Java CAPS, OpenESB, GlassFishESB

Providing Policy-driven Web Services Security support using an XML Security Gateway

Michael.Czapski@sun.com

February 2009

Contents

1. Abstract

Securing web services, to be invoked over the Internet, is both essential and difficult. Using appropriate tools and technologies makes it easier to accomplish the task. Developer-dependent solution, where security is embedded directly into consumers and providers, is inflexible and labour-intensive. Gateway-based solutions are more flexible, more dynamic and easier to manage. In this note Java CAPS 6-based web service consumer and provider pair are developed. The solutions are exercised first without, then with the web services security gateway. This enables demonstration of how web services can be secured, how policies can be developed and propagated and

how WS-Security-mandated XML markup can be dealt with outside the development shop. The Layer 7 SecureSpan XML Gateway, and its oft forgotten companion, the SecureSpan VPN Client, are used to explore the topic. The reader should be able to acquire enough knowledge to obtain and deploy the SecureSpan XML Gateway, and to use its basic functionality to implement gateway-mediated secure web services solutions.

2. Introduction

Securing web services, to be invoked over the Internet, is both essential and difficult. Using appropriate tools and technologies makes it easier to secure web services as certain development environments, like NetBeans, provide checkbox-style interface for enabling web services security for certain kinds of web services. Other kinds of web service, for example JBI-based or Java CAPS 6 Repository-based web services, are more difficult to secure because the tooling support is less developed or less feature-rich. Securing web services at build time, as implied above, has certain advantages and certain disadvantages. Most notable disadvantages are the need to re-build and re-deploy each service if the security policy changes, and the reliance on developers to implement and maintain security policies in a fragmented and variable manner. Removing the need for developers of services to deal with service security is one way to address the major disadvantages of developer-dependent security policy implementation. By exploiting the properties of the HTTP protocol, Web Services security and other non-functional requirements can be abstracted to a Proxy component. This component, a Security Gateway, intercepts service requests, applies or parses security information, decorates or parses service responses, enforces SLAs, maintains policies and policy versions, implements firewall functionality and provides a variety of other services.

In this Note I am discussing, with examples, a Gateway-based secure web services solution for Sun Java CAPS-based services. Both the Web Service Provider and the Web Service Consumer are built. The invoker consumes the service provided by the provider. Both are built and deployed as plain, unsecured web service components. The Gateway and the VPN Client are added as intermediaries and various security policies are developed, deployed and illustrated by inspection of message exchange between the consumer and the provider.

I am using Java CAPS 6 Repository-based web services because these have fewer Web Services Security features supported out of the box. Provision of web services security for Repository-based web services is harder; therefore the benefits of using a webs services security gateway are much more clear cut. EJB-based and JBI-based web services can take advantage of the Metro stack directly so the benefits of using a security appliance are less obvious but still significant. This will be discussed further in sections dealing with the SecureSpan XML Gateway.

The Layer 7 SecureSpan XML Gateway is the gateway I picked to develop and deploy secure web services solutions discussed in this Note. Sections 15.1, “Obtain the Layer 7SecureSpan XML Gateway”, 15.2, “Configure the SecureSpan XML Gateway” and 15.3, “Install and Configure the SecureSpan VPN Client”, deal with obtaining, installing and initially configuring the Gateway. The gateway will be used and certain of its features exercised in examples which will be discussed in various

sections in this Note. It may be appropriate to skip over and work through sections 15.1, 15.2 and 15.3, before continuing.

Note that I am not associated with Layer 7 Technologies, I am not writing this Note on their behalf, and I am not a Layer 7 SecureSpan Gateway expert. I spent some time with the documentation and the product. What I have seen convinced me that it may be worthwhile to write about some of the possibilities and some of the obvious advantages of deploying a security gateway to protect web services. Some of the secure web services solutions that are called for nowadays are much easier to implement, and much easier to manage, when using a web security gateway. There are other vendor's gateway products on the market and likely offer these and other services that may be of invaluable for an enterprise. Layer 7 SecureSpan Gateway is the one I had access to. From my perspective any other gateway product should provide at least the same kind of flexibility and benefits that the SecureSpan XML Gateway does.

In the Note I will discuss web services security topics to set the context, discuss solution to be developed, walk through creation of the service provider and service consumer, configuration of the gateway and VPN Client infrastructure and a series of iterations of policy creation and testing.

3. Securing Web Services - Discussion

In Java CAPS 6 Update 1 one can construct at least three kinds of web services – EJB-based web services, JBI-based web services and Repository-based web services. Depending on how the services are designed, and in what environment the services execute, they can be considered "secure" or not "secure". In some circumstance service security is critical to its viability; in others it is a luxury. Providing the appropriate level of "security" to a service may be easy or difficult, depending on a number of factors. Not the least of these factors is the designer's knowledge of the web services security standards, and the means available in the tooling to assist in implementing standard security mechanisms.

Why would we care if the service invocation is "secure"? That would depend on what the consequences of the message exchange being seen or altered by third parties might be. The likelihood that third parties can see or alter message exchange is also an important factor. The more likely an authorised activity is, and the more severe the consequences of such an activity, the more important it is to secure the message exchange, and the greater the complexity of security measures that need to be taken.

Let's assume that the service is to be invoked over the Internet and that the consequences of message exchange being seen or altered are severe.

Let's discuss what is meant by security in the context of web services, discuss security characteristics of various kinds of Java CAPS-based web services and introduce the concept of a security gateway.

One can "secure" a web service by requiring the use of WS-Security Standard-supported XML Digital Signatures, XML Encryption, Username Token, Timestamp Token, SAML Token, etc., individually or in combination. By requiring and using different combinations of security tokens one obtains varying degrees of "security".

Digital Signatures, more specifically XML Digital Signatures used in securing SOAP messages, are used to ensure integrity of messages on the wire, that is, facilitate detection of message alteration in transit. They are also used to convey authenticity of messages. The entire message or selected parts of the message can be digitally signed. The reason one would digitally sign parts of the message rather then the entire message is typically cost. If only certain parts of the message must be protected from tampering and must be guaranteed to be authentic then signing selected parts of the message will minimise the high resource consumption typically associated with cryptographic operations.

Conveyance of authenticity relies on the properties and use patterns of key pairs in public key cryptography. Both keys of the pair are generated at the same time. The two keys are related in such a way that one cannot be derived from the other and that plaintext encrypted with one can only be decrypted with the other. One of the keys, the private key, is kept confidential by the party that owns the key pair. The other key, the public key, embedded in a "certificate" which guarantees its authenticity and integrity, is distributed to any party with whom secure communication will be undertaken. Public Key Infrastructure (PKI) is the means of guaranteeing public key authenticity and integrity through issuance and revocation of "certificates', and possibly distribution of public keys (certificates). There is a great deal more to all this but for the purpose of this discussion it is enough to say that if the owner of the private key encrypts some plaintext and sends it to the recipient, the recipient will be able to decrypt it only with sender's public key, to which he/she has ready access. Because of the properties of the key pair the recipient knows that only the "other" key of the pair could have been used to encrypt the plaintext that he/she just decrypted. Because the "other" key, the private key, is supposed to be kept secret by its owner the recipient assumes that only the owner of the private key could possibly have encrypted the plaintext. This guarantees message authenticity.

Encryption, and specifically XML Encryption used in securing SOAP messages, can be used to ensure message integrity and protect confidentiality of information on the wire. Either the entire message or selected parts of the message can be encrypted for the same reasons that an entire message or selected parts of a message would be digitally signed. If the encrypted parts of the message are tampered with, decryption will fail and the recipient will conclude that the message was tampered with. The confidentiality of the message is guaranteed in a way similar to authenticity guarantee when using digital signatures. By encrypting the message with a public key of the recipient, the sender of the message ensures that only the recipient, the holder of the private key of the key pair, can possibly decrypt the message. This allows anyone to send a confidential message to the owner of the private key regardless of where there was a prior communication between them.

Using a Timestamp Token allows the infrastructure to reject messages that are "too old", combating "message reply" attacks, but only if it can be guaranteed that the timestamp itself cannot be modified without detection. By itself the Timestamp Token is pretty useless as a security device. In combination with Encryption or Digital Signature its integrity can be guaranteed and it can be used for message reply detection.

The Username Token, whether with a plaintext password or a digest password, can be used to provide user credentials for authentication. Much as the Timestamp Token, Username Token cannot be trusted unless it is protected from eavesdropping. XML Encryption must be added to the mix to ensure that the Username Token can not be intercepted and subsequently used for rouge access to resources that require authentication.

Whether to use specific WS-Security tokens, what combination of tokens, what token attribute values are appropriate, etc., is typically subject to organisation's security policies. Configuring a web service consumer or a web service provider to use/support/require specific security tokens is the application of security policies.

Web Services security policy-mandated tokens are conveyed as part of the SOAP message using the SOAP Header extension mechanism. Different SOAP Header components convey different security policy tokens. I will not discuss this in details. It will suffice to say that whatever means we use to apply security to SOAP messages, these means will add/modify SOA Headers, in many cases very extensively.

When developing EJB-based web services one can use the NeBeans IDE facilities to specify security policies to use for a particular web service. An example of this, Figure 3-1, shows specification of the Username Token using the NetBeans checkboxes and dropdown menus.

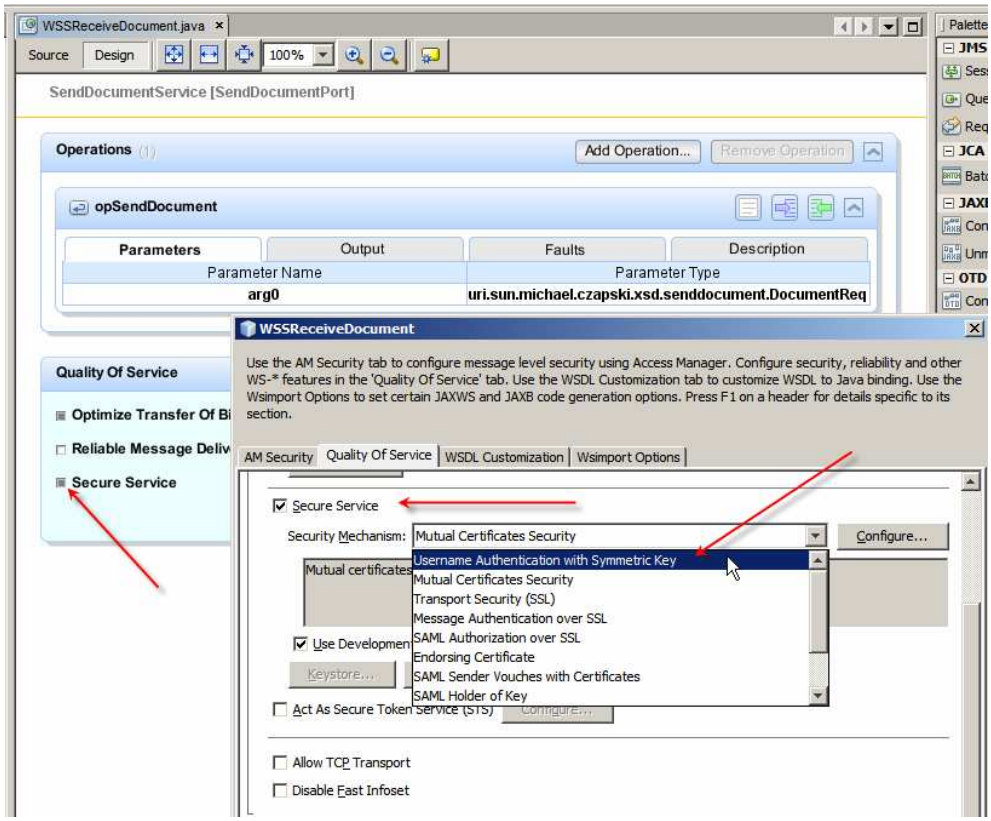

Figure 3-1 Adding Service Security – Username Token

Figure 3-2 shows the dialogue box used to specify the parts of the request message to which to apply encryption and/or which to digitally sign.

It is quite easy to specify security policies for EJB-based web services using the NetBeans IDE but it must be borne in mind that this is a design-time specification. To apply this security policy, or any changes to this security policy, it is necessary to build and deploy the service. To see what security policy is to be applied to this service one must have access to the development environment to look at the configuration files. The onus is on the web service developer to apply appropriate security policies and to maintain them as organisational policies change.

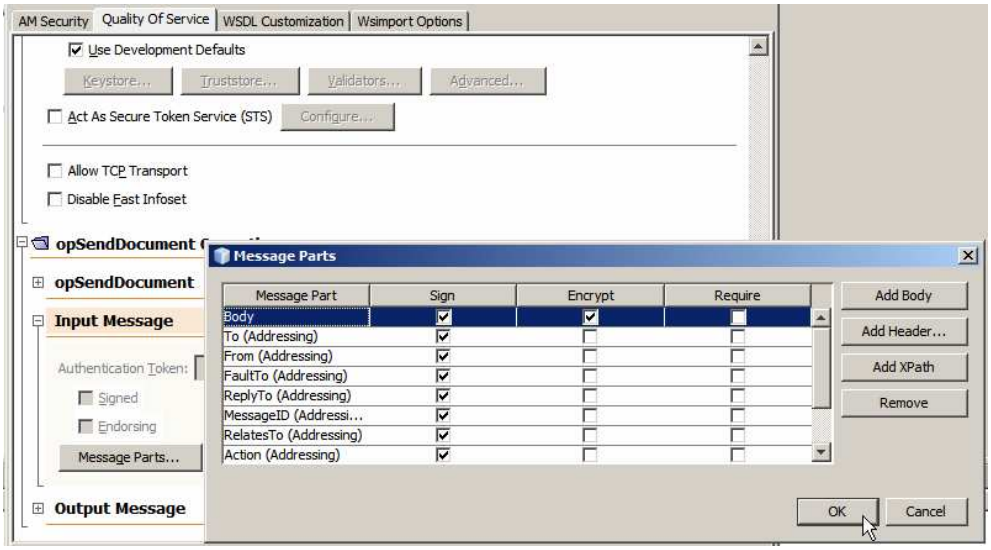

Figure 3-2 Specifying message parts to sign and/or encrypt

For JBI-based web services a subset of web services security policies can be specified using graphical means, another subset requires manual modification of the service WSDL and others are not supported at this time. As is the case with applying web services security policies to EJB-based web services, applying web services security policies to the JBI-based webs services is a design-time activity. The service, or its configuration, must be modified, then the service assembly must be built and deployed for the policy change to take effect. Similarly, access to development environment and development knowledge are required to determine what policy is used for which service. The onus is again on the web service developer to apply appropriate security policies and to maintain them as organisational policies change.

Repository-based web services are the hardest to secure programmatically. They offer the fewest graphical configuration options and require development of SOAP handlers for anything not provided through the graphical interface.

