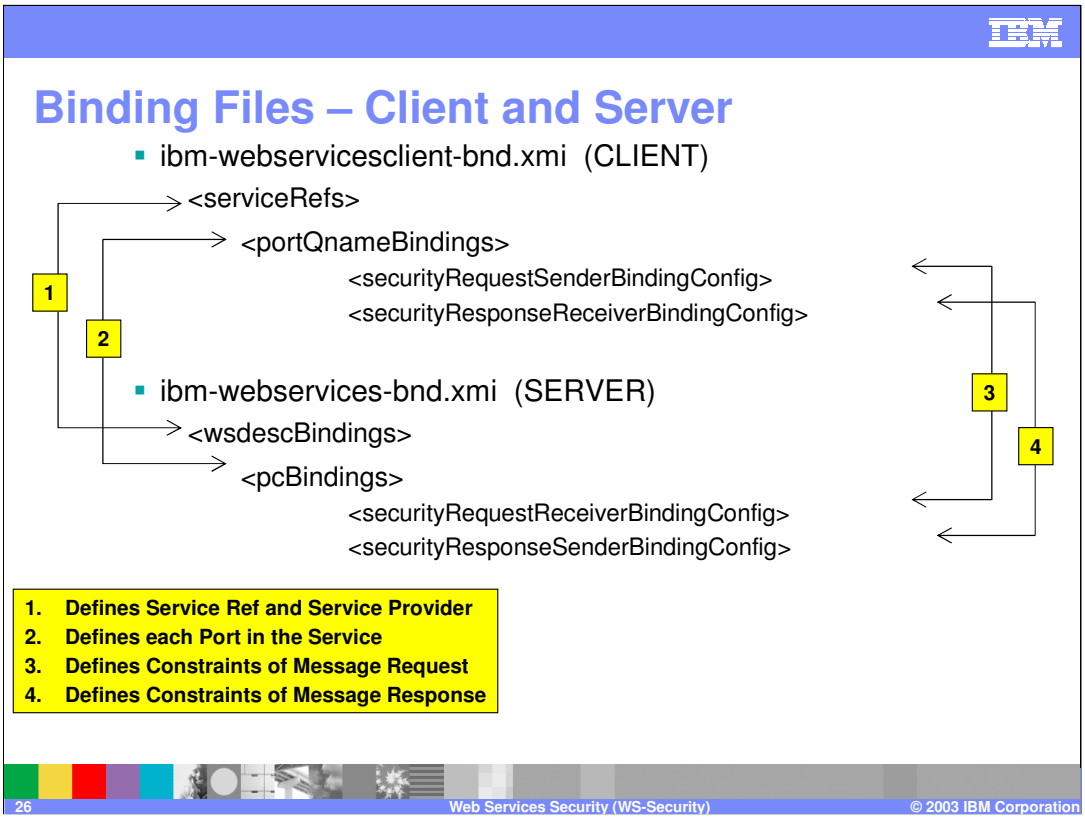


Let's look at the security constraints in the extensions files for the security response. On the left is the Response Sender configuration which is a part of the ibm-webservices-ext.xmi file. The response sender configuration defines the security requirements of the SOAP response message. The security handler signs, encrypts, or generates the time stamp for the SOAP response message before the response is sent to the caller.

In the Integrity section , you can select multiple parts of the message to be digitally signed. In the Confidentiality section you can encrypt the body content of the message. You can also have a time stamp for checking the freshness of the message. Also, you can specify an expiration time for the time stamp, which helps defend against replay attacks.

On the left is the Response Receiver configuration which is a part of the ibm-webservicesclient-ext.xmi file. The response receiver configuration defines the security requirements of the response received from a request to a Web service. You can specify Required Integrity, Required Confidentiality and Received TimeStamp for the ResponseReceiver configuration

Keep in mind that the security constraints for response sender must match the security requirements of the response receiver. If the constraints do not match, the response is not accepted by the caller or sender.



The Binding Files describe how to execute the security specifications found in the extensions.

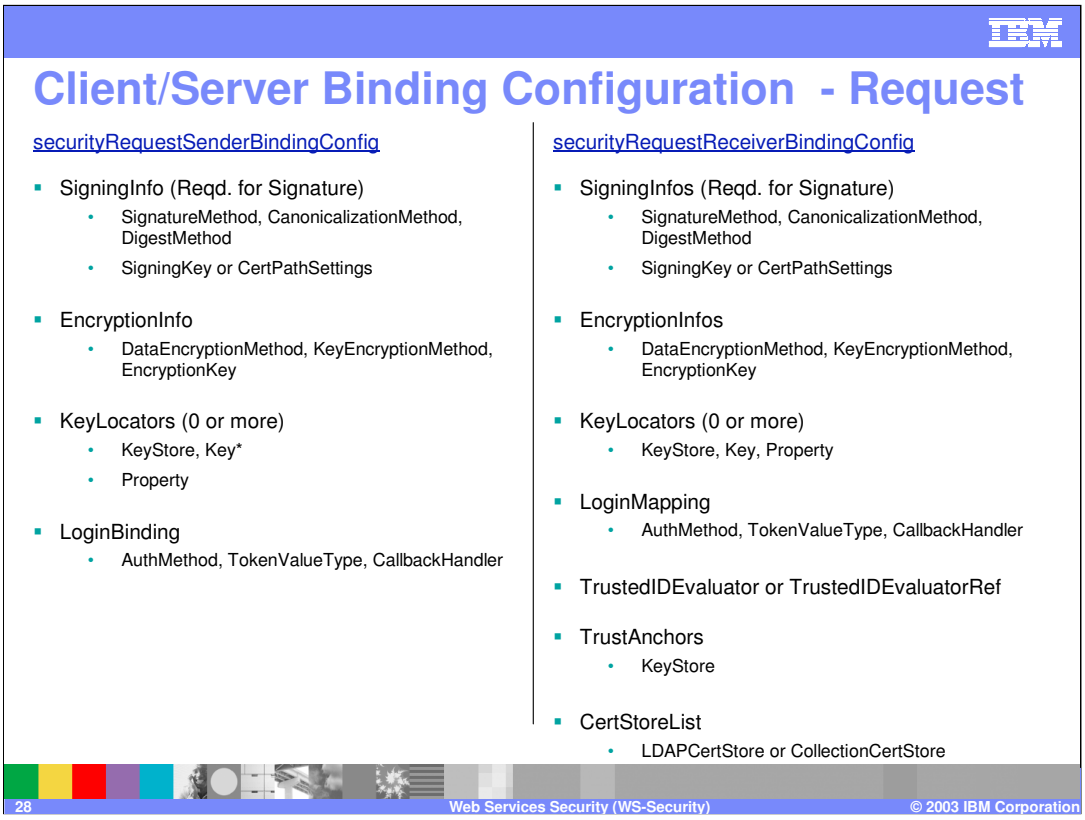
On the top is the format of the web services client binding file and on the bottom is the web services server binding file. The arrows point out the matching pieces of information in these two files. On the client side, the service reference and the service provider have to be specified. For each port in the service, you specify the client security bindings. There are two bodies of information in the client security bindings namely the Request Sender configuration and the Response Receiver Configuration. All the security constraints that needs to be done before sending the message to the server has to be specified in the securityRequestSenderBindingConfig part. The matching part for this in the server binding file is the securityRequestReceiverBindingConfig.

