WS-I Sample Application Blog Series: Securing the WS-I Sample Application

Applies to:

SAP NetWeaver Web Application Server 2004s Java Stack, SPS 7

Summary

As part of the WS-I Sample Application Blog series, this article solely focuses on the security requirements of the new WS-I BSP Sample Application and the utilization of the WS-Security Stack in SAP NetWeaver Web Application Server to implement the appropriate security measures.

Created on: 7 August 2006

Author Bio

As a Standards Architect with SAP's Industry Standards team, Martin works in the area of standardization and interoperability testing of new Web Services technologies, focusing on message security and identity management. Martin is a frequent speaker at conferences and author of books and articles relating to information security, integration middleware and J2EE development.

Table of Contents

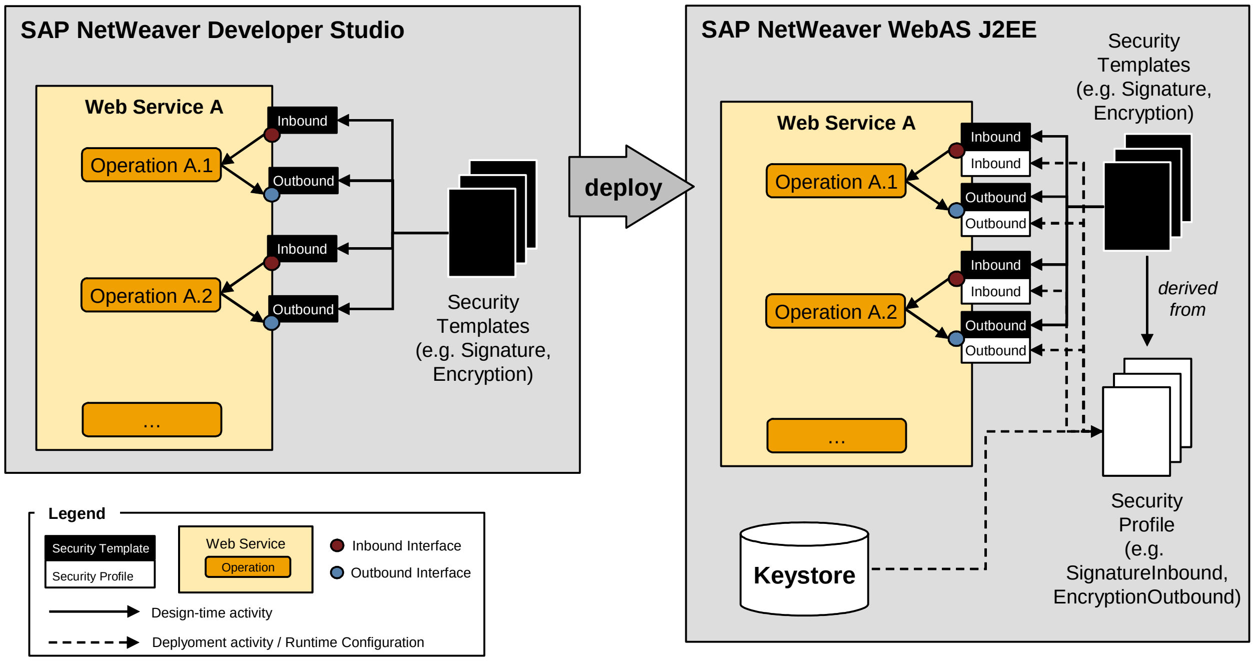
Introduction

To safeguard SOAP messages against threats like tampering, eavesdropping or spoofing, the Web Services Security (WS-Security) standard provides a flexible and extensible framework to implement a wide variety of security scenarios. However, that flexibility and extensibility adds significant complexity to the development process and is potentially a huge hurdle for interoperability.

SAP NetWeaver approaches these issues in different ways. At design time, the developer can select from a list of well designed WS-Security templates in SAP NetWeaver Developer Studio and assign them to the in- and outbound messages of each operation provided by the Web Service. Based on configurations that are commonly found in typical scenarios, the templates describe from a high-level point of view what will be protected in the SOAP message and summarize these options under a self-explanatory name (e.g. ‘Signature’ or ‘Username + Encryption’). This simplifies the development process and does not require the programmer to be an expert in any detailed security technologies such as cryptographic algorithms and secure hash functions. For instance, the security template ‘Signature’ adds a timestamp to the message and signs it together with the SOAP body. This ensures that the message can be verified by the recipient and prevents replay attacks.