Java CAPS 5.x and Java CAPS 6 Repository-based web services use the JAX-RPC technology with support for WS-Security 1.0 (2004). For Java CAPS 5.x there is no way to directly provide JAX-WS-based WS-Security 1.1 support using SOAP handlers since JAX-WS requires the GlassFish Application Server runtime infrastructure and Java CAPS 5.1 applications are not supported on GlassFish. Java CAPS 6 Repository-based web services are deployed to the GlassFish Application Server so JAX-WS-based WS-Security can be provided through the SOAP handlers.

The EJB-based web services wrappers can be used to secure Repository-based web services, much the same way as providing MTOM support for Repository-based services, which I discussed in my Blog in entries “Java CAPS 6 Update 1 - Invoking

MTOM Web Service using Java CAPS Classic Web Service Client" - http://blogs.sun.com/javacapsfieldtech/entry/java_caps_6_update_1 and "Java CAPS - Exposing MTOM-capable Java CAPS Classic Web Service" - http://blogs.sun.com/javacapsfieldtech/entry/java_caps_exposing_mtom_capable. Again, access to development environment and development knowledge are required to determine what policy is used for which service. The onus is again on the web service developer to apply appropriate security policies and to maintain them as organisational policies change.

All of this requires a developer to add security code to the services at build time, however little or much the IDE helps. It requires the developer to have some knowledge of security. It requires services to be modified, re-built and re-deployed for any policy changes to take effect. It does not offer central audit and control unless such is custom built and programmatically invoked from each service. Specific development activities must be undertaken to support service lifecycle management like versioning or retirement. Specific infrastructure must exist to support differential service levels, service virtualisation, protocol conversion or security zone traversal.

For few services, the developer-dependent security policy application may be acceptable. For many services, it becomes very hard to manage.

The reason for this Note is to discuss a Gateway-based approach to securing web services. This approach addresses all the issue that have been mentioned so far and, while it adds a couple of issues not present in the developer-dependent approach, it is superior in my opinion for enterprises with more then a few services that need securing.

The basic idea is that a developer develops a "plain" web service, that is one where no security policies are embedded at build time, and this service is deployed to the local infrastructure. A "web services security gateway", which understand web services security, performs auditing, authentication, service versioning, XML transformation, and whatever other activities can be profitably deployed to a gateway, is put in front of the service to mediates access to it. This can be done because web services are merely SOAP (XML) messages traversing a channel that is HTTP-based and HTTP facilitates the use of proxy and forwarding infrastructure. A service consumer sends the secure request to the Gateway. The Gateway invokes the plain service, gets the response and passes it to the consumer. Neither the consumer nor the provider is aware that it is dealing with the other through an intermediary. The intermediary, the gateway, consults the security policy defined for each service and processes requests and responses as might be required. For inbound requests, the Gateway verifies that the request conforms to the security policy, for example is encrypted, is signed, has Username Token, etc., decrypts the request (if encrypted), verifies digital signature (if signed), validates credentials against its own or 3rd party Authentication infrastructure (if credentials are provided), validates Timestamp (if present), strips all security-related headers and forwards the plain request to the actual service that implements the business functionality. When the service response reaches the Gateway it adds security headers as required, for example adds a Timestamp, and digitally signs and encrypts the response if required by the policy. It then returns the response to the original invoker.

The developer of the service does not have to have any knowledge of webs services security as he/she does not have to deal with it. Security policies can be handed over to security people to define and enforce since they are the logical people to own the security gateway. Being externalised, security policies can be easily changed and can be dynamically applied without any changes to the service implementation or the need to build or deploy the service. With the right kind of a gateway other facilities, such as auditing, runtime governance and SLA definition and enforcement, are available. The right kind of Gateway will be standards-compliant and will be maintained such that it is always current as technology and standards evolve.

It is worth remembering that two parties are involved in a web service-based message exchange. Both parties must agree on the security polices to use, and their security implementations must be standards-compliant and compatible. As a piece of infrastructure separate from the service delivery infrastructure, the gateway has a monetary and logistical cost associated with it, not just for the enterprise that deploys it but also for the parties who are intended to use the services. When exposing secure services for consumptions by multiple third parties, an organisation effectively forces the would-be consumers to make an investment in their own web services security-capable infrastructure. They must either deploy a security gateway, or add security requirement to the client development projects, or find some other way to deal with invocation of secure web service. For small target organisations this can be a considerable obstacle so the enterprises embarking on the journey must consider the flow-on effects on others of securing their own services. The right kind of a gateway solution may have a small-footprint proxy component, or an API library, which will be able to be distributed to the would-be service invokers to facilitate their use of secure web services to minimise the cost and the complexity of the task.

Typically one would deploy a Gateway in one's enterprise to secure web services being exposed to others, and to secure invocation by our clients of other parties' services that require securing. One assumes that these other parties have their own web services security infrastructure that takes care of the other end of the service invocation/provision. It is implied that the invoking party somehow "knows" the security policy to use for our secure service and that we somehow "know" the security policy for the other party's service which we need to invoke.

Imagine we have to change a security policy applicable to a particular service. With the gateway we will be working with it is easy for us, as will see later. Whether it is equally easy for the service consumers depends to a huge extent on what technology they use for web service security. If they use the developer-dependent method, discussed earlier, they are in for more development work every time we need to change the policy. If there are many consumers the logistics of changing policy may become quite complex and may take a considerable amount of time. If the security policy change is a consequence of a security vulnerability in the service protected by the original policy, we may have to knowingly run vulnerable services in order to allow our service consumers to upgrade their capabilities to accommodate the new policy we wish to introduce.

Let's say that we are a dominant player, with the resources to procure and deploy a web services security gateway, and we expose secure web services for consumption by other parties. Let's further say that the would-be consumers of our secure web

services don't have the size and resources that would justify procurement of a fully-fledged webs services security gateway of their own. If we are a good citizen with resources to spare, for example a government department or a major supplier, we could offer the consumers of our service access to a VPN Client proxy or an API, on some basis. This would make it easy for even the smallest consumer to use our services, however secure.

4. Solution Schematics

In this Note we will not discuss the topic of securing Java CAPS web services which does not involve a Gateway. This is because this Note is about the use of the Gateway for the purpose. Securing EJB-based web services has been discussed extensively and a fair bit of material on the topic is publicly available. Securing JBI-based web services has been discussed to a much lesser extent, consistent with the age of the JBI technology, but material is publicly available trough the OpenESB Project site, open-esb.dev.java.net. I have some material on securing Repository-based web services using JAX-RPC in my Blog, in entry entitled "Java CAPS 5.1, Implementing WS-Security 1.0 (2004) with JAX-RPC" at http://blogs.sun.com/javacapsfieldtech/entry/java_caps_5_1_implementing.

To give us a good idea of what is supposed to happen the solution schematic diagrams show the components of the solution and the stages in which they will be developed, installed, configured and exercised. The ultimate solution is shown in Figure 4-1.

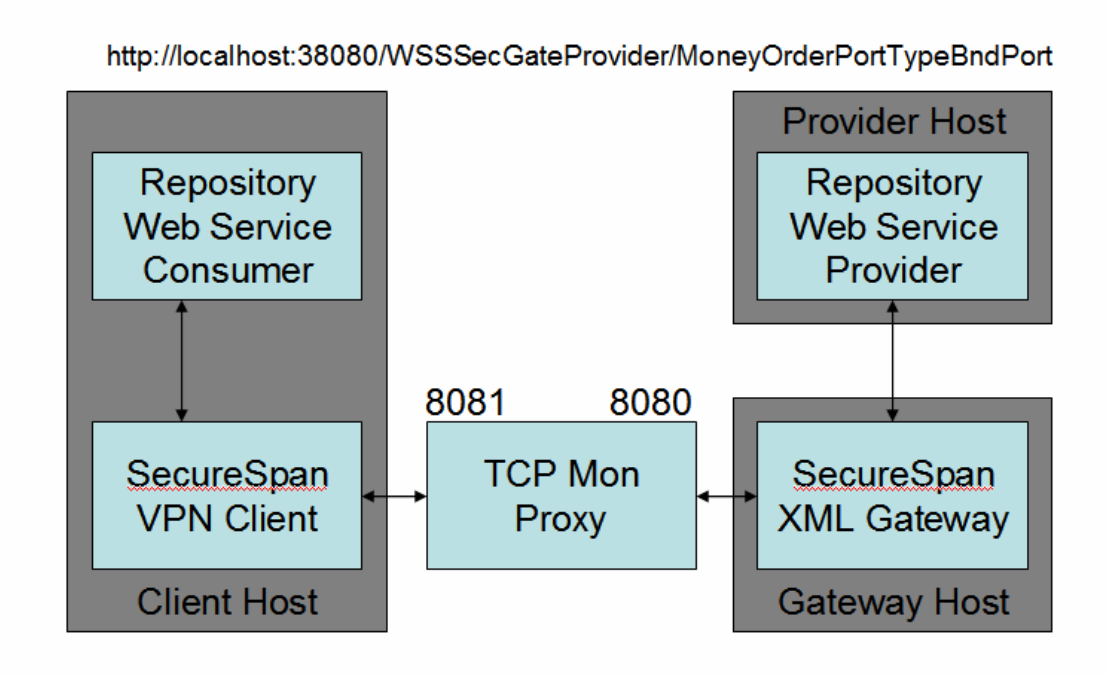

Figure 4-1 Secure Web Services solution to be built and deployed in this Note

In Stage 1 we will implement and deploy the Repository Web Service Provider, Figure 4-2, Section 7, "Build Repository-based Web Service Provider".

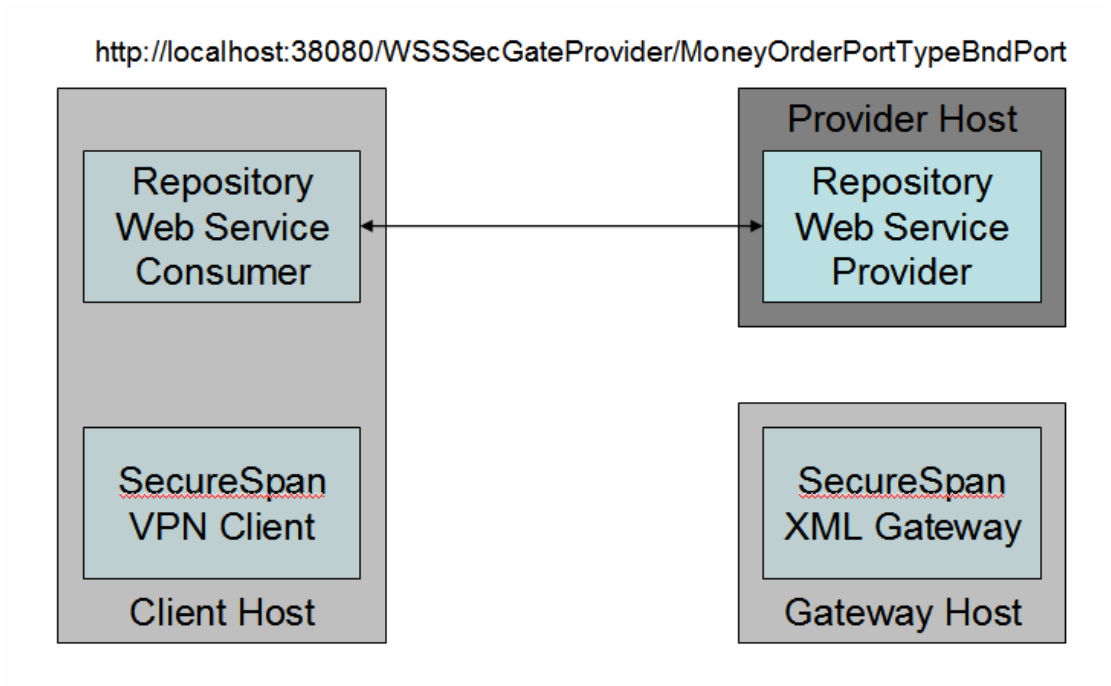

Figure 4-2 Stage 1 - Repository-based Web Service Provider

In Stage 2 we will implement and deploy the Repository Web Service Consumer, Figure 4-3, Section 8, “Build Repository-based Web Service Consumer”.

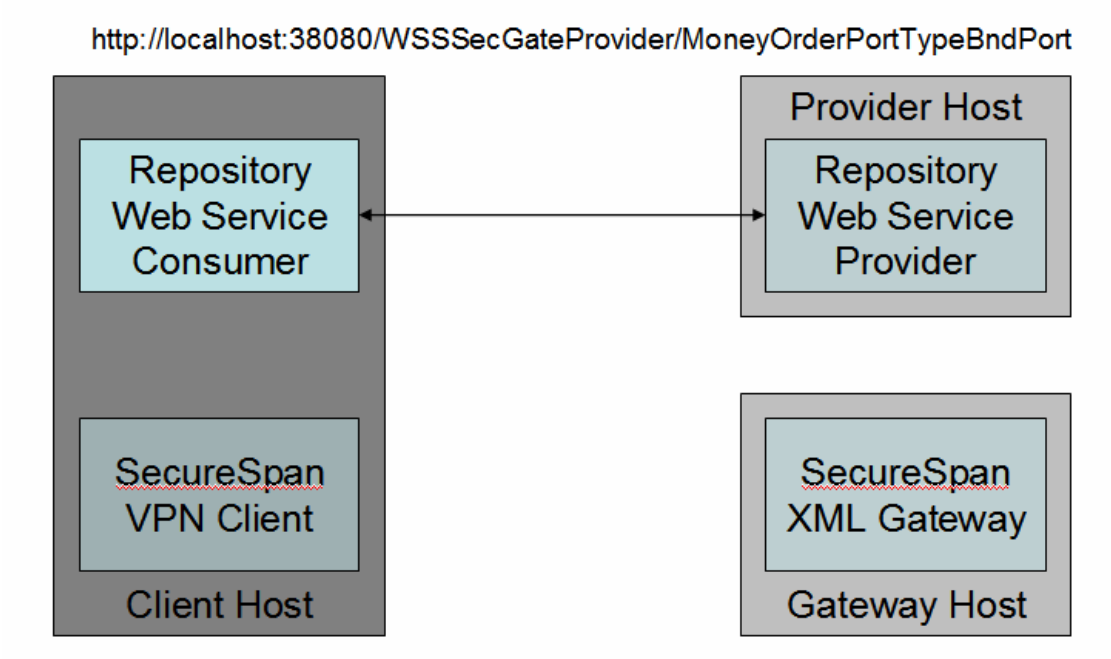

Figure 4-3 Stage 2, Repository-based Web Service Consumer

In Stage 3 we will test the Provider and the Consumer together, Figure 4-4, and trace on-the-wire message exchange, Figure 4-5.