For the message response from the server, the parts securityResponseReceiverBindingConfig on the client side and the securityResponseSenderBindingConfig on the server side will specify the security constraints.

WS-Security Binding - Important Elements

- TrustAnchorList**: trusted root certificates for signature verification**
- CertStoreList**: CRLs and non-trusted certificates verification**
- KeyLocators**: locates the keys used for retrieving signing (verification) / encryption (decryption) key**
- TrustedIDEvaluators**: evaluates the trust of the received identity before identity assertion**
- LoginMappings**: JAAS configurations for <AuthMethod> specified in the deployment descriptors**

CRL – Certificate Revocation List



SigningInfo: Specifies the required information for signature and its verification.

•Sender side: SigningKey must be specified. This denotes an abstract key name and this is resolved by KeyLocators.

•Receiver side: Currently only X.509 is supported. CertPathSettings must be specified. Two "ref" attributes refer to the "name" attribute of TrustAnchor and *CertStore element.

EncryptionInfo: Specifies a triple of DataEncryptionMethod, KeyEncryptionMethod and EncryptionKey. Here EncryptionKey refers to a abstract "name" of the key and the name is resolved by KeyLocators. If the name is absent, locatorRef attribute must be present and the actual key is resolved by the specified KeyLocator. If KeyEncryptionMethod is present, XML message is encrypted using a random-generated key and the key is encrypted using the key specified in EncryptionKey. Otherwise, XML message is encrypted using the key specified in EncryptedKey.

KeyLocator: Specifies the name of a class that chooses the key for signature/encryption using a name given as input or its internal mechanism. This class must implement com.ibm.xml.soapsec.KeyLocator. Properties which are used for initialization of the class can be specified as well. Some kinds of implementation are possible such as:

•Always returns a fixed (default) key

•Simply maps the recipient key name to an alias in KeyStore

•Maps an authenticated client ID to an alias in KeyStore

If the locator returns a key, the handler uses it. Otherwise, the handler tries the other locators in the list by turns in the order of their occurrence. Most common configuration is to have keystore and key specified.

LoginBinding:

Specifies the class name of CallbackHandler implementation to provide login information, per each AuthMethod. These classes must implement javax.security.auth.callback.CallbackHandler. LoginConfig/AuthMethod must much one of LoginBinding/AuthMethod's. For AuthMethod's which use UsernameToken (that is, BasicAuth and IDAssertion), TokenValueType can be omitted (ignored if any). If basic-auth element is specified, the userID and the password are passed to the CallbackHandler (it has nothing to do with AuthMethod of BasicAuth). The Property is the catch all.

TrustedIDEvaluator orTrustedIDEvaluatorRef: Specifies the class name which evaluates whether the received ID is trusted. This class must implement com.ibm.xml.soapsec.token.TrustedIDEvaluator.

TrustAnchor: Trusted root certificates for signature verification

CertStoreList: CertStores are the place to look for CRLs and non-trusted certificates. LDAPCertStore is very experimental

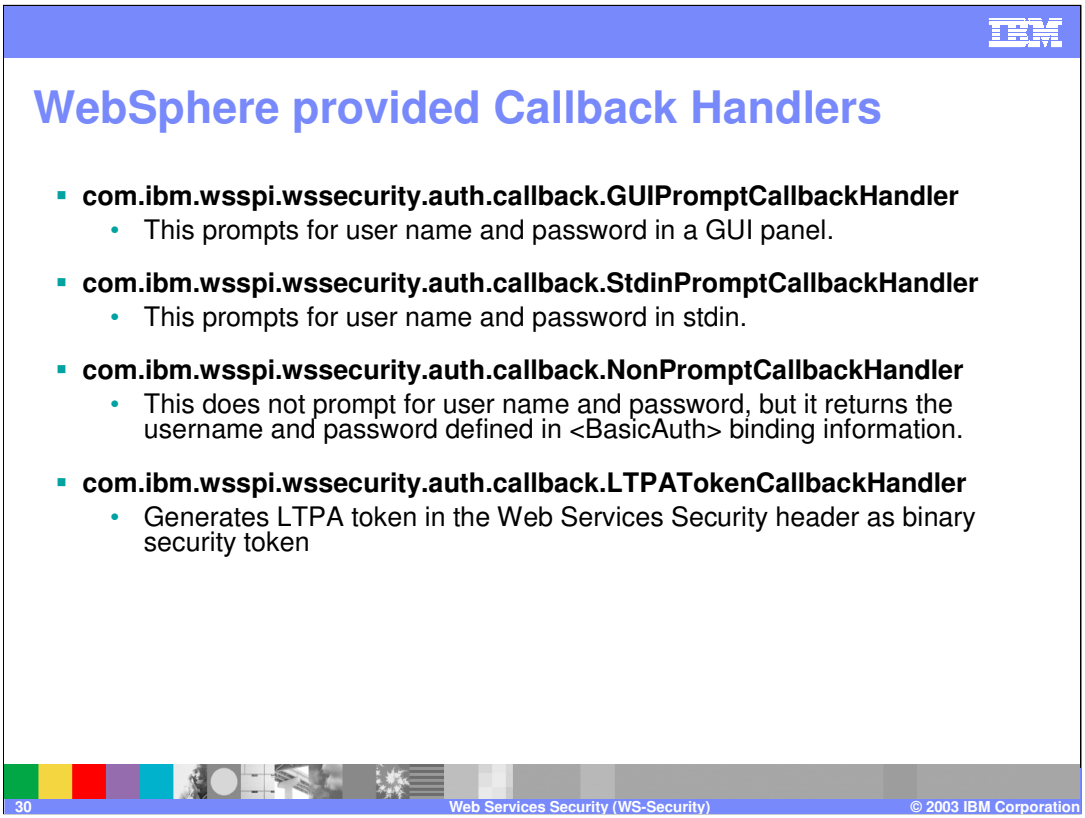
Client/Server Binding Configuration - Response