For each of the WS-Security templates assigned to the service operations at design time, a so called security profile with runtime configuration settings, such as a valid username and password or an X.509 certificate, is required. These security profiles are created and assigned to the Web Service operations in the Visual Administrator tool.

The figure below depicts this programming model for the design- and runtime components in SAP NetWeaver.Configuration of the inbound or outbound messages of an operation have different meanings depending on whether they are applied to a service consumer or provider. During the course of security configuration one has to be aware of the following concepts: For a consumer, an inbound message refers to a SOAP response, whereas an outbound message refers to a SOAP request. For a provider, it is just the other way round.



Security Requirements of the BSP Sample Application

Within the security tab of the Web Service Configuration in NetWeaver Developer Studio, the developer can choose between four pre-configured WS-Security templates for in- and outbound message processing. Although these templates cover typical scenarios found in many deployments, the security requirements based on the analysis conducted in the Supply Chain Management (SCM) Security Architecture Document (http://www.ws-i.org/SampleApplications/SupplyChainManagement/2006-04/SCMSecurityArchitectureWGD5.00.doc) are slightly different. Therefore, the installation archive of SAP's new Sample Application implementation (http://www.ws-i.org/SampleApplications/BSP/SAP-BSP-SA-2006.05.zip) adds additional templates to the runtime environment of the J2EE engine.

To illustrate this in more detail, we'll first have a closer look at the SCM Security Architecture document from WS-I. The main part of this document describes the security requirements for all entities defined by the initial SCM Architecture document (http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-12/SCMArchitecture1.01.pdf) like the Retailer or the Warehouse services on a per-operation basis, further broken down into the request and response message. Table 1 on page 17 in this document provides a summary of these requirements, categorized by integrity, authentication and confidentiality. The getCatalog request for example must be protected by these WS-Security mechanisms:

- Authentication: The Web Client authenticates itself with a WS-Security UsernameToken and adds an XML Signature to the message using its private key
- Integrity: To protect the message against tampering, the SOAP Body, UsernameToken and Timestamp elements are signed by the Web Client
- Confidentiality: The Web Client uses the Retailer's public key certificate to encrypt the SOAP Body and the signature in the WS-Security header

The security analysis in the SCM Security Architecture document follows the recommendations of the Basic Security Profile (BSP, http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html). One such advice from the BSP is that signatures in the WS-Security Header should be encrypted.

In general, a WS-Security template is an XML document that consists of two parts: The main part security expressions (assertions). These assertions define which mechanisms available from WS-Security must be applied to a message. In addition, a configuration section defines some placeholder variables (not shown in the example below) used to configure the assertions in the template with the Visual Administrator tool. The intent of this declarative approach is that it allows flexibility in terms of the tokens, cryptography, and mechanisms used, and that it separates any security aspects from the code. An excerpt from the “GetCatalogRequestOutbound” security template for the getCatalog request of the Web Client is illustrated below:

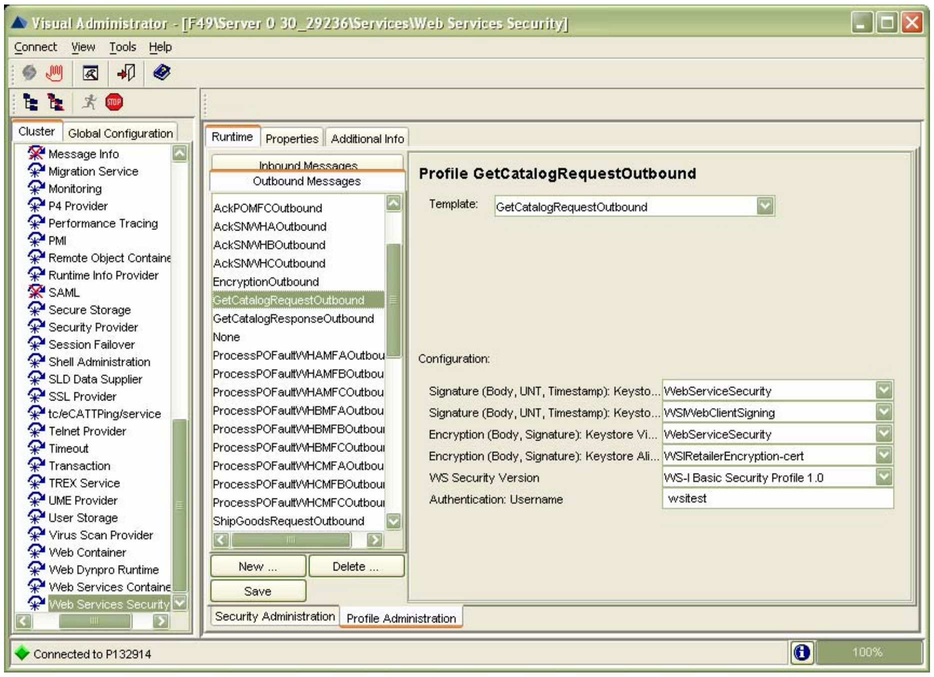
```
<wsp:SecurityHeader MustUnderstand="1"/>
<sap:WSSecurityVersion xmlns:sap="http://sap.com/wssec"
             Version="{4}"/>
<wsp:MessageAge TargetID="timestamp" Age="3600"/>
<sap:Username xmlns:sap="http://sap.com/wssec" Role="soap:finalActor"
             ContainsNonce="false" ContainsCreated="true"
             TargetID="username">
  <sap:Name>{5}</sap:Name>
</sap:Username>
<wsp:Integrity BinarySecurityTokenType="wsse:X509v3">
  <wsp:SecurityToken>
    <wsp:TokenType>wsse:X509v3</wsp:TokenType>
    <sap:KeystoreKey View="{0}" Alias="{1}"/>
  </wsp:SecurityToken>
  <wsp:MessageParts Dialect="http://security.sap.com/ID">
    <wsp:Algorithm Type="wsse:AlgCanonicalization"
             URI="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <wsp:URI>#body</wsp:URI>
  </wsp:MessageParts>
  <wsp:MessageParts Dialect="http://security.sap.com/ID">
    <wsp:Algorithm Type="wsse:AlgCanonicalization"
             URI="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <wsp:URI>#timestamp</wsp:URI>
  </wsp:MessageParts>
  <wsp:MessageParts Dialect="http://security.sap.com/ID">
    <wsp:Algorithm Type="wsse:AlgCanonicalization"
             URI="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <wsp:URI>#username</wsp:URI>
  </wsp:MessageParts>
</wsp:Integrity>
<wsp:Confidentiality>
  <wsp:EncryptionType>http://www.w3.org/2001/04/xmlenc#Content
             </wsp:EncryptionType>
  <wsp:Algorithm Type="wsse:AlgEncryption"
             URI="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
  <wsp:SecurityToken>
    <wsp:TokenType>wsse:X509v3</wsp:TokenType>
    <sap:KeystoreCertificate View="{2}" Alias="{3}"/>
```

```
31     </wsp:SecurityToken>
32     <wsp:MessageParts Dialect="http://security.sap.com/ID">
33       <wsp:URI>#body</wsp:URI>
34     </wsp:MessageParts>
35     <wsp:MessageParts Dialect="http://security.sap.com/ID">
36       <wsp:URI>#sig-0</wsp:URI>
37     </wsp:MessageParts>
38   </wsp:Confidentiality>
```

Here is a quick summary: Lines 1 and 2 cause the Web Services runtime to use WS-Security (that is to include a WS-Security header in the message). Lines 4 to 6 take care of the UsernameToken that must only include the username. All integrity related settings are configured in lines 7 to 24: The security token type used by the receiver to verify the XML Signature (X.509v3), the keystore alias of the key to sign the data (line 10), and the parts of the message that will be signed (12-15 SOAP Body, 16-19 Timestamp, 20-23 UsernameToken). The last section of the template (lines 25-28) fulfills the confidentiality requirements of the getCatalog operation: An X.509 Public Key Certificate (line 29) is used to encrypt the SOAP Body (lines 32 to 34) and the Signature in the WS-Security header (lines 35-38).

The Integrity- and Confidentiality-assertions in a WS-Security template use the <wsp:URI> element with a unique identifier to reference the message parts that will be signed or encrypted. The Web Service stack therefore has to add an Id attribute with this unique value to the respective elements (e.g. Id="sig-0" on the <ds:Signature> element in the WS-Security header or Id="body" to the <SOAP-ENV:Body> element).