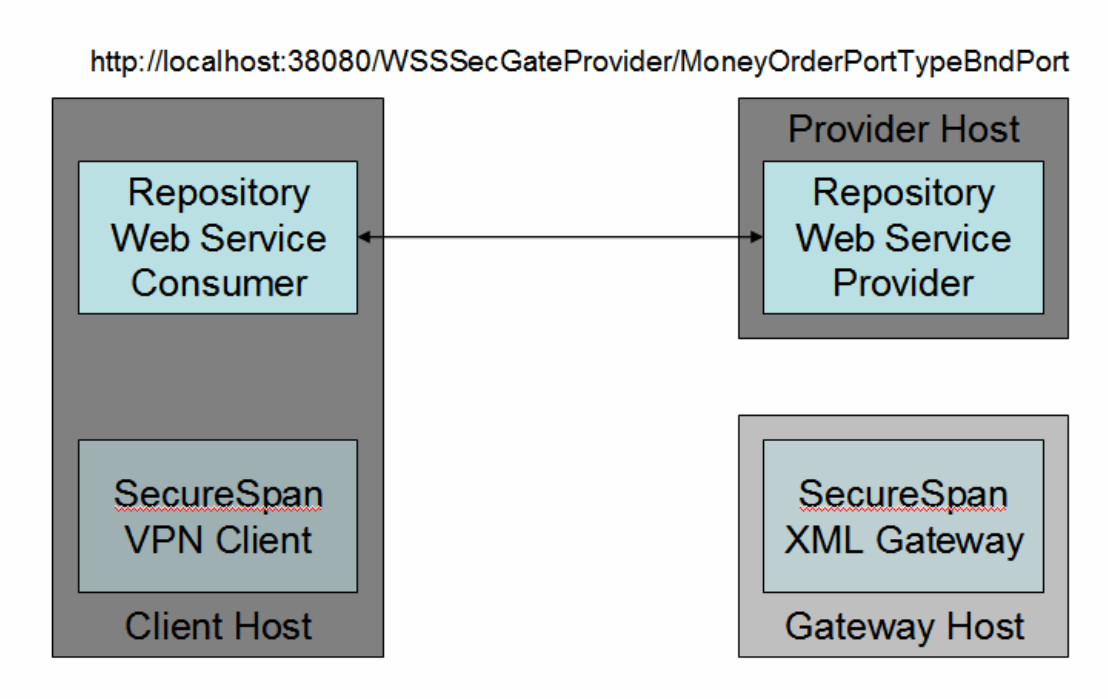

Figure 4-4 Stage 3 – Testing the Consumer and Provider

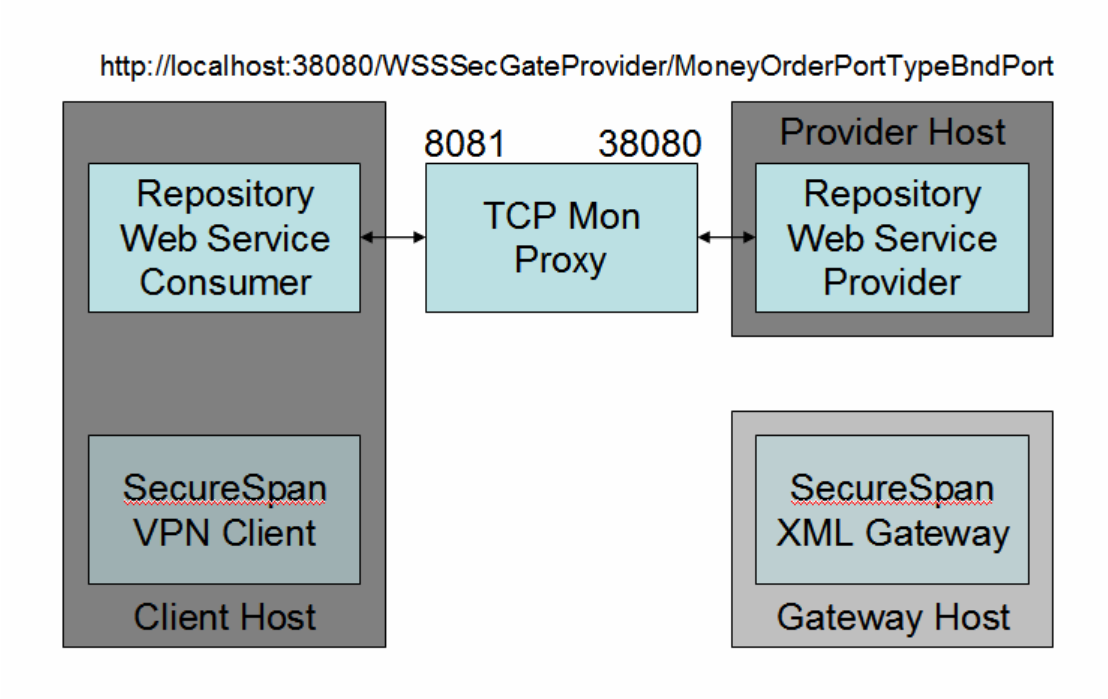

Figure 4-5 Trace on-the-wire message exchange

In Stage 4 we will add the SecureSpan XML Gateway to the mix, Figure 4-6, test the Repository Web Service Consumer using the SecureSpan XML Gateway-mediated method of invocation and observe the on-the-wire message exchange.

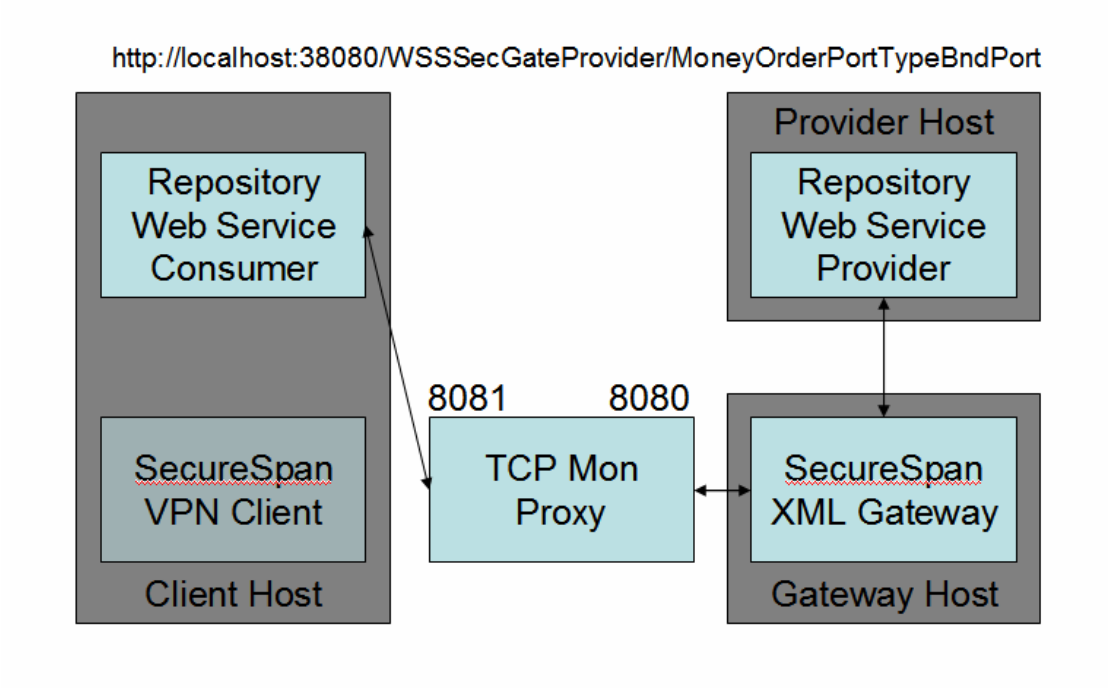

Figure 4-6 Stage 4 – introducing the SecureSpan XML Gateway as an intermediary

Finally, in Stage 5 we will add the SecureSpan VPN Client, Figure 4-1, and will spend a deal of time dynamically configuring various security policies, exercising the solution and observing the on-the-wire message exchanges.

5. Logging and Tracing

To watch what is happening on-the-wire from the GlassFish Application Server perspective, as web services consumers and providers exchange SOAP messages, add the following properties to the Application Server->Logging->Log Levels->Additional Properties.

com.stc.bpms.bpelImpl.runtime	FINEST
com.stc.eways	FINEST
com.stc.wsclient	FINEST
com.stc.wscommon	FINEST
com.stc.wsserver	FINEST
STC.eGate.CMap.Collabs	FINEST
STC.eWay.HTTP.client	FINEST
STC.eWay.HTTP.server	FINEST

com.stc.wsxxxxxx apply to Repository-based Web Services.

When done, re-start the application server to have the changes take effect.

Since the SecureSpan XML Gateway and the SecureSpan VPN Client will modify SOAP Requests after they have left the Repository-based Web Service Consumer and before they are received by the Repository-based Web Service Provider, and SOAP Responses will similarly be modified outside the control of the GlassFish Application Server, we will configure the solution to use the TCP Mon as an additional

intermediary. Section 15.6, “Obtain and use the Apache TCP Mon”, discusses how to get hold of the TCP Mon and how to configure it for different use. We will repeat configuration information at the time we add the TCP Mon to the solution for message exchange tracing.

6. Preliminaries

Note that, in the environment I used for this Note, all default TCP ports, changeable during Java CAPS 6 Update 1 installation, are modified by the addition of 30000, so 8080 (default HTTP port) becomes 38080, and so on.

Let’s beging by creating a project group, WSSecGate, and pointing it at a convenient file system directory. Once the project group is created, let’s create a new BPEL Module Project, WSSecGateCommon, to contain out XML Schema and WSDL documents. When doing so let’s make sure the correct path to the directory that will contain our artefacts is selected.

Let’s create a New -> XML Schema, MoneyOrder. Figure 6-1 illustrates a step in the process.

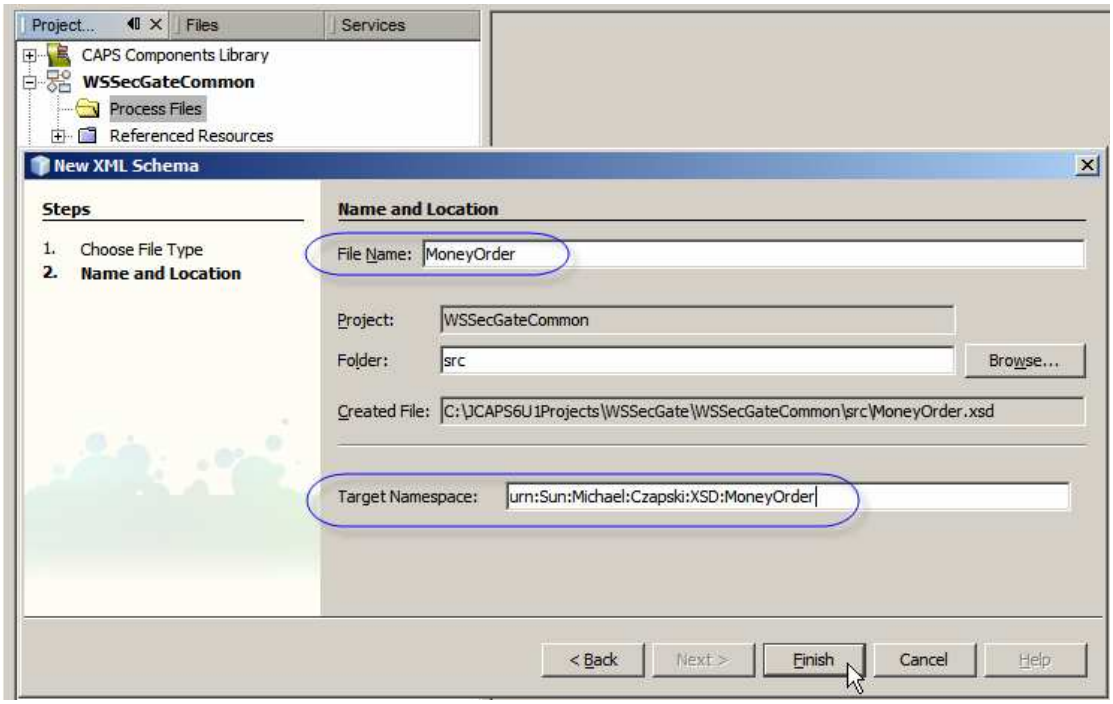

Figure 6-1 New XML Schema, namespace urn:Sun:Michael:Czapski:XSD:MoneyOrder

Once the skeleton is created, let’s open it in the XSD Editor, switch to the Source mode and replace the whole thing with the XML Schema shown in Figure 6-2.

Figure 6-2 MoneyOrder XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   targetNamespace="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
   xmlns="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
    <xs:element name="MoneyOrderReq">
        <xs:complexType>
```

```
        <xs:sequence>
            <xs:element name="OrderDetails">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="dateTime" type="xs:dateTime"/>
                        <xs:element name="seq" type="xs:nonNegativeInteger"/>
                        <xs:element name="total" type="xs:decimal"/>
                        <xs:element name="fee" type="xs:decimal"
                                    minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="SenderDetails" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="customerName" type="xs:string">
                        </xs:element>
                        <xs:element name="driverLicenseNumber"
                                    type="xs:string" minOccurs="0">
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="CreditCardDetails" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="cardType" type="xs:string">
                        </xs:element>
                        <xs:element name="nameOnCard" type="xs:string">
                        </xs:element>
                        <xs:element name="cardNumber" type="xs:string">
                        </xs:element>
                        <xs:element name="securityCode" type="xs:string">
                        </xs:element>
                        <xs:element name="validUntil" type="xs:string">
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="ReceipientDetails">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="familyName" type="xs:string">
                        </xs:element>
                        <xs:element name="givenName" type="xs:string"
                                    minOccurs="0">
                        </xs:element>
                        <xs:element name="streetAddress" type="xs:string">
                        </xs:element>
                        <xs:element name="cityTown" type="xs:string">
                        </xs:element>
                        <xs:element name="stateProvince" type="xs:string">
                        </xs:element>
                        <xs:element name="postalCode" type="xs:string">
                        </xs:element>
                        <xs:element name="country" type="xs:string"
                                    minOccurs="0">
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="MoneyOrderRes">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="OrderDetails">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="dateTime" type="xs:dateTime"/>
                        <xs:element name="seq" type="xs:nonNegativeInteger"/>
                        <xs:element name="total" type="xs:decimal"/>
                        <xs:element name="orderStatus" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
```

```
                <xs:element name="SenderDetails" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="customerName" type="xs:string">
                            </xs:element>
                            <xs:element name="driverLicenseNumber"
                                        type="xs:string" minOccurs="0">
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Let's create an Abstract WSLD document, MoneyOrder, that will represent the interface to the MoneyOrder service we will build next. Right-click on the name of the WSSecGateCommon project and choose New->WSDL Document. Name the document MoneyOrder, set the Target Namespace, keep the WSDL Type as Abstract and click Next, as shown in Figure 6-3.