securityResponseReceiverBindingConfig

- SigningInfo (Reqd. for Signature)
  - SignatureMethod, CanonicalizationMethod, DigestMethod
  - SigningKey or CertPathSettings
- EncryptionInfo
  - DataEncryptionMethod, KeyEncryptionMethod, EncryptionKey
- KeyLocator (0 or more)
  - KeyStore, Key*
  - Property
- TrustAnchor
  - KeyStore
- CertStoreList
  - LDAPCertStore or CollectionCertStore

securityResponseSenderBindingConfig

- SigningInfo (Reqd. for Signature)
  - SignatureMethod, CanonicalizationMethod, DigestMethod
  - SigningKey or CertPathSettings
- EncryptionInfo
  - DataEncryptionMethod, KeyEncryptionMethod, EncryptionKey
- KeyLocator (0 or more)
  - KeyStore, Key*
  - Property



The JAAS CallbackHandler APIs is used for token generation by the request sender. This can be extended to generate custom token to be inserted in the Web services security header. If BasicAuth is configured as the required security token, specify the CallbackHandler in the binding file to collect the basic authentication data. The following are the default implementations provided by WebSphere Application Server:

com.ibm.wsspi.wssecurity.auth.callback.GuiPromptCallbackHandler

The implementation prompts for BasicAuth information (user name and password) in a GUI panel. Use this implementation in the client environment only.

com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler: Collects the basic authentication data in the standard in (stdin). Use this implementation in the client environment only.

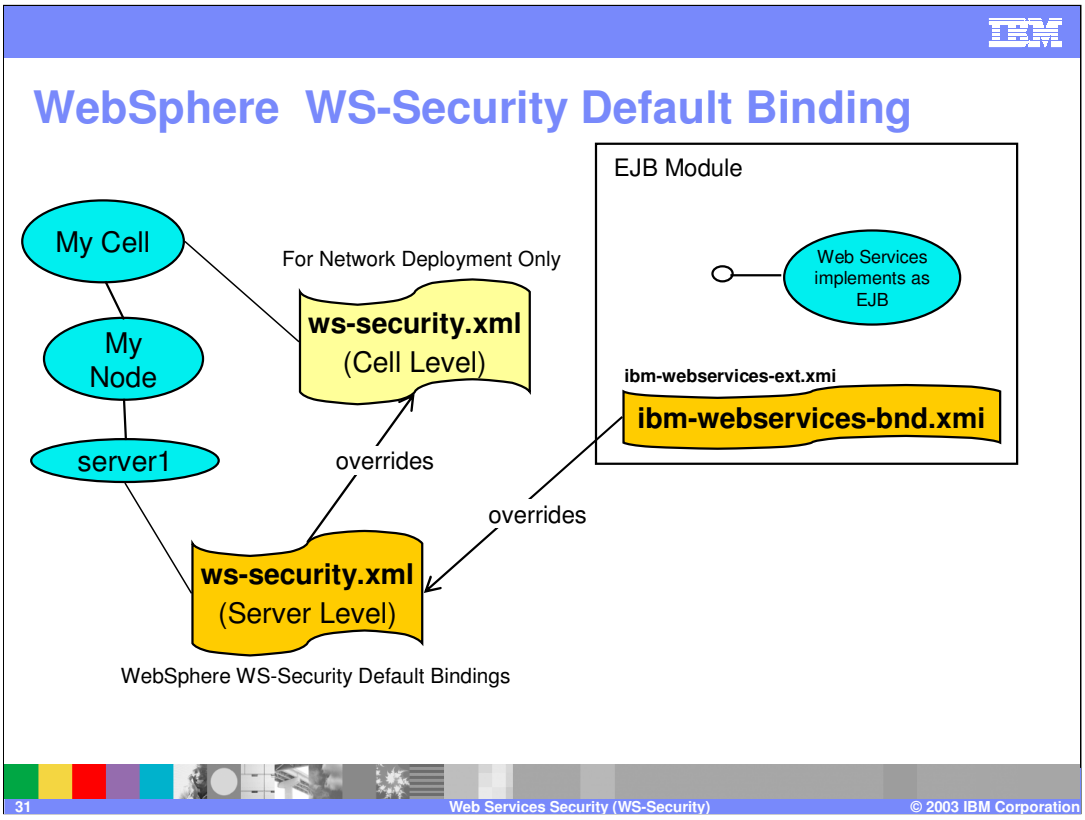
com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler

This implementation reads the BasicAuth information from the binding file. This might be used on the server side to generate a user name token.

com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler:

Generates LTPA token in the Web services security header as binary security token. If there is basic authentication data defined in the application binding file, it is used to perform a login, extract the LTPA token from the WebSphere credentials, and insert the token in the Web services security header. Otherwise, it will extract the LTPA security token from the invocation credentials (run as identity) and insert the token in the Web services security header.