In Visual Administrator, the security profile assigned to the GetCatalogRequestOutbound template and used to configure its runtime settings can be found under the same name in the Profile Administration for outbound messages in the Web Services Security Service:



Message Info
Migration Service
Monitoring
P4 Provider
Performance Tracing
PMI
Remote Object Containe
Runtime Info Provider
SAML
Secure Storage
Security Provider
Session Failover
Shell Administration
SLD Data Supplier
SSL Provider
tc/eCATTPing/service
Telnet Provider
Timeout
Transaction
TREX Service
UME Provider
User Storage
Virus Scan Provider
Web Container
Web Dynpro Runtime
Web Services Containe
Web Services Security

AckPOMFCOutbound
AckSNWHAOutbound
AckSNWHBOutbound
AckSNWHCOutbound
EncryptionOutbound
GetCatalogRequestOutbound
GetCatalogResponseOutbound
None
ProcessPOFaultWHAMFAOutbou
ProcessPOFaultWHAMFBOutbou
ProcessPOFaultWHAMFCOutbou
ProcessPOFaultWHBMFAOutbou
ProcessPOFaultWHBMFBOutbou
ProcessPOFaultWHBMFCOutbou
ProcessPOFaultWHCMFAOutbou
ProcessPOFaultWHCMFBOutbou
ProcessPOFaultWHCMFCOutbou
ShipGoodsRequestOutbound