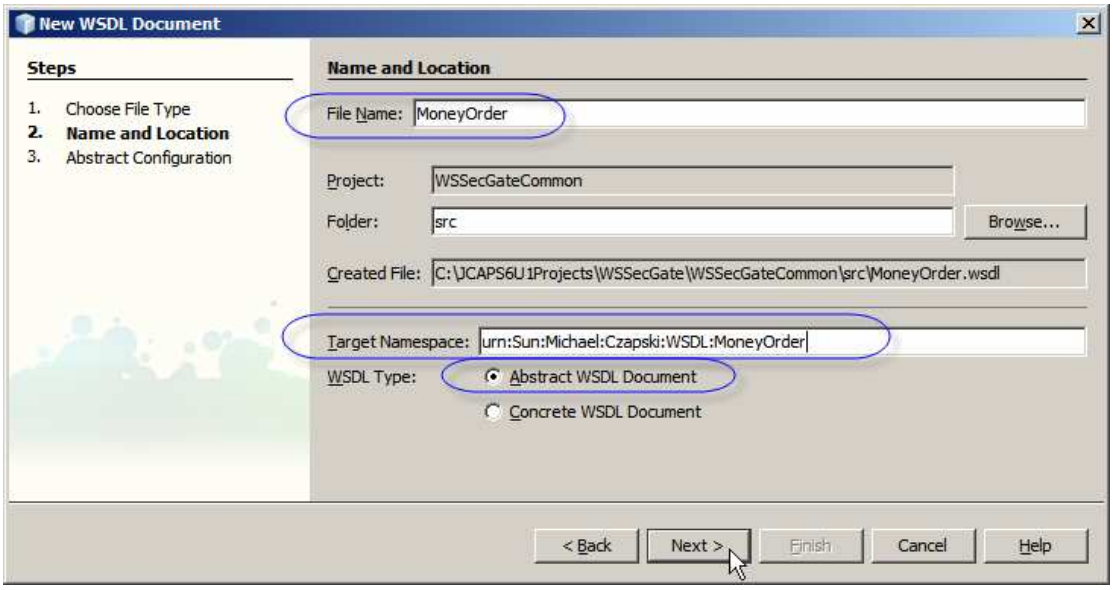

Figure 6-3 Start creation of the MoneyOrder WSDL

Change Operation Name to opMoneyOrder, name the Input and Output messages, choose the data types from the XML Schema and Finish. Figure 6-4 illustrates part of that process.

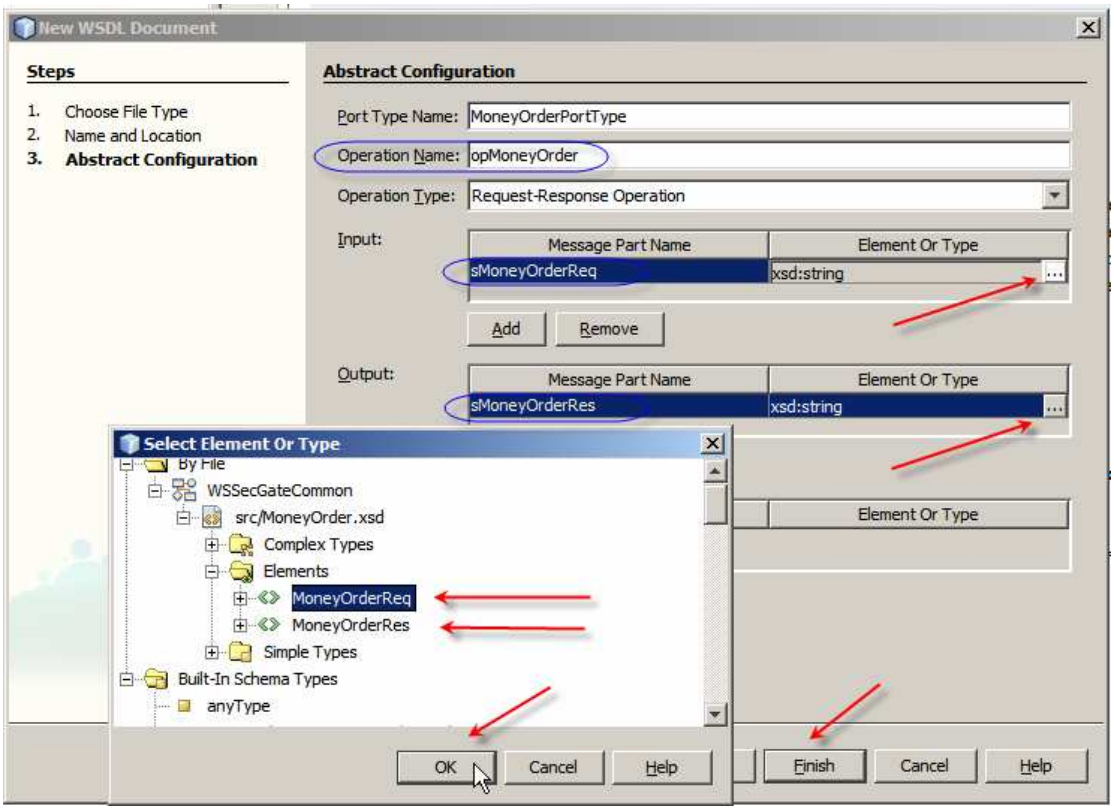

Figure 6-4 Name the operation, and name and type the input and output messages

To make the WSDL more interoperable we will create a variant of it in which the XML Schema is inlined, rather then being included from an external schema document.

Copy the MoneyOrder.wsdl and paste it with the new name of MoneyOrderInlined.wsdl. Open the XML Schema document, switch to source mode, select all except the first line and copy to the clipboard. Figure 6-5 illustrates this.

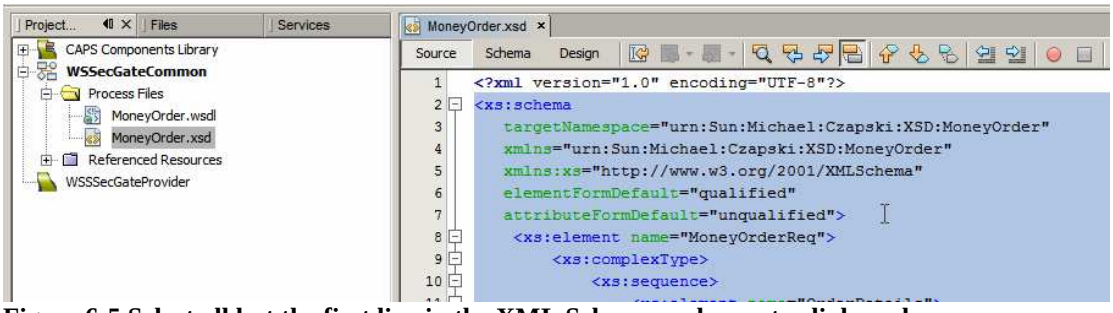

Figure 6-5 Select all but the first line in the XML Schema and copy to clipboard

Open the MoneyOrderInlined.wsdl, switch to source mode, select the three line starting with <xsd:schema … and finishing with </xsd:schema>, them paste the content of the clipboard, expected to be the XML Schema document's content, in their place. Figure 6-6 highlights the lines to select and replace with the content of the XML Schema MoneyOrder.xsd.

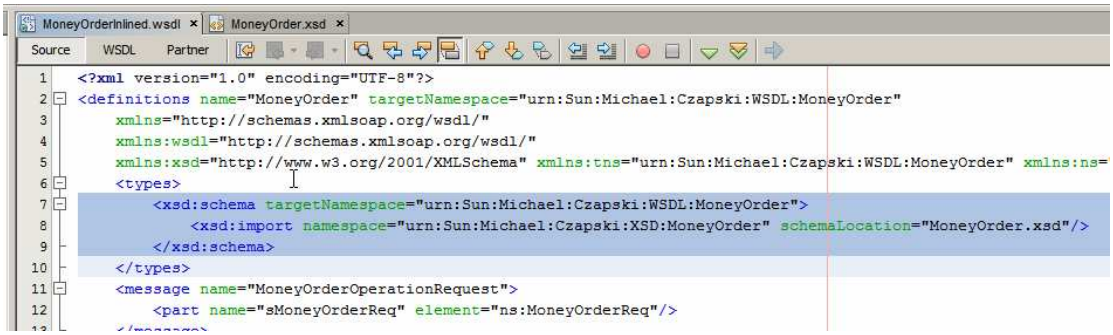

Figure 6-6 Select schema import statements

Check and Validate XML, as shown in Figure 6-7. Format the document with Alt+Shift+F or a right-click menu option.

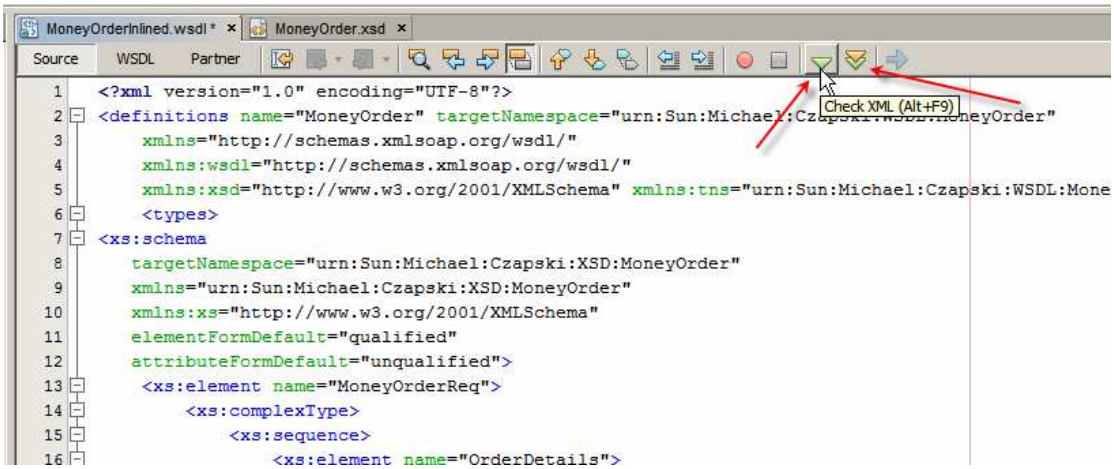

Figure 6-7 Check and Validate XML in the resulting WSDL document

This gives us the XML documents we will use in constructing the web service provider in the next section.

7. Build Repository-based Web Service Provider

Let's create a New Project->CAPS->ESB->CAPS Repository-based Project, named WSSSecGateProvider. The path to the option is shown in Figure 7-1.

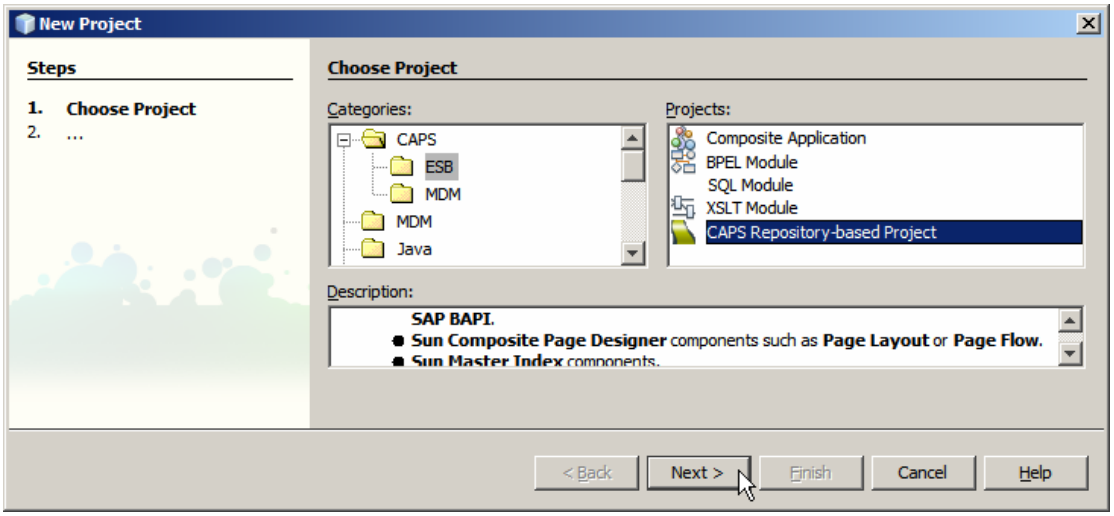

Figure 7-1 Create a New CAPS Repository-based Project

Let’s import the web service definition from the MoneyOrderInlined.wsdl file. Before we do, let’s copy the file system location of the WSDL document. Righ-click on the WSDL name, choose Properties, copy to the clipboard the value of the All Files property. This is shown in Figure 7-2.

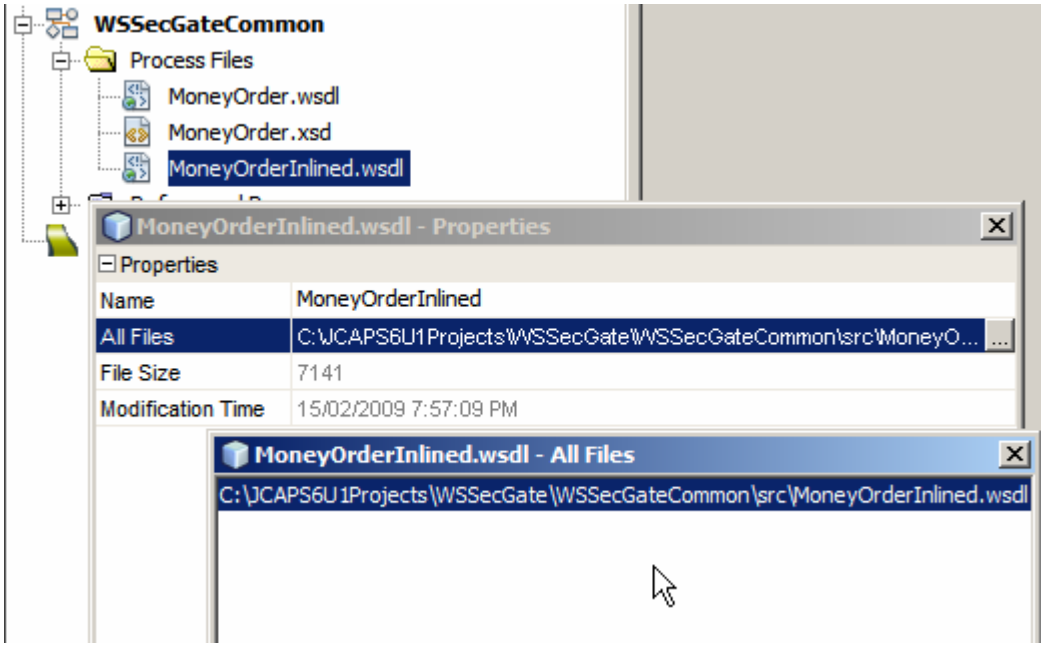

Figure 7-2 Copy the value of the All Files property to the clipboard.

Import Web Service Definition into the WSSSecGateProvider project. Figure 7-3 points out the menu options.