Call Back Handler information has to be specified in the client's web services bindings file. To configure the login binding information, click Applications > Enterprise Applications > application_name. Under Related Items, click Web Module > URI_file_name Web Services: Client Security Bindings. Under Request Sender Bindings, click Edit > Login Binding. If the web services client is a J2EE application client, then use the ATK to modify the client web services bindings.



Some of the binding information could be shared between applications, for example, the trust stores, key stores, authentication method (token validation) and etc..

WebSphere Application Server provides support for default binding information. Administrators can define binding information at the server level and cell level (Network Deployment only) and applications can refer to the binding information. The default binding information is defined in ws-security.xml and can be administered by the Admin Console or scripting.

In the Base Application Server, each server has a copy of ws-security.xml (default binding information for Web services security). There is no cell level copy of

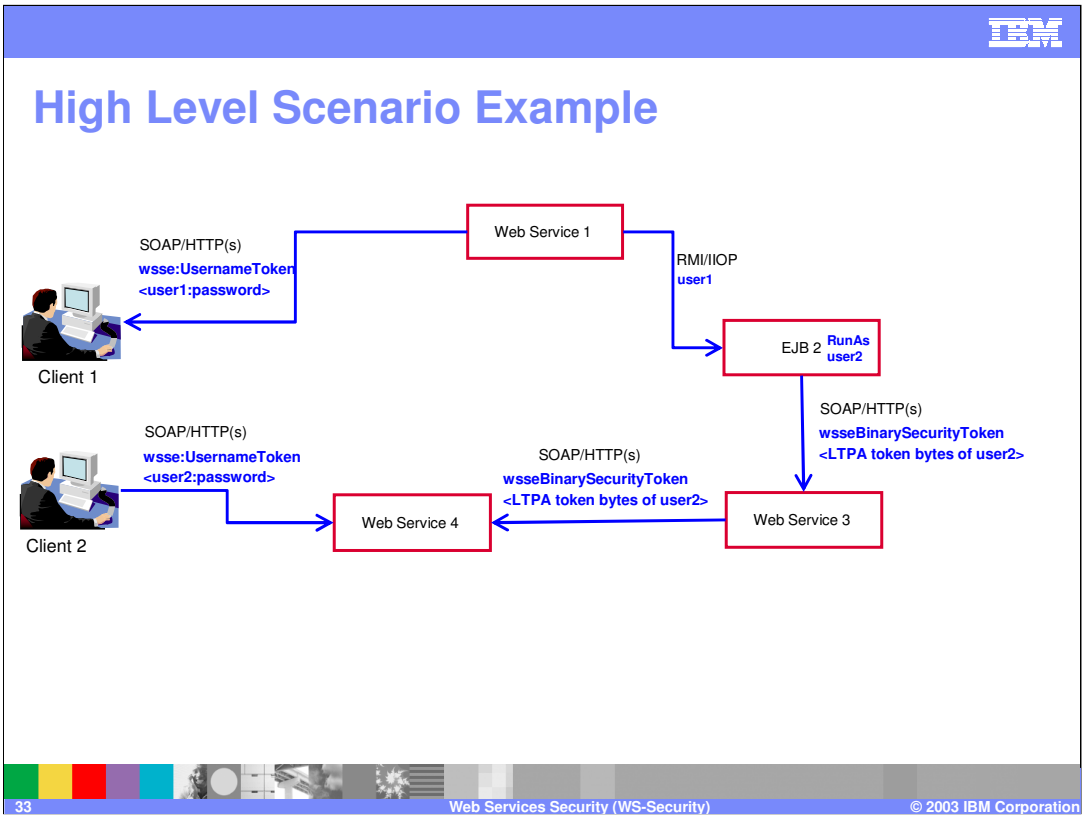
ws-security.xml which is only available on the Network Deployment installation. To navigate to the server level of default binding in the Admin Console, "Servers > Application Servers > server1" and click the link "Web Services: Default bindings for Web Services Security" link in the "Additional Properties“ section.

The Web services security runtime uses the binding information in the application EJB or Web module binding file (ibm-webservices-bnd.xmi or ibm-webservicesclient-bnd.xmi if Web services is acting as client in the server) if the binding information is defined in the application level binding file. For example, if key locator "K1" is defined in both the application level binding file and the default binding (ws-security.xml), then the "K1" in the application level binding file is used.

When the Base Application Server is federated to a Network Deployment cell, the default binding file (ws-security.xml) of the server is added along with other server level configuration to the new cell. If the desire is to use the cell level default binding, then the entries of the server level default binding has to be removed. There is a cell level default binding (ws-security.xml) for Network Deployment installation. In Network Deployment installation, server level binding is optional. To navigate to the cell level of default binding in the Admin Console, "Security > Web Services“.