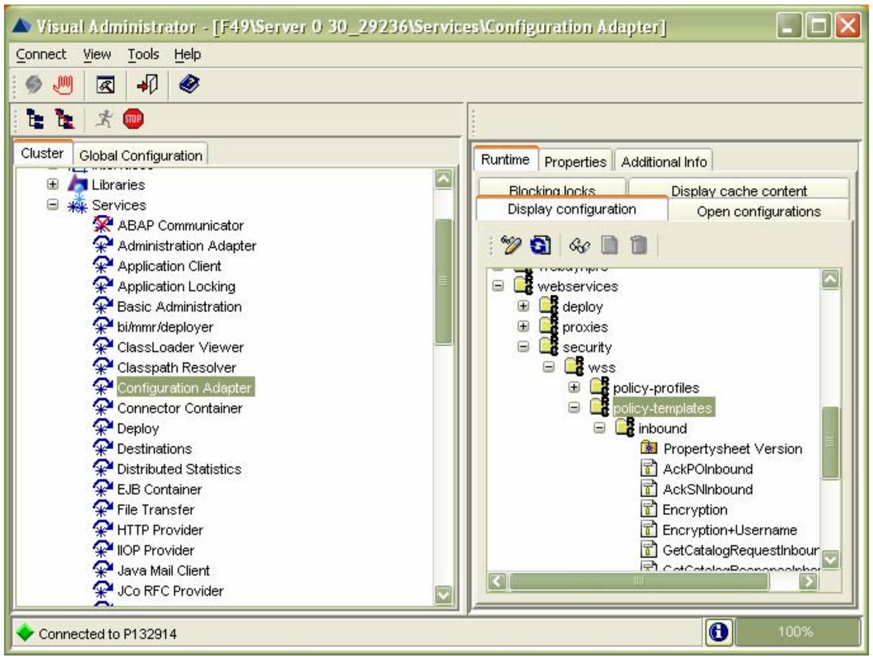
Profile GetCatalogRequestOutbound

Template: GetCatalogRequestOutbound

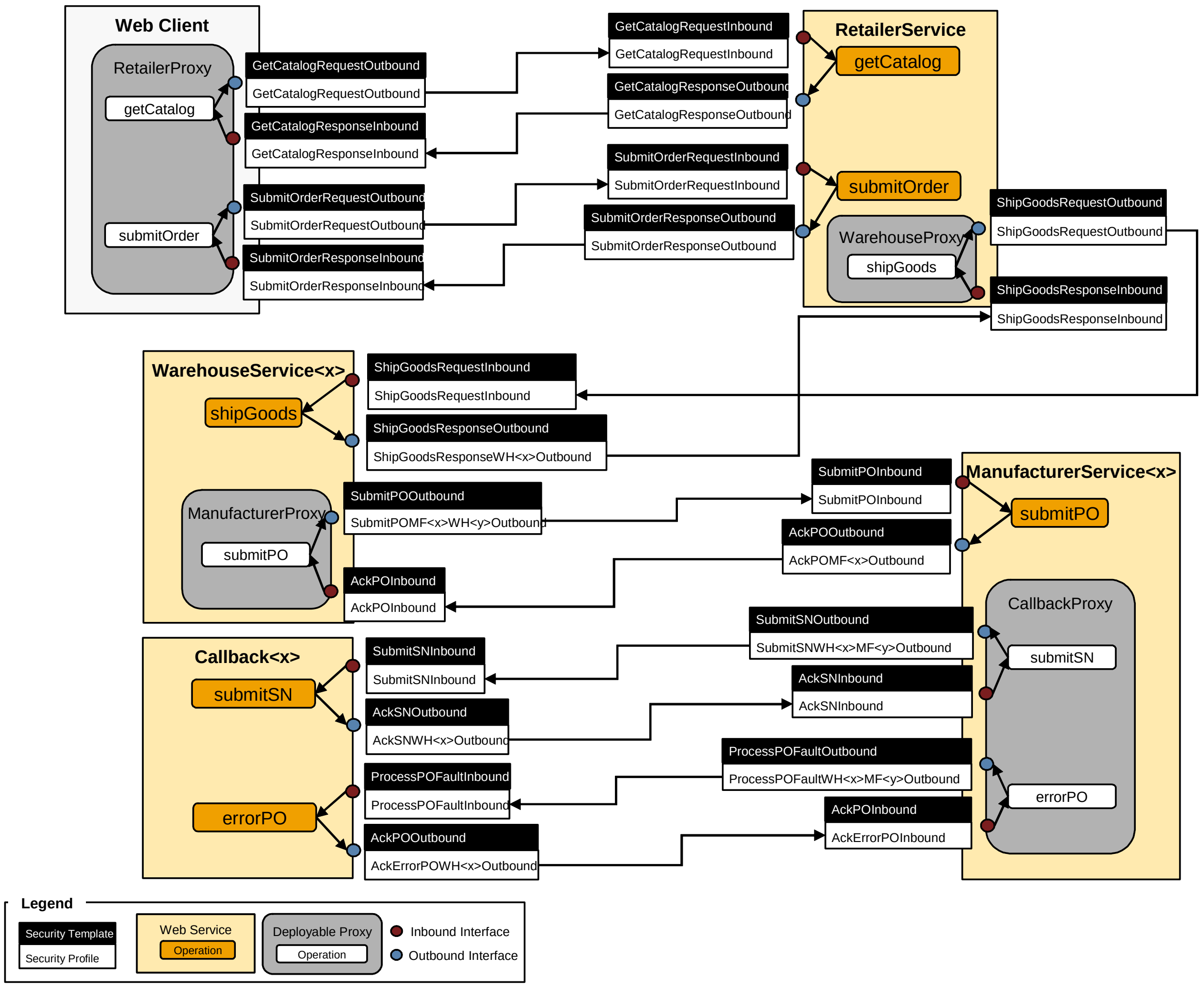
Configuration:

Signature (Body, UNT, Timestamp): Keysto...	WebServiceSecurity
Signature (Body, UNT, Timestamp): Keysto...	WSIWebClientSigning
Encryption (Body, Signature): Keystore Vi...	WebServiceSecurity
Encryption (Body, Signature): Keystore Ali...	WSIRetailerEncryption-cert
WS Security Version	WS-I Basic Security Profile 1.0
Authentication: Username	wsitest

Even though there are one-to-one relationships between profiles and templates (e.g. for getCatalog operation) in the WS-I Sample Application, the intension is to maximize the reuse of templates, e.g. for services with the same operations but different security runtime configurations. As an example, each Warehouse Service (A, B and C) has the same integrity requirements defined for the shipGoods operation, but each Warehouse instance uses its own key pair for signing the (outbound) response message. Hence, three different outbound security profiles (ShipGoodsResponseWH<A/B/C>Outbound>) are assigned to the Warehouse Services outbound interfaces, but each of these profiles is based on the same WS-Security template (ShipGoodsResponseOutbound).



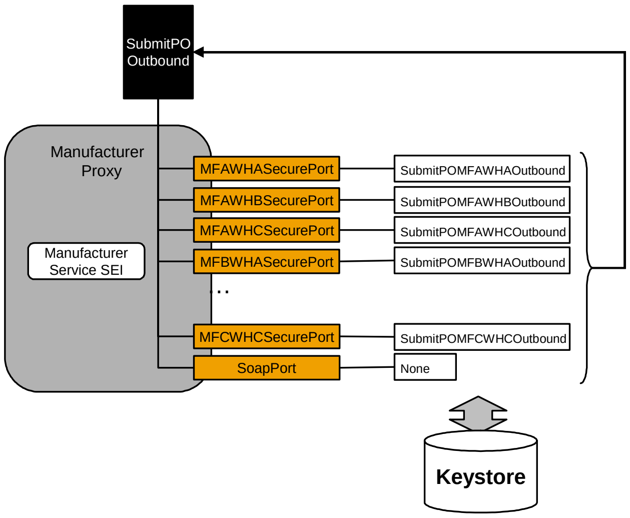
All in- and outbound security templates deployed with the WS-I Sample Application can be found under the Configuration Adapter node webservices/security/wss/policy-templates. The following figure depicts the detailed template and profile configuration of all WS-I Sample Application Web Service consumers (client proxies) and providers:



Secure client-side Web Service invocation

In order to utilize WS-Security on the client-side, SAP's WS-I Sample Application implementation uses Deployable Proxies to call secured services provided by SAP's or any other Sample Application instance. A Deployable Proxy always consists of a Service Endpoint Interface (SEI) which exposes the business methods provided by a particular Web Service. Any configuration of an SEI regarding its security features is done through the definition of one or more logical port(s). They are always associated with an SEI and each logical port, unambiguously identified by a proxy-wide unique name, can be assigned to a separate security profile in Visual Administrator after deployment. This enables the reuse of the same client proxy with different security configurations.

As an example, the Manufacturer Proxy defines 9 logical ports (MF<A/B/C>WH<A/B/C>SecurePort) to invoke the Manufacturer Services A, B and C's submitPO operation from the Warehouse shipGoods operation:



Each logical port is assigned to a different security profile (SubmitPOMF<A/B/C>WH<A/B/C>Outbound) which are all based on the same security template (SubmitPOOutbound). This security template defines the general security mechanisms based on the SCM Security Architecture that must be applied to the submitPO request. The profiles configure this template with regard to the keys used for signing the request (private signature keys of the Warehouses A, B and C) and the keys used to encrypt the data for the recipient (public encryption keys of the Manufacturers A, B, and C). This results in a 3-by-3 matrix and sums up to 9 different profiles.

Which port to choose for the secured submitPO invocation by the Manufacturer proxy can only be determined at runtime. Based on product numbers and ordered quantities from the Retailer's catalog, each of the Warehouse services could submit the purchase order to replenish their stocks from the Manufacturers. To obtain a specific logical port (and therewith its associated security profile configuration) at runtime, the following code is used in the orderFromManufacturer method in the Warehouse EJBs:

```
String portName = new String("MF" + manufacturerId.toUpperCase() + "WH" +
       getWarehouseName().toUpperCase() + "SecurePort");

manufacturer = (ManufacturerServiceAViDocument)
       mfsa.getLogicalPort(portName);
```

Dynamically switch between BP and BSP

In addition, all deployable proxies have one logical port with the name "SoapPort" which is not assigned to any of the security templates and profiles. This port is used for unsecured service invocations according to the WS-I Basic Profile (BP, http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile) 1.0 which promotes interoperability clarifications and amendments for the core set of Web services specifications, such as SOAP, WSDL and UDDI.

The development team at SAP decided to implement an extensible architecture for the new SAP Sample Application that allows adding new WS-I profiles as they evolve over time. In the current download, BP 1.0 and BSP 1.0 are supported. After successful login, the user simply has to choose the profile type from the drop down box in the lower section of the configuration page. According on this selection, the code picks the "SoapPort" or one of the secure ports for each client proxy call at runtime by applying the same programming model as illustrated above.

Conclusion

This blog covered everything you need to know about security and the BSP-support in the new SAP Sample Application implementation. Please note that some of the features used in this context (e.g. encryption of the Signature in the WS-Security header) are only possible due to an unofficial patch that is part of the installation archive. It is highly recommended to use this patch only on a test system and never in production.

Although the concepts of security templates and profiles might sound a little bit confusing for developers that are used to read API docs and study code samples, there are important advantages of this approach:

- The task of configuring the security of a service is broken down into a selection between a few high-level scenarios that reduce the need for many lower-level decisions
- Service architects and developers do not have to be security experts
- It provides a clear separation of concerns by decomposing non-functional security requirements from the business logic of the application into well-separated modules

SAP NetWeaver's programming model actually follows a common trend in Web Services standards and development methodologies as there are many initiatives in the industry that promote the definition and use of metadata description languages and domain-specific assertions for allowing Web Services to express their constraints and requirements, such as WS-Policy or WS-SecurityPolicy.

What's next?

In the next installment of this blog series we'll make a sample walkthrough using the Web Dynpro user interface and do some interoperability testing against another vendor's endpoints. We'll also take a closer look at the advanced debugging and logging features of the new Sample Application.

Disclaimer and Liability Notice