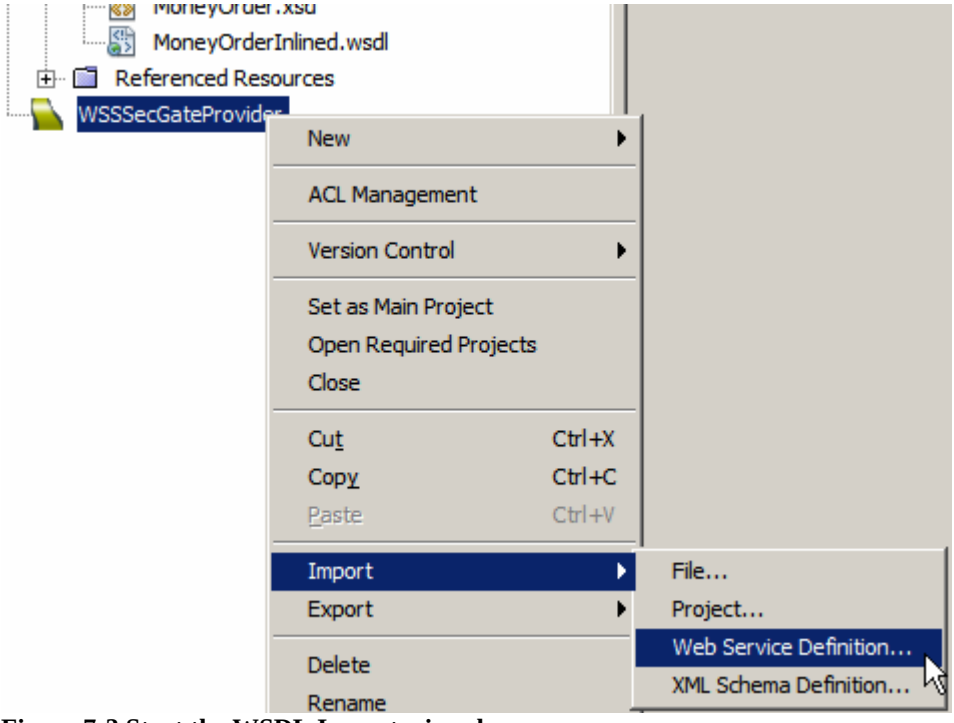

Figure 7-3 Start the WSDL Import wizard

Accept default Location Type of File System and click Next.

Paste the document location path into the File Name text box and press Enter. Until you do, the Next button will be inaccessible. Figure 7-4 illustrates this dialogue box.

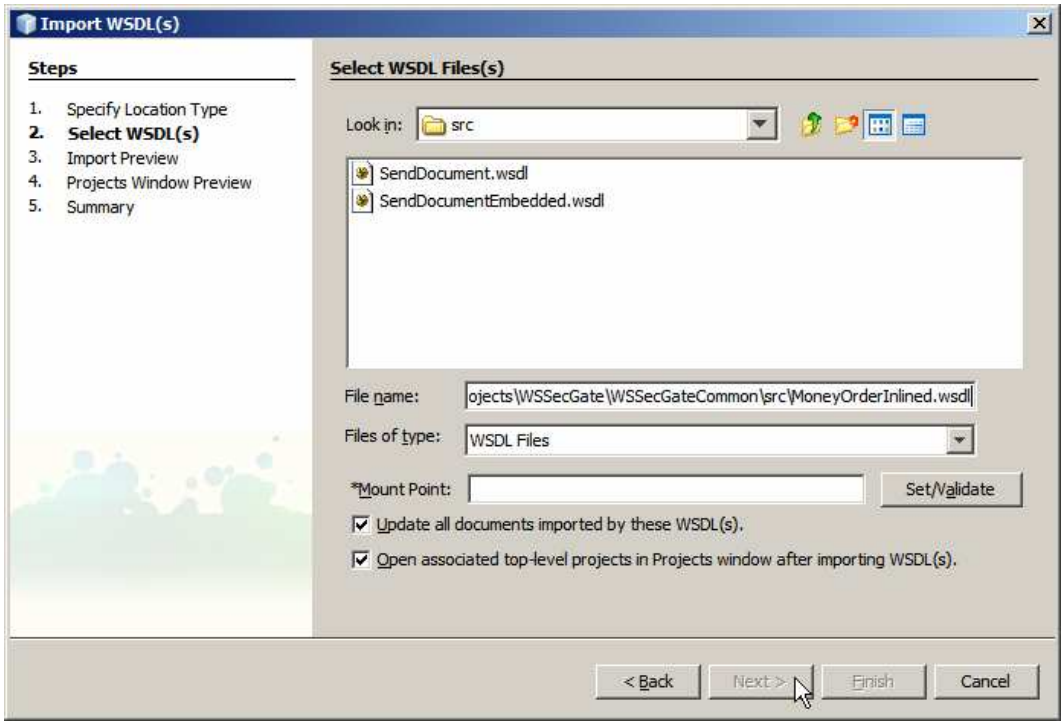

Figure 7-4 Paste the document location and press Enter

Press Next, Next, Next then Finish. The MoneyOrderInlined WSDL will be imported into the Project as shown in Figure 7-5.

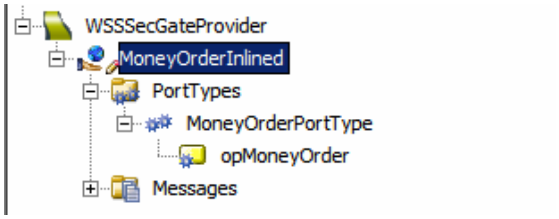

Figure 7-5 MoneyOrderInlined WSDL imported

Let's now create a new Business Process, bpWSSSecGateProvider, drag the opMoneyOrder operation onto the canvas and choose to "Implement the operation", as shown in Figure 7-6.

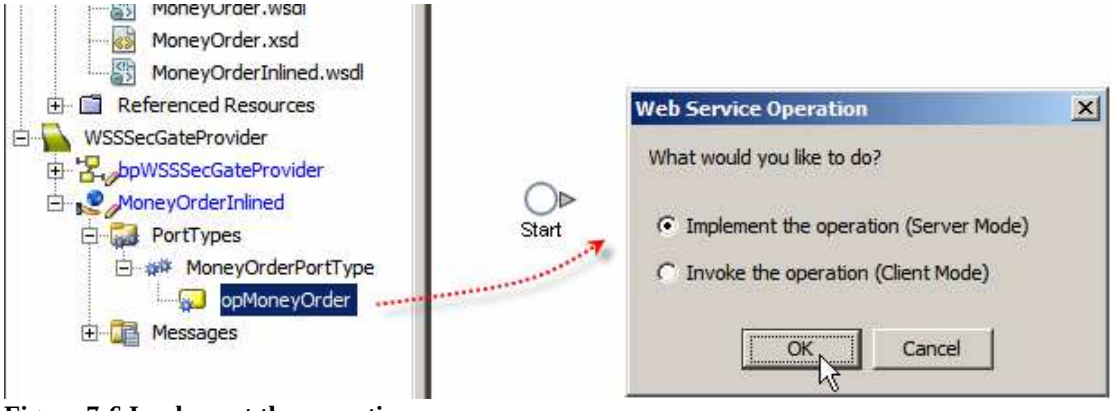

Figure 7-6 Implement the operation

Connect the activities, add a business rule between the Request and the Reply activities and map as shown in Figure 7-7.

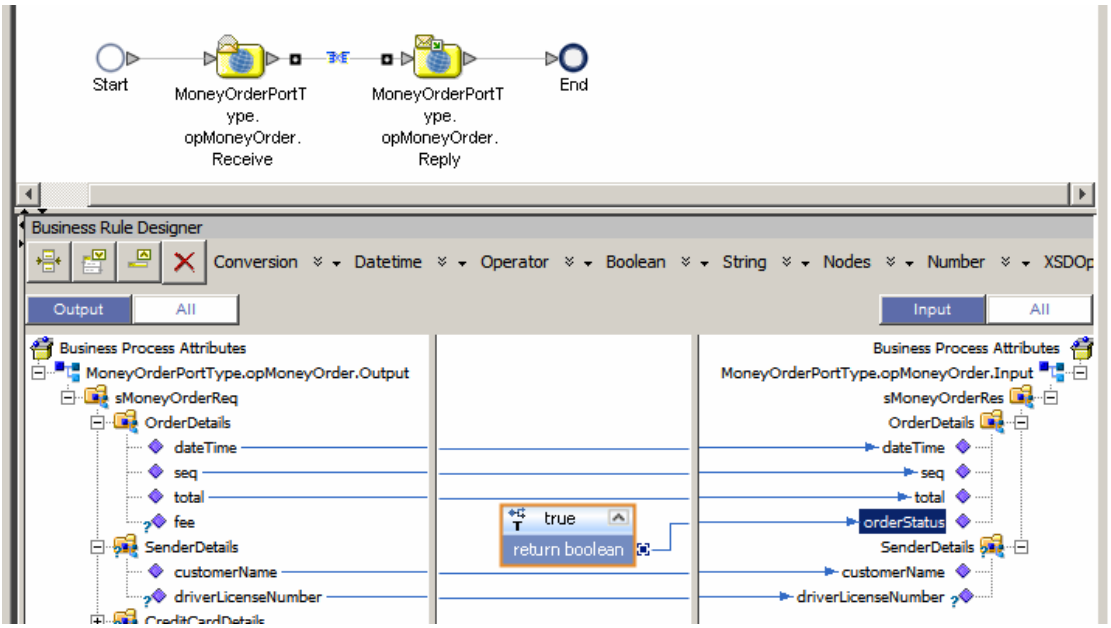

Figure 7-7 Map data from the Request to the Reply

Create a Connectivity Map, cmWSSSecGateProvider, drag the business process onto the connectivity map editor, add a Web Service External Application, connect and save. Figure 7-8 shows the completed connectivity map.

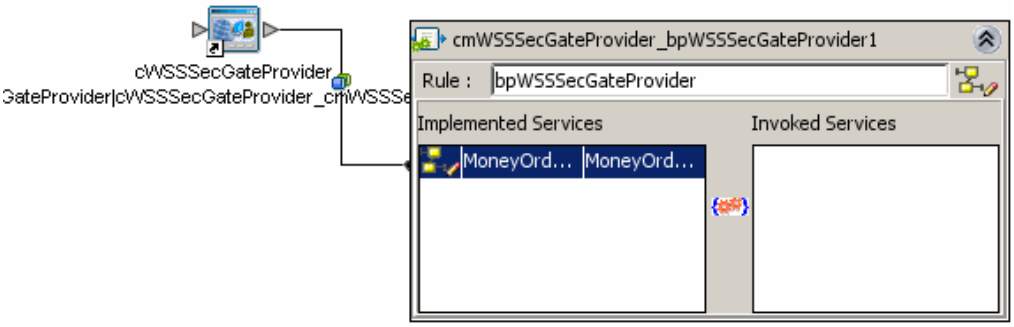

Figure 7-8 Completed Connectivity Map

If you have not done so, branch out at this point and create a Java CAPS Environment, as described in section 15.5, "Create Java CAPS Environment“, before continuing.

To this Environment we will now add the SOAP/HTTP Web Service External System … container, Figure 7-9, and configure it with the address, port and servlet context, Figure 7-10. The menu options are New->SOAP/HTTP Web Service External System …, off the CAPS Environment under the WSSecGateEnv environment.

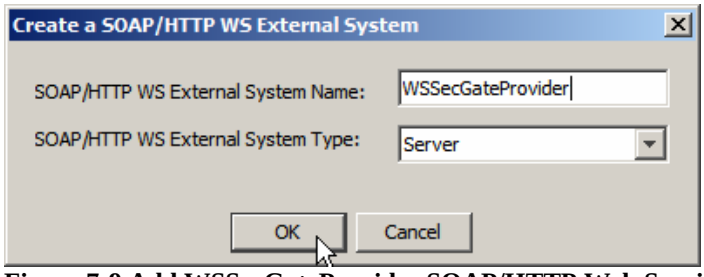

Figure 7-9 Add WSSecGateProvider SOAP/HTTP Web Service External System (Server)

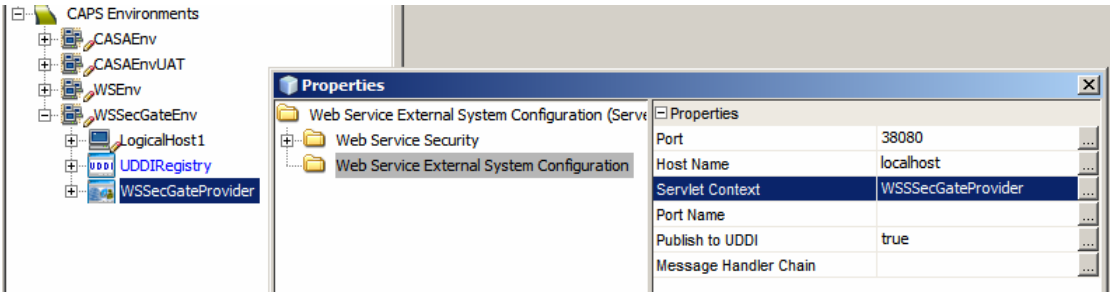

Figure 7-10 Configure SOAP/HTTP Web Service External System

Switch back to the Projects Tab and create a new Deployment Profile, dpWSSSecGateProvider using the WSSecGateEnv Environment. Figure 7-11 illustrates a step in the process.

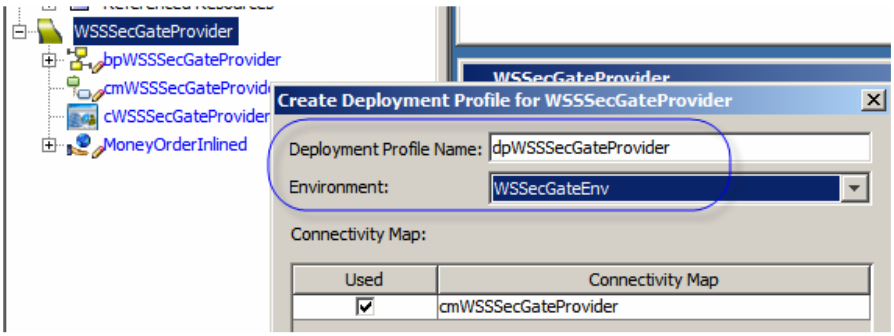

Figure 7-11 Deployment Profile using the WSSecGateEnv Environment

Automap the objects, build and deploy the solution. When building, you will be asked whether to publish the WSDL to UDDI. Allow this to happen.

The Repository-based Web Service Provider is not ready. Let's proceed to the Repository-based Web Service Consumer.

8. Build Repository-based Web Service Consumer

Let's create a New Project->CAPS->ESB->CAPS Repository-based Project, named WSCSecGateConsumer.