Section

Scenarios



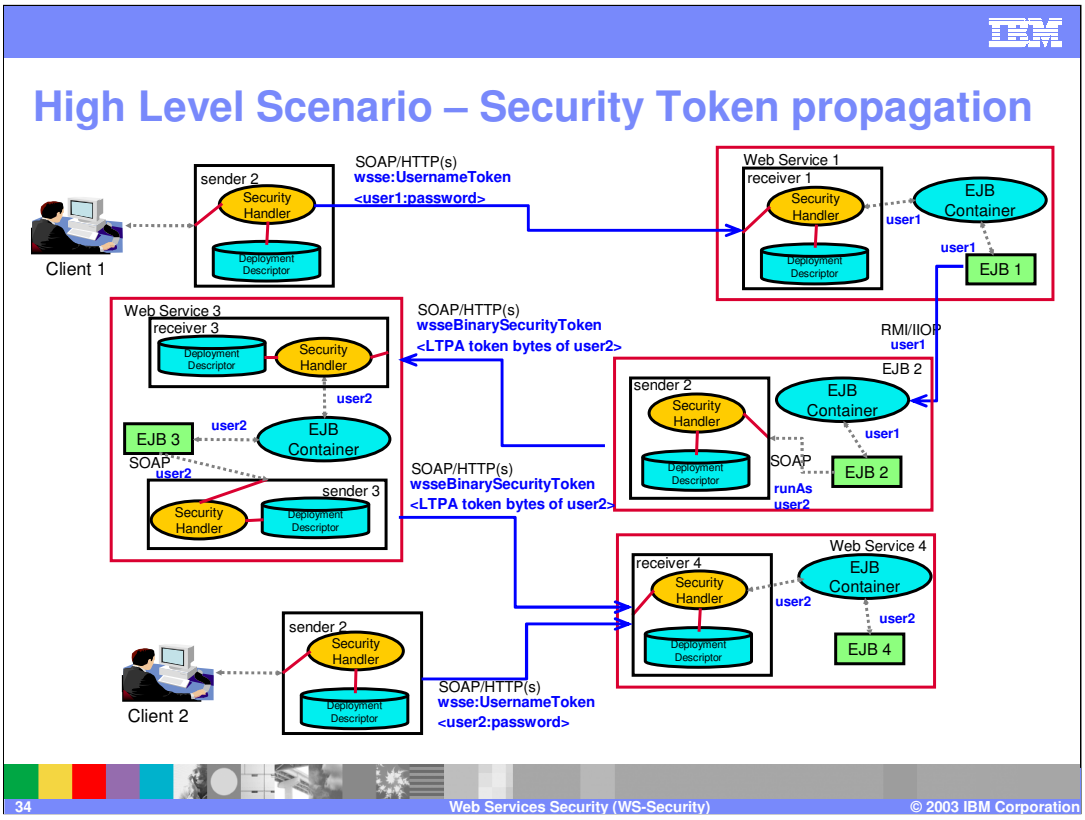
In the above figure, client 1 invokes Web service 1. Then Web Service 1 calls EJB 2. EJB 2 calls Web Service 3 and Web Service 3 calls Web Service 4. The Client 2 also calls web service 4.

The above scenario shows how to propagate security tokens using Web services security, the security infrastructure of the WebSphere Application Server, and Java 2 Platform, Enterprise Edition (J2EE) security. Web service 1 is configured to accept <wsse:UsernameToken> only and use the BasicAuth authentication method. However, Web services 4 is configured to accept either <wsse:UsernameToken> using the BasicAuth authentication method or Lightweight Third-party Authentication (LTPA) as <wsse:BinarySecurityToken>.

Client 1 sends a Simple Object Access Protocol (SOAP) message to Web services 1 with user1 and password in the <wsse:UsernameToken> element. The user1 and password values are authenticated by the Web services security run time and set in the current security context as the Java Authentication and Authorization Service (JAAS) Subject. Web services 1 invokes EJB2 using the Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) protocol. The user1 identity is propagated to the downstream call. The EJB container of EJB2 performs an authorization check against user1. EJB2 calls Web services 3 and Web services 3 is configured to accept LTPA tokens. The RunAs role of EJB2 is set to user2. The LTPA CallbackHandler implementation extracts the LTPA token from the current JAAS Subject in the security context and Web services security run time inserts the token as <wsse:BinarySecurityToken> in the SOAP header. The Web services security run time in Web service 3 calls the JAAS Login Configuration to validate the LTPA token and set it in the current security context as the JAAS Subject.

Web service 3 is configured to send LTPA security to Web service 4. In this case, assume that RunAs is not configured for Web services 3. The LTPA token of user2 is propagated to Web services 4. Client 2 uses the <wsse:UsernameToken> element to propagate the basic authentication data to Web services 4.

Web services security compliments the WebSphere Application Server security run time and the J2EE role-based security. In this case, it demonstrates how to propagate security tokens across multiple resources such as Web services and EJB files.



The same scenario as in previous page is depicted in more detail here. Again, client 1 invokes Web service 1. Then Web Service 1 calls EJB 2. EJB 2 calls Web Service 3 and Web Service 3 calls Web Service 4. When the client1 calls web service1 , since security is passed through the SOAP message, the client-side security handler will insert the necessary basic authentication information as specified in the client web services security extension and binding files. The security handler on webservice 1 will authenticate the request and then calls the EJB1. The EJB will do the necessary role-based authorization. If the user1 is authorized to access the EJB method, then the EJB call is made. EJB1 makes an RMI/IIOP call to EJB2. EJB2 is the web service client to web service 3. Hence, the security handler will insert the security information by introspecting the web service client extension and binding files in EJB2's EJB Module. In this case, a binary security token is send to web service 3. The security handler on web service 3 authenticates the request and then passes the request to web service 4

Client2 is a simple scenario whereby the security handler introspects the client web services extension and binding files and adds the integrity, confidentiality etc to the SOAP message. On the server side for the web service 4, the security handler will introspect the server side web services extension and binding files and authenticate the incoming request. Also, note that the request receiver on web service 4 can accept either Binary Security Tokens or Basic Authentication token. Client 2 uses the Basic Auth token where as web service3 (acting as client to web service 4) uses the Binary Security token