Let's switch to the Java CAPS UDDI Browser servlet, typically accessible through http://{GlassFishHost}:{GlassFishAdminPort}/CAPSUDDI/uddibrowse.jsp, click on the WSDL URL to view the WSDL and copy the WSDL URL to the clipboard. Figure 8-1 and Figure 8-2 illustrate the process.

Figure 8-1 Locate the WSDL URL for the Repository-based Web Service Provider

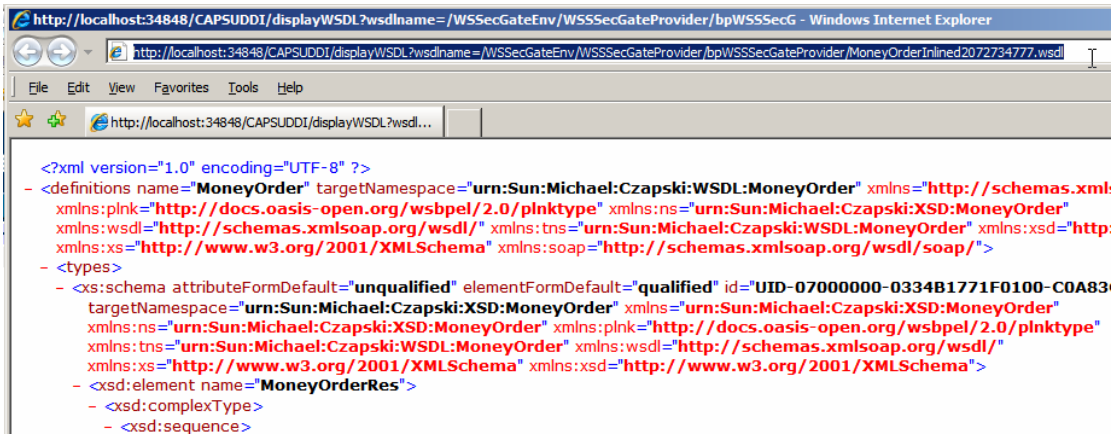

Figure 8-2 Copy WSDL URL to the clipboard

Right-click the name of the project, WSCSecGateConsumer, and choose New->Web Service Definition. Specify Location Type as URL, paste the WSDL URL into the URL text box, click the Add button and click Next as shown in Figure 8-3.

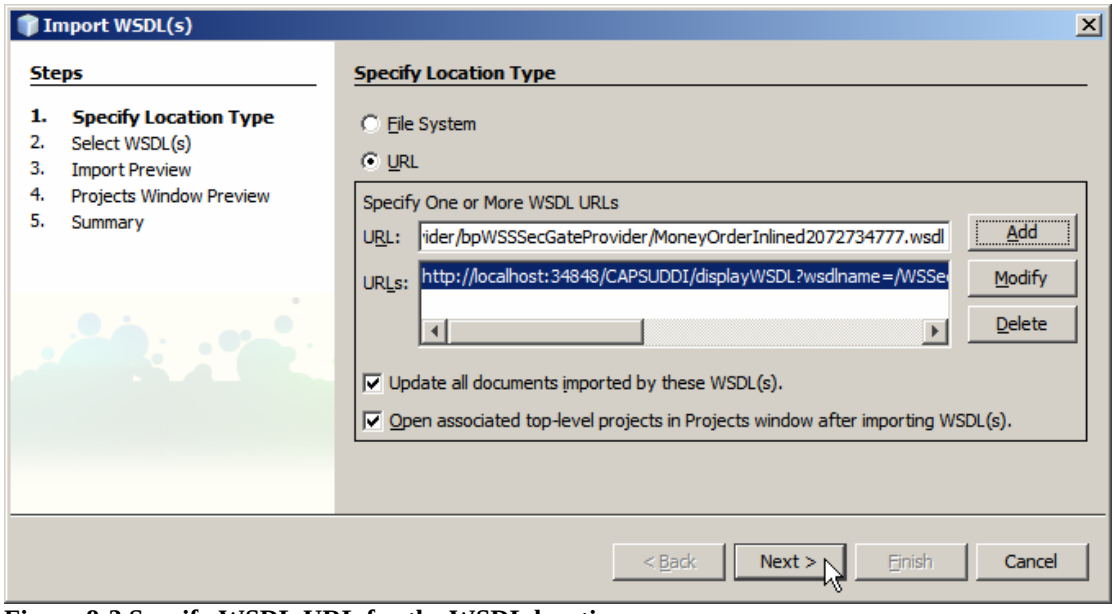

Figure 8-3 Specify WSDL URL for the WSDL location

Click Next, Next and Finish, to complete the import process. The WSDL object will appear inside the project. Figure 8-4 illustrates the project with the WSDL.

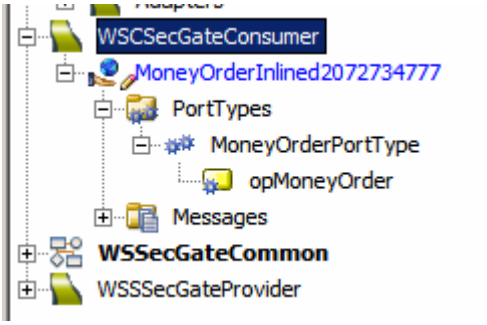

Figure 8-4 Newly imported WSDL object

Right-click on the name of the project, WSCSecGateConsumer, and choose New->Business Process. Name the process bpWSCSecGateConsumer.

The process will be invokes using a JMS TextMessage and will send the service response as a JMS TextMessage to a JMS Queue, with a time-to-live set to a couple of minutes so we have the time to look at the message but don't have to worry about clearing old messages from the Queue.

Let's expand the CAPS Components Library hierarchy through the Enterprise Service Bus and JMS nodes and drag the send and the receive services onto the business process editor canvas. Figure 8-5 illustrates this,

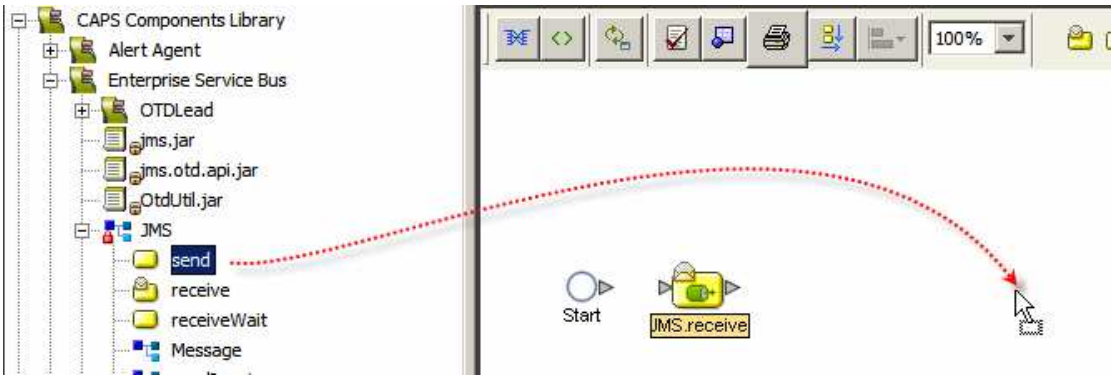

Figure 8-5 Add JMS receive and send services

Drag the opMoneyOrder operation from the project artefacts tree onto the business process canvas and choose to "Invoke …". Figure 8-6 illustrates this.

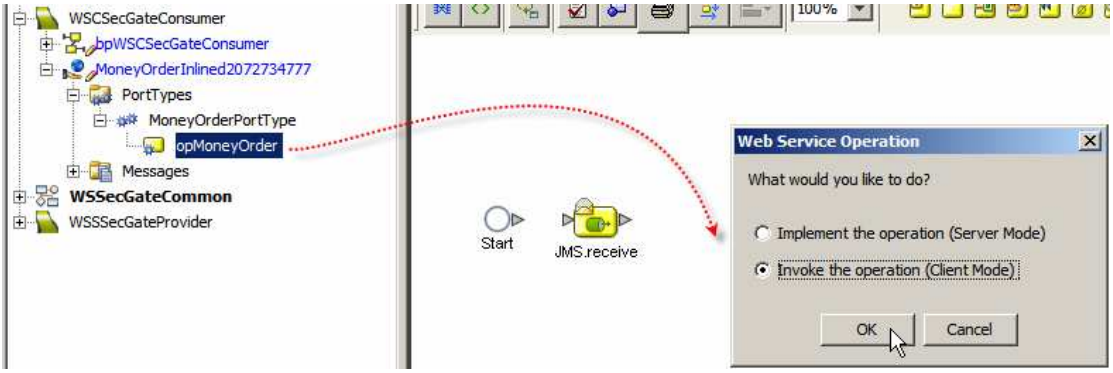

Figure 8-6 Add the opMoneyOrder service to the canvas and choose "Invoke …"

Connect all activities, add business rules to map request and response and map the request. Mapping in Figure 8-7 is not particularly clear. Table in Figure 8-8 spells out the mapping rules.

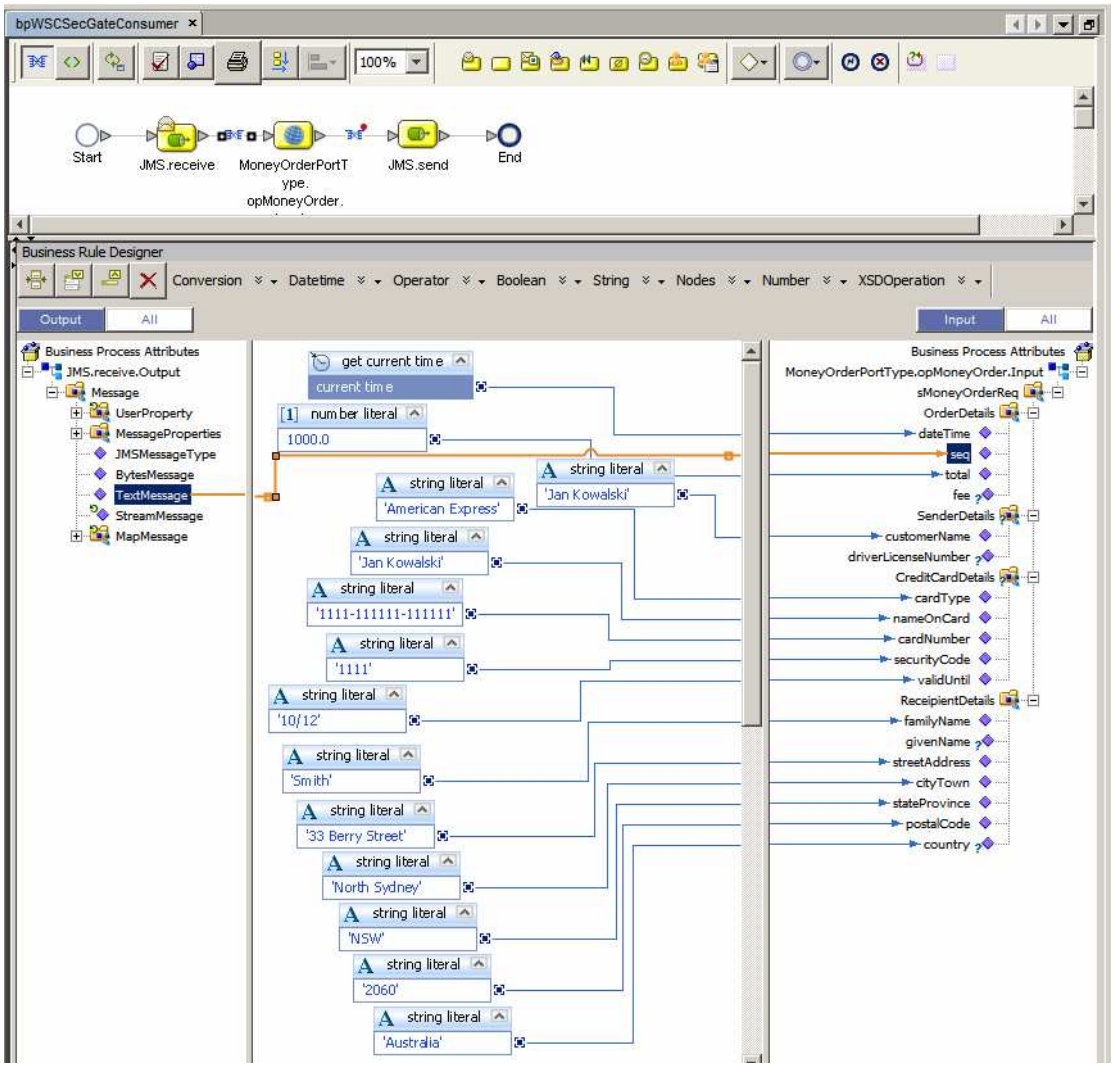

Figure 8-7 Business process activities connected, with mapping rules for the Request

Figure 8-8 Mapping rules for opMoneyOrder request

Source	Destination
Function “get current time”	sMoneyOrderReq->OrderDetails->dateTime
Message->TextMessage	sMoneyOrderReq->OrderDetails->seq
Literal “1000”	sMoneyOrderReq->OrderDetails->total
Literal “Jan Kowalski”	sMoneyOrderReq->SenderDetails->customerName
Literal “American Express”	sMoneyOrderReq->CreditCardDetails->cardType
Literal “Jan Kowalski”	sMoneyOrderReq->SenderDetails->nameOnCard
Literal “1111-111111-1111”	sMoneyOrderReq->SenderDetails->cardNumber
Literal “1111”	sMoneyOrderReq->SenderDetails->securityCode
Literal “10/12”	sMoneyOrderReq->SenderDetails->validUntil
Literal “Smith”	sMoneyOrderReq->ReceipientDetails->familyName
Literal “33 Berry Street”	sMoneyOrderReq->ReceipientDetails->streeAddress
Literal “North Sydney”	sMoneyOrderReq->ReceipientDetails->cityTown
Literal “NSW”	sMoneyOrderReq->ReceipientDetails->stateProvince

Literal “2060”	sMoneyOrderReq->ReceipientDetails->postalCode
Literal “Australia”	sMoneyOrderReq->ReceipientDetails->country

Use the XSDOperation->marshal function to obtain the XML string version of the web service response to send to the outbound JMS Queue, and configure a time-to-live property to 120000 milliseconds. Figure 8-9 illustrates the mapping and points out the dropdown where the marshal function is hiding.