Section

Tools

Tools to specify WS-Security Constraints

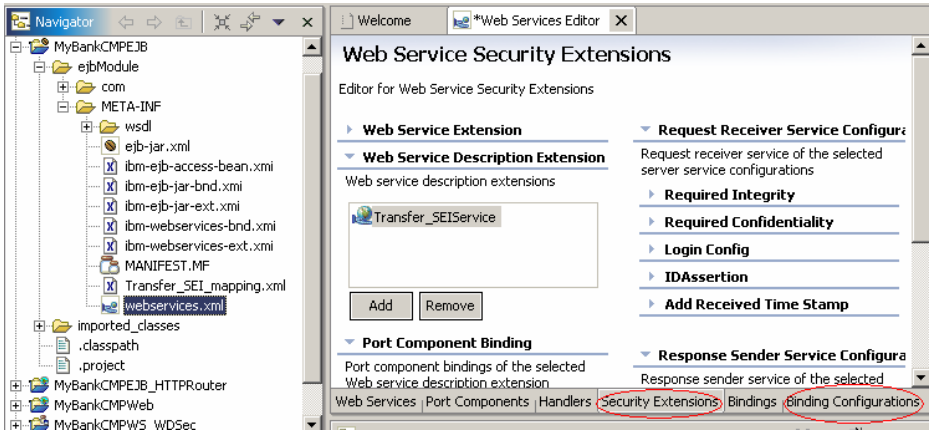
- **Application Server Toolkit and WebSphere Studio V5.1**
  - **Specify WS-Security bindings**
  - **Specify WS-Security extensions**
- **Admin Console**
  - **Create/Update application WS-Security bindings**
  - **Create/Update default WS-Security bindings**
    - **View/Update server-side default WS-Security bindings**

Use either the Application Server Toolkit or WebSphere Studio Application Developer V5.1 to specify the security constraints. XML editors are available for editing the ibm-webservices-bnd.xmi, ibm-webservices-ext.xmi, ibm-webservicesclient-bnd.xmi and ibm-webservicesclient-ext.xmi files. The WebSphere Application Server Administrative Console can be used to view the web services security extensions and the bindings for the client and server. The Administrative Console can be used to update the webservices security client and server-side bindings. However , note that this is not applicable to J2EE application clients. For J2EE application clients, you will have to use the Application Server Toolkit itself to create all the bindings. For clients residing in the server (servlets etc), we can use the Admin Console to create the bindings.

The Admin Console can also be used to view and update the default web services security bindings. So, an administrator can define binding information at the server level and applications can refer to this binding information.

Application Server Toolkit – Web Services Editor

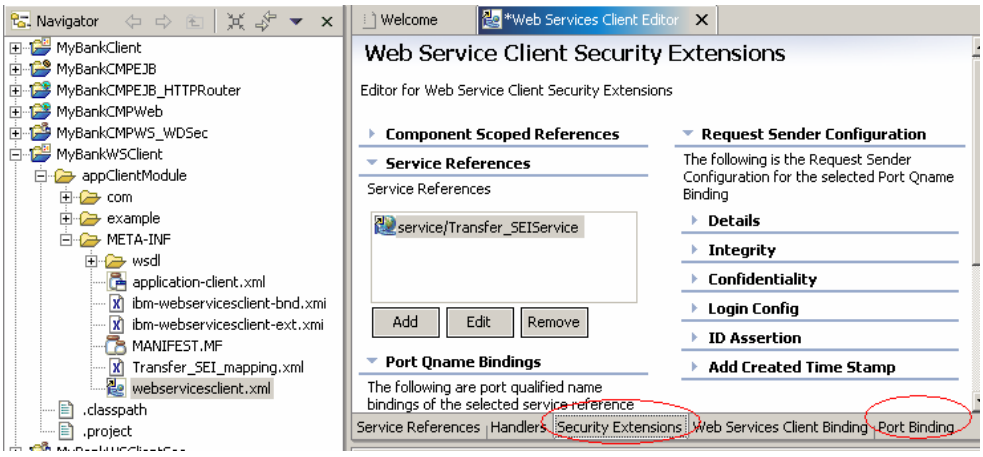
- Specify server-side WS-Security constraints
  - Security Extensions (ibm-webservices-ext.xmi)
  - Binding Configurations (ibm-webservices-bnd.xmi)
- Same editor available in WebSphere Studio V5.1



The server side web services security constraints can be specified using the Application Server Toolkit. Import the web service application to the tool. If it is an EJB web service, then expand ejbmodule > META-INF. Double click webservices.xml. The web services security extensions can be edited in the Security Extensions page. Any editions made in this page will be reflected in the ibm-webservices-ext.xmi file. The web services binding configuration can be specified in the Binding Configuration page. Any editions made in this page will be reflected in the ibm-webservices-bnd.xmi file.

Application Server Toolkit – Web Services Client editor

- Specify client-side WS-Security constraints
  - Security Extensions (ibm-webservicesclient-ext.xmi)
  - Port Bindings (ibm-webservicesclient-bnd.xmi)
- Same editor available in WebSphere Studio V5.1



The client side web services security constraints can be specified using the Application Server Toolkit. Import the application to the tool. If it is a J2EE application client, then expand appClientModule > META-INF. Double click webservicesclient.xml. The web services client security extensions can be edited in the Security Extensions page. Any editions made in this page will be reflected in the ibm-webservicesclient-ext.xmi file. The web services binding configuration can be specified in the Port Binding page. Any editions made in this page will be reflected in the ibm-webservicesclient-bnd.xmi file.