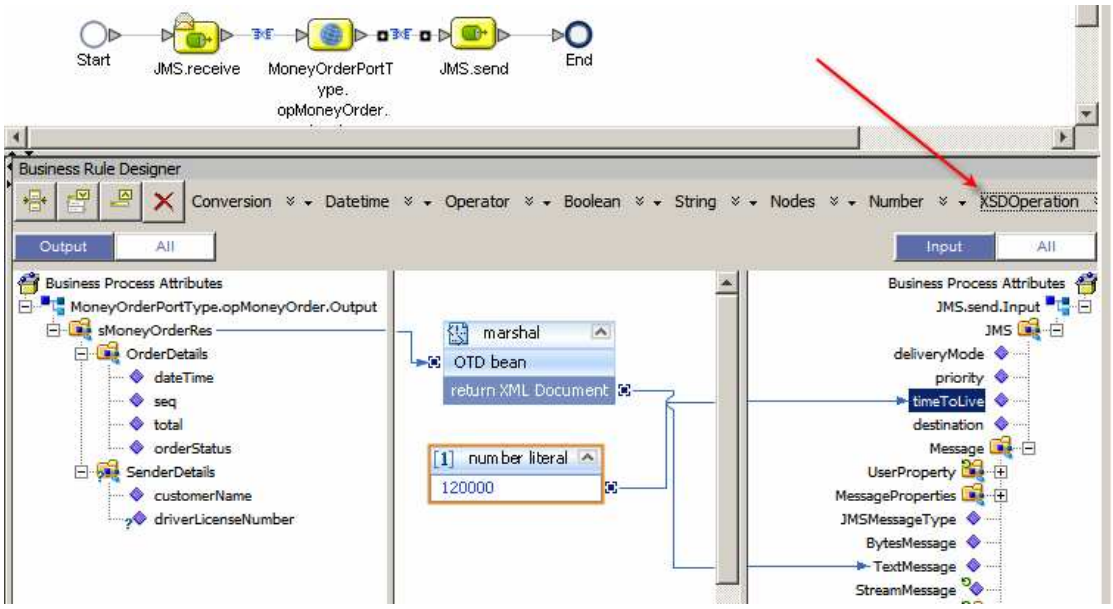

Figure 8-9 Map response to a JMS TextMessage

With the business process ready, let’s create a Connectivity Map, cmWSCSecGateConsumer, drag the Web Service External Application connector, and two JMS Queue objects – qWSCSecGateConsumerIn and qWSCSecGateConsumerOut, connect and save. Figure 8-10 shows the expanded connectivity map.

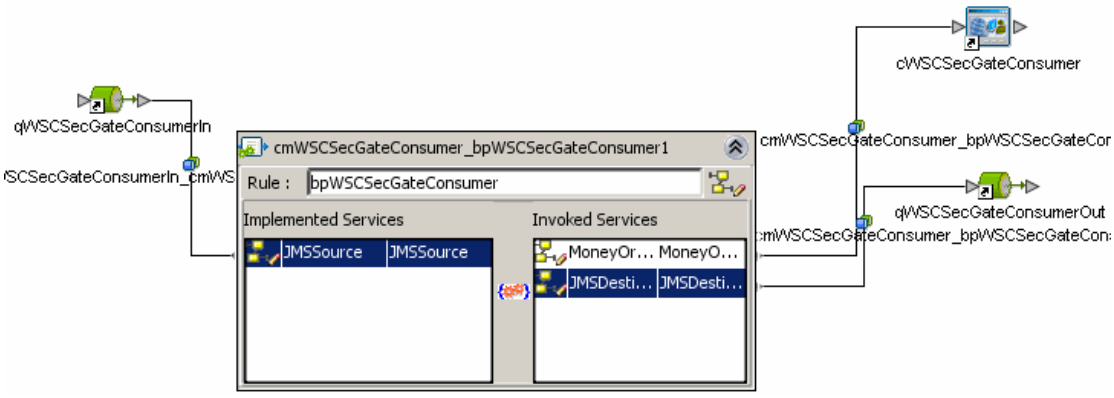

Figure 8-10 Connectivity Map, cmWSCSecGateConsumer

To deploy this project we need an additional SOA/HTTP Web Service External System Client container in the CAPS Environment. Let’s switch to the Services Tab, expand the CAPS Environments, WSSecGateEnv and add the container, WSCSecGateConsumer – see Figure 8-11.

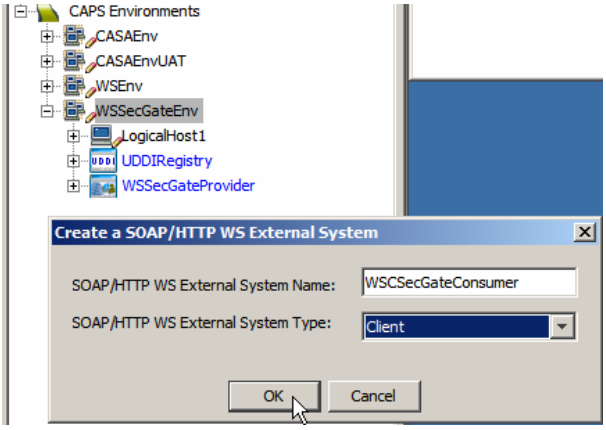

Figure 8-11 Add SOAP/HTTP Web Service External System (Client) container

Start the UDDI Browser Web UI, click on the WSDL associated with the WSSecGateEnv, bpWSSecGateProvider process, scroll down to the soap:address tag and copy to the clipboard just the servlet context. We need this context to configure the Client Container. Figure 8-12 highlights the servlet context.

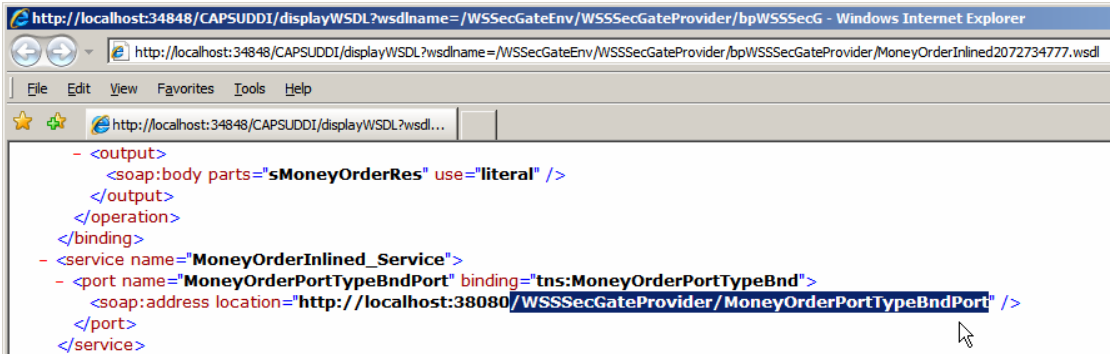

Figure 8-12 Copy servlet context to clipboard

Switch back to NetBeans, right-click the WSCSecGateConsumer container in the WSSecGateEnv Environment and configure as shown in Figure 8-13, changing host and port to suit your environment, if necessary.

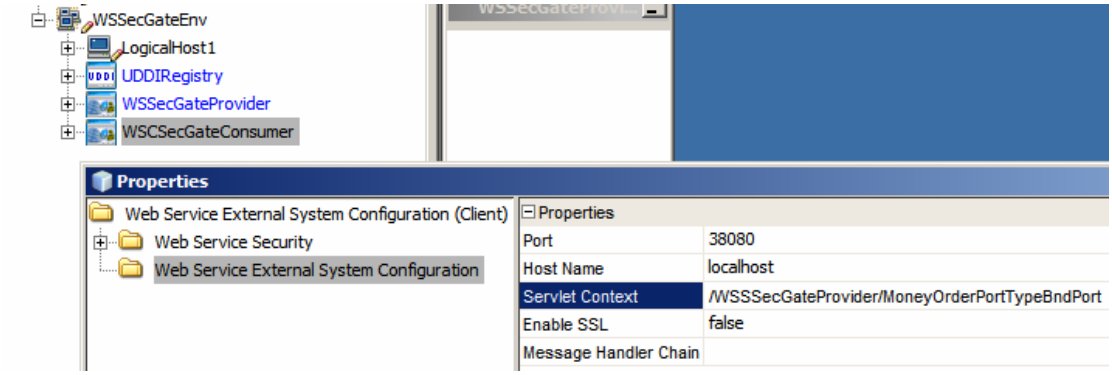

Figure 8-13 Configure the WSCSecGateConsumer container

Switch back to the Projects Tab in NetBeans, create a New -> Deployment Profile, dpWSCSecGateConsumer for the CAPS Environment WSSecGateEnv, automap, build and deploy.

9. Exercise Consumer and Provider

Let's now exercise the solution, with the consumer directly invoking the provider, by submitting a message to qWSCSecGateConsumerIn and observing the response in qWSCSecGateConsumerOut using the Java CAPS Enterprise Manager.

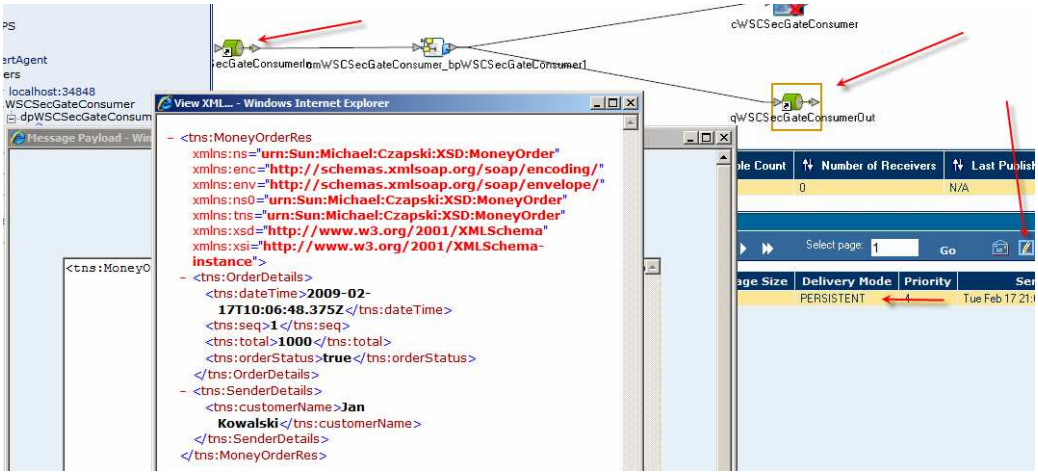

Figure 9-1 Response message in qWSCSecGateConsumerOut

Indeed the request was submitted and the response was returned.

To observe what is happening on-the-wire, we will add the TCP Mon to the solution and modify the service consumer to connect to the provider via the intermediary. Figure 9-2 illustrates the configuration.

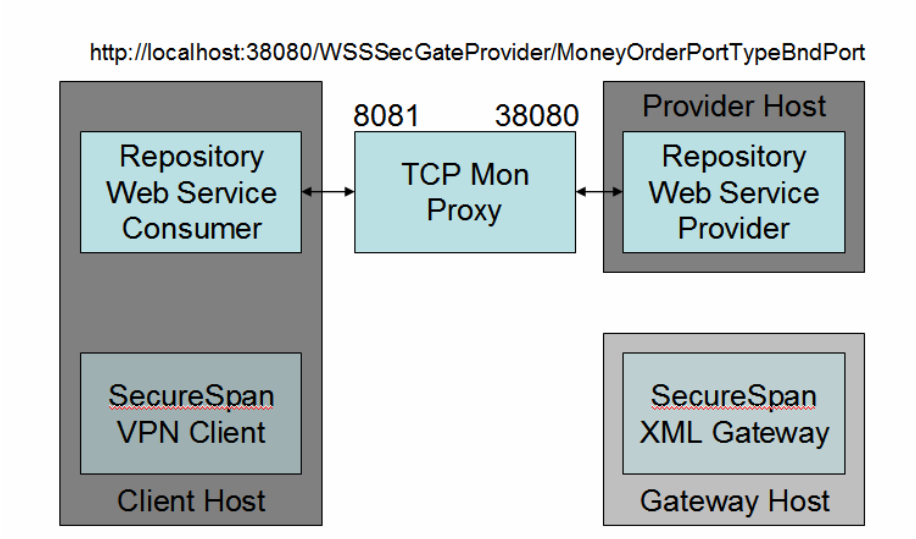

Figure 9-2 Configuration with TCP Mon

If you have not already done so, obtain and "install" the TCP Mon. Section 15.6, "Obtain and use the Apache TCP Mon", discusses how to obtain and configure the tool.

Change the working directory to the location of the tcpmon.bat (if on Windows) and start TCP Mon with the following command:

```
C:> cd C:\tools\tcpmon-1.0-bin\build
C:> tcpmon.bat 8081 localhost 38080
```

Click on the “Port 8081” Tab. This is where the request and response will appear. Check the XML Format checkbox, Figure 9-3, to get the XML pretty printed.

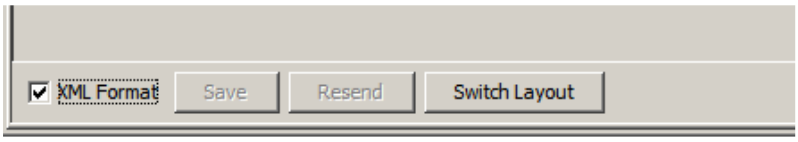

Figure 9-3 Enable pretty printing of XML

Switch to the NetBeans IDE, Services Tab, CAPS Environment, WSSecGateEnv and edit properties of the WSCSecGateConsumer, see Figure 9-4. Change Port number from 38080 to 8081, save, switch to project Tab, and build and deploy the project WSCSecGateConsumer using the dpWSCSecGateConsumer deployment profile.

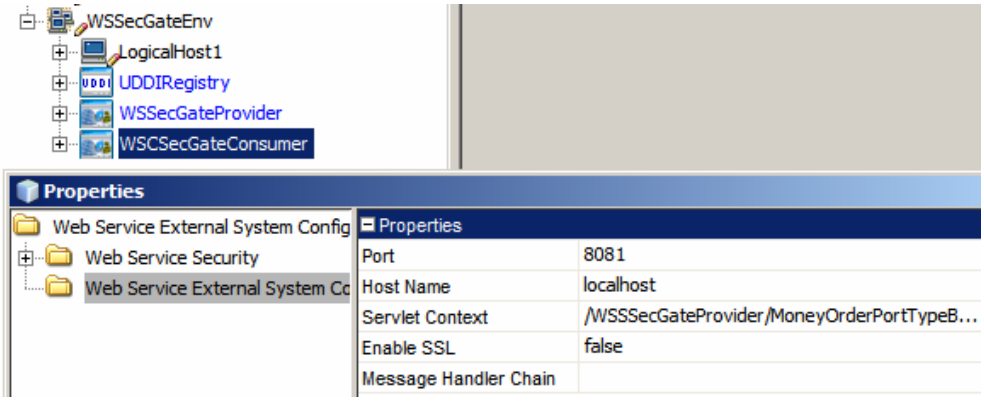

Figure 9-4 Change port number to 8081

Once the modified consumer project is deployed use the Java CAPS Enterprise Manager to submit a message to the inbound queue, check the outbound queue for the response then switch to TCP Mon and look at the request and the response through it.

The response, following submission of a JMS TextMessage containing the numeral 111, is shown in Figure 9-5.