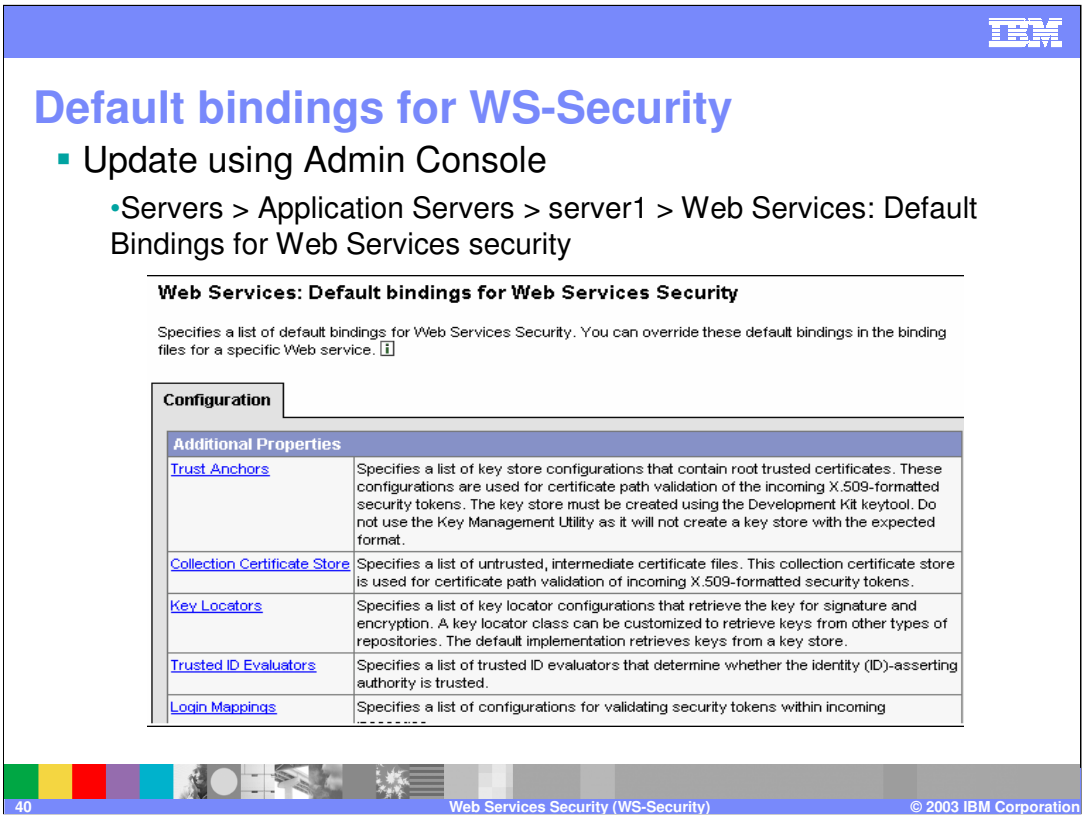
Application WS-Security Bindings – Admin Console

- Admin Console can be used to view and update security bindings in an EJB/Web Module's Additional Properties
  - Server security bindings
  - Client security bindings
    - Only for Clients that reside on the server

The web services server security bindings and web services client security bindings can be updated within the WebSphere administrative console. Expand Applications > Enterprise Applications. Select the Application and then select the EJB/Web Module. The EJB/Web Module's additional properties is shown above. Hence the bindings need not be applied at assembly stage in the Application Server Toolkit or in Web Sphere Studio. The web services security binding configuration can be specified at deployment stage. However, keep in mind that this updating the client security bindings in the Administrative Console applies only to clients that reside on the server. For J2EE application client, the bindings have to be configured in the application server toolkit (or WebSphere Studio Application Developer V5.1).

Default bindings for WS-Security

- Update using Admin Console
  - Servers > Application Servers > server1 > Web Services: Default Bindings for Web Services security



Use this page to manage the default bindings for trust anchors, the collection certificate store, key locators, trusted ID evaluators, and login mappings. The default binding configuration provides a central location where reusable binding information is defined. The application binding file can reference the information contained in the default binding configuration.

It contains:

•Trust Anchors: trusted root certificates for signature verification

The "Trust Anchor Name" is used in the binding file (ibm-webservices-bnd.xmi and ibm-webservicesclient-bnd-xmi when Web services is running as client) to refer to the trust anchor defined in the default binding information.

•Collection Certificate Store: CRLs and non-trusted certificates verification.

The "Certificate Store Name" is used in binding files (ibm-webservices-bnd.xmi and ibm-webservicesclient-bnd-xmi when Web services is running as client) to refer to the certificate store defined in the default binding information.

•Key Locators: locates the keys for digital signature and encryption.

The "Key Locator Name" is used in binding file (ibm-webservices-bnd.xmi and ibm-webservicesclient-bnd-xmi when Web services is running as client) to refer to the key locator defined in the default binding information.

•TrustedID Evaluators: evaluates the trust of the received identity before identity assertion.

The "Trusted ID Evaluator Name" is used in binding file (ibm-webservices-bnd.xmi) to refer to the trusted identity evaluator defined in the default binding information.

Login Mappings: This defines mappings of AuthMethod of security token to JAAS Login Configuration. The mapping is used to authenticate the incoming security token embedded in the Web services security SOAP header of the message. WebSphere Application Server provides definition of BasicAuth (authenticate username and password), Signature (map the subject DN in the certificate to a WebSphere Credential), IDAssertion (map the identity to a WebSphere Credential) and LTPA (authenticate a LTPA token) AuthMethods. After the identity is authenticated then it's Credential is used in the down stream call.

When there is LoginConfig (AuthMethod) defined in the IBM extension deployment descriptor (ibm-webservices-ext.xmi), but no LoginMapping bindings (ibm-webservices-bnd.xmi) defined for the AuthMethod, then the Web services security runtime tries to use the LoginMapping defined in the default binding information.

Sample Configurations

- Sample key stores with WebSphere Application Server V5.0.2
  - Available in {install_root}/etc/ws-security/samples
  - Use for samples and testing, do NOT use in production
- Sample configuration of Trust Anchors, Key Locators, Cert Stores, Trusted ID Evaluators and Login Mappings
  - Available in WebSphere Application Server Default web services security Bindings
    - {install_root}/config/cells/<hostname>/nodes/<nodename>/servers/server1/ws-security.xml
    - Edit in Admin Console

WebSphere application server include some sample key stores and sample configuration of trust anchor, certificate store, key locators, trusted identity evaluators and login mappings.

The sample key stores are:

{USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks (key store password is "client")

{USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks (key store password is "server")

{USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks (key store password is "storepass")

{USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks (key store password is "storepass")

{USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer, the intermediatory "Int CA2".

NOTE: Refer the InfoCenter for more details on these sample key stores. The above sample key stores are for testing and sample purpose only, do not use them in production environment.

Websphere application server provides the following default binding information in ws-security.xml. These can be viewed from the Admin Console as shown in previous foil:

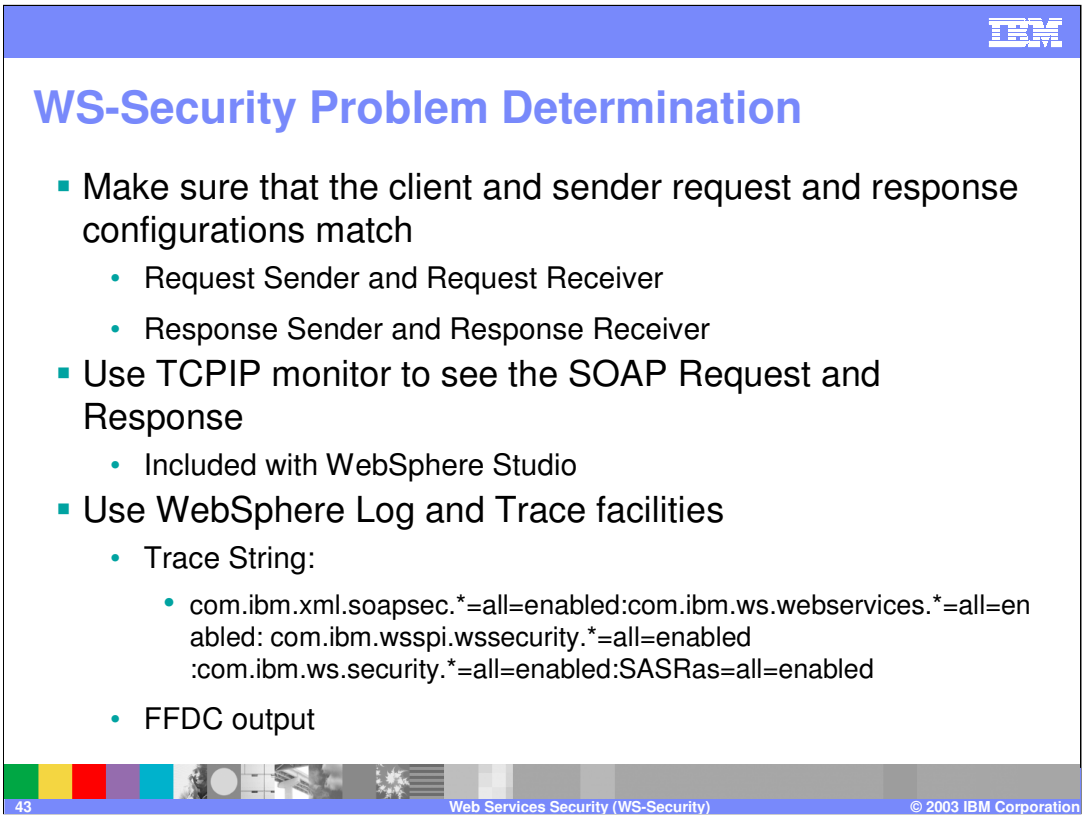
The Trust Anchors SampleClientTrustAnchor and SampleServerTrustAnchor and the Collection Certificate Store SampleCollectionCertStore are available. Also available are the Key Locators :SampleClientSignerKey, SampleServerSignerKey, SampleSenderEncryptionKeyLocator , SampleReceiverEncryptionKeyLocator and SampleResponseSenderEncryptionKeyLocator.

Refer to the InfoCenter for a detailed list of all the default security configurations that are available.

NOTE that the default bindings for Trust Anchors, Collection Certificate Store and Key Locators are for testing or sample purpose only, they should not be used for production. However you can create your own key locators ,trust anchors etc within the Default web services security binding and use it within your application's bindings. You can do this in the Admin Console as shown in previous foil.

Section

Problem Determination



Troubleshooting Web services security is best done by reviewing the configurations in the Application Server Toolkit so that you can match up the client and server request and response configurations in the extensions file. These configurations must match. A client request sender configuration must match a server request receiver configuration. Also, a server response sender configuration must match a client response receiver configuration.

Also, verify that the client security bindings and server security bindings are correctly configured. When the client authentication method is signature, make sure that the server has a login mapping to handle it. When the client uses the public key cn=Bob,o=IBM,c=US to encrypt the body, verify that this Subject is a personal certificate in the server key store so that it can decrypt the body with the private key. You can configure the security bindings using either the Application Server Toolkit or the WebSphere Application Server Administrative Console.

The TCP/IP monitor that is available with WebSphere Studio Application Developer can be used to see the SOAP request and response. For example if Encryption is specified, then this tool can be used to make sure that the SOAP message is encrypted properly.

Check the SystemOut.log file in the ${USER_INSTALL_ROOT}/logs/server1 directory for messages that might provide information about the problem.

Enable trace for Web Services Security by using the following trace specification:
com.ibm.xml.soapsec.*=all=enabled:com.ibm.ws.webservices.*=all=enabled:
com.ibm.wsspi.wssecurity.*=all=enabled:com.ibm.ws.security.*=all=enabled:
SASRas=all=enabled
Type the previous three lines as one continuous line.

Section

Summary

References:

WebSphere Application Server, Version 5.0.2 provides an implementation of the key features of Web services security based on the following specifications:

Specification: Web Services Security (WS-Security) Version 1.0 05 April 2002

(http://www-106.ibm.com/developerworks/webservices/library/ws-secure/)

Web Services Security Addendum 18 August 2002

(http://www-106.ibm.com/developerworks/webservices/library/ws-secureadd.html)

Web Services Security: SOAP Message Security Working Draft 12, Monday, 21 April 2003

(http://www.oasis-open.org/committees/download.php/1686/WSS-SOAPMessageSecurity-12-04021.pdf).

Trademarks and Disclaimers