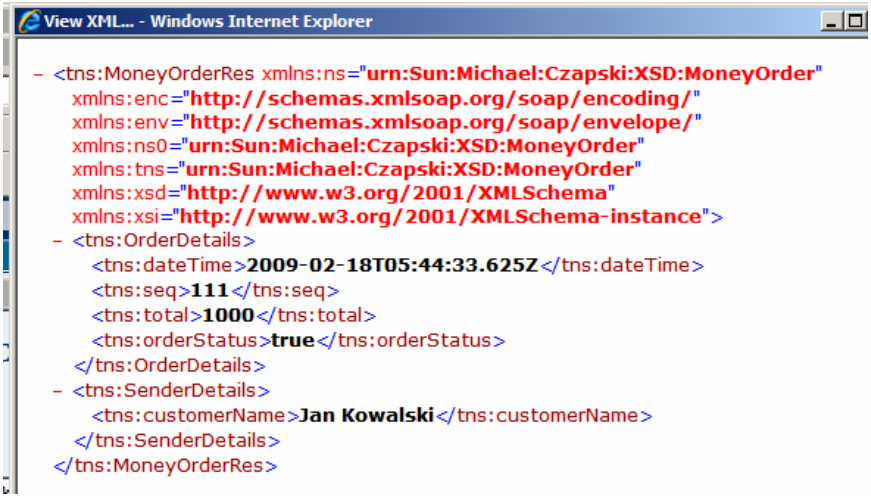

Figure 9-5 Response to a message containing numeral 111

Part of the request and the response, as seen by TCP Moon, is shown in Figure 9-6.

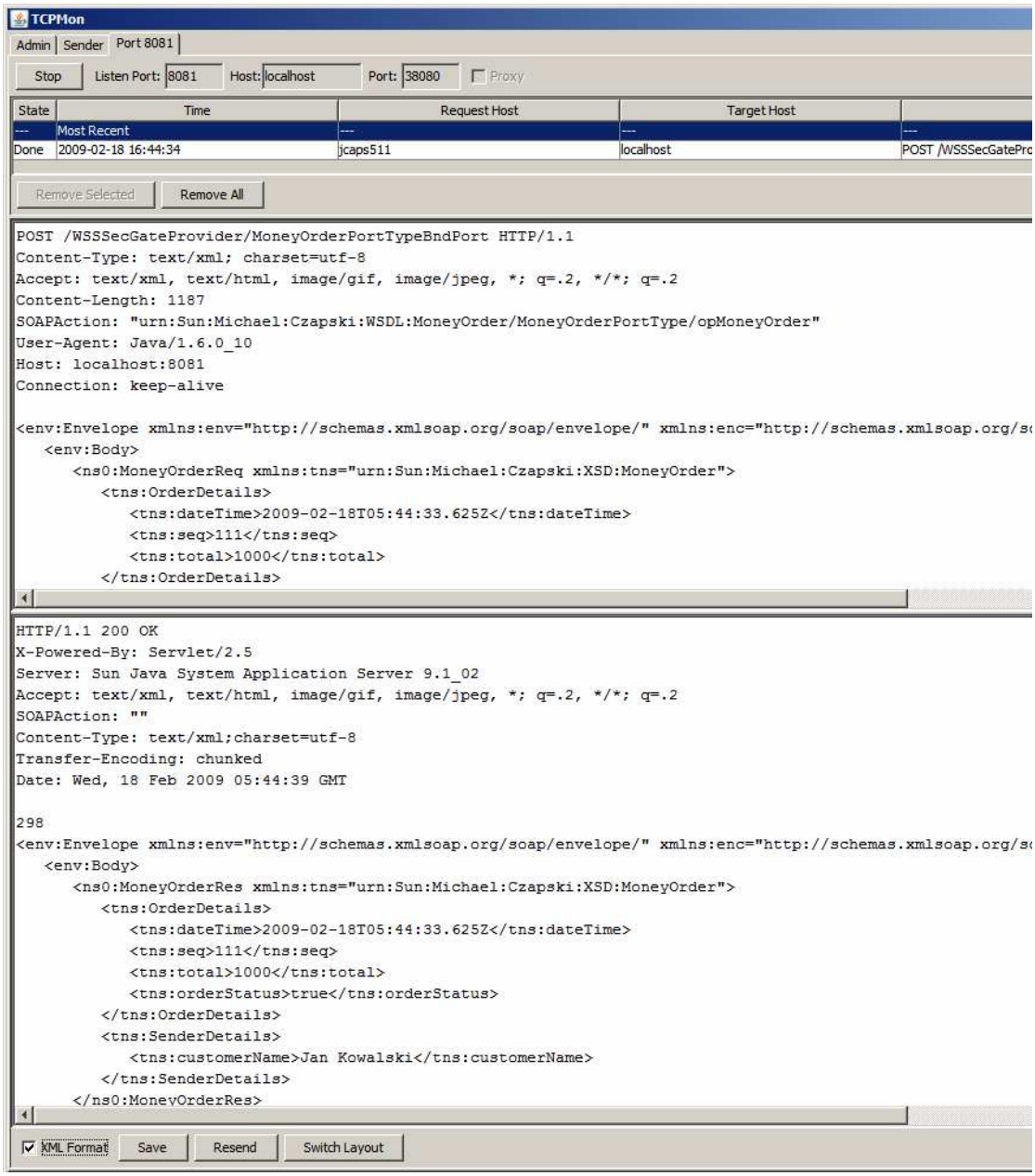

Figure 9-6 Part of the request and the response in TCP Mon

Note that no SOAP Headers are present in either the request or the response.

10. Add a Gateway to the Mix

Let's add the SecureSpan XML Gateway to the mix. The schematic of the solution is shown in Figure 10-1.

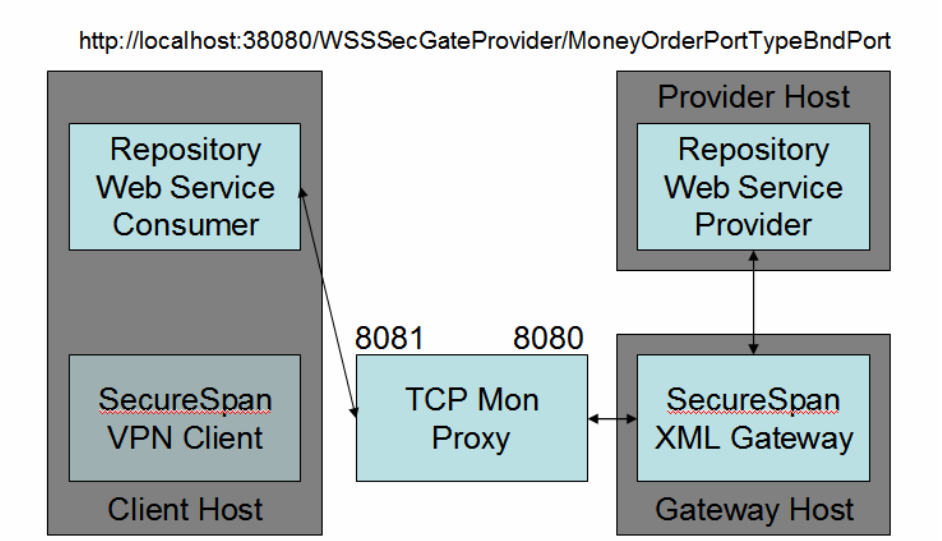

Figure 10-1 Solution that includes the SecureSpan XML Gateway

If you have not done so already, branch out to sections 15.1, “Obtain the Layer 7SecureSpan XML Gateway” and 15.2, “Configure the SecureSpan XML Gateway” to install and configure the SecureSpan XML Gateway so it is ready to be used as we work through this section.

Assuming the Gateway has been installed and configured, start it.

The gateway will run, for me, on the host ssg.aus.sun.com. Figure 10-2 shows host and port assignments for the components of the solution. These reflect my environment. Your environment will use different host names and, likely, different port numbers.

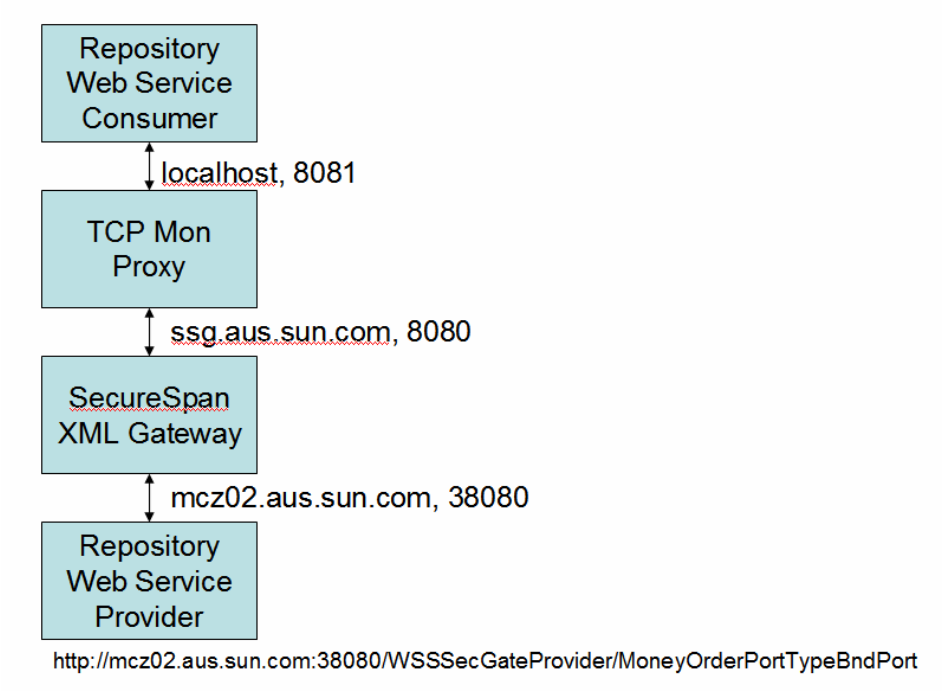

Figure 10-2 Host and port assignments

Stop the TCP Mon client and start it, using the new host and port number for the target host. Figure 10-3 shows the command line.

```
C:\tools\tcpmon-1.0-bin\build>tcpmon.bat 8081 ssg.aus.sun.com 8080_
```

Figure 10-3 Start the TCP Mons and point it at the SecureSpan XML Gateway

We have not yet configured the gateway to recognise the web service provider and to redirect requests to it. Let's submit the JMS message to the consumer and observe message exchange in the TCP Mon window. As was to be expected, the gateway returns a SOAP Fault. Figure 10-4 show the exchange.

```
POST /WSSSecGateProvider/MoneyOrderPortTypeBndPort HTTP/1.1
Content-Type: text/xml; charset=utf-8
Accept: text/xml, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Length: 1187
SOAPAction: "urn:Sun:Michael:Czapski:WSDL:MoneyOrder/MoneyOrderPortType/opMoneyOrder"
User-Agent: Java/1.6.0_10
Host: ssg.aus.sun.com:8081
Connection: keep-alive

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:enc="http://schemas.xmlsoap.org/soap
   <env:Body>
      <ns0:MoneyOrderReq xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
         <tns:OrderDetails>
            <tns:dateTime>2009-02-18T08:16:52.656Z</tns:dateTime>
            <tns:seq>121</tns:seq>
            <tns:total>1000</tns:total>
         </tns:OrderDetails>
```

```
HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Content-Length: 712
Date: Wed, 18 Feb 2009 17:42:06 GMT
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
         <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>Policy Falsified</faultstring>
            <faultactor>http://ssg.aus.sun.com:8081/WSSSecGateProvider/MoneyOrderPortTypeBndPort</faultactor>
            <detail>
               <l7:policyResult
                        status="Service Not Found.  The request may have been sent to an invalid URL,
                        or intended for an unsupported operation."
                    xmlns:l7="http://www.layer7tech.com/ws/policy/fault"/>
            </detail>
         </soapenv:Fault>
      </soapenv:Body>
   </soapenv:Envelope>
```

Figure 10-4 SOAP Fault from the Gateway

Note that I reformatted the text associated with the status attribute of the policyResult node so it could fit into the picture and be readable.

To allow the consumer and the provider to communicate we must "introduce" the provider to the Gateway. Let's start the browser-based SecureSpan Manager. The URL for me will be https://ssg.aus.sun.com:8443/ssg/webadmin.

Let's click the "Publish SOAP Web Service" "button", see Figure 10-5.

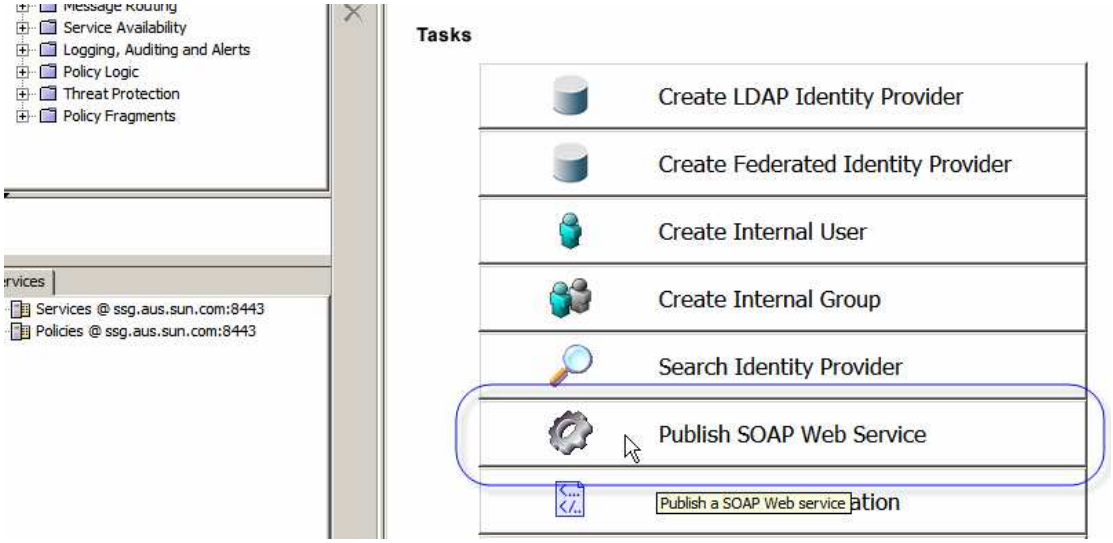

Figure 10-5 Click the Publish SOAP Web Service "button"

Start the UDDI Browser web interface, navigate to the WSDL for the provider and copy WSDL URL to the clipboard. We will need it to configure the service in the Gateway. Remember that the SSG Gateway runs on its own host so any references to "localhost" will have to be replaced with the Fully Qualified Domain Name (FQDN) of the host to which the web service is actually deployed. For me this will be mcz02.aus.sun.com. Also make sure that the Gateway host can resolve the FQDN of the web service host. If it cannot then you will need to use the IP address or modify the Gateway hosts file to add this web service host to it. Note also that if you modify the hosts file it will be revert on reboot of the SSG.

Paste the WSDL URL into the text box, see Figure 10-6, modify the host name if required, and click Next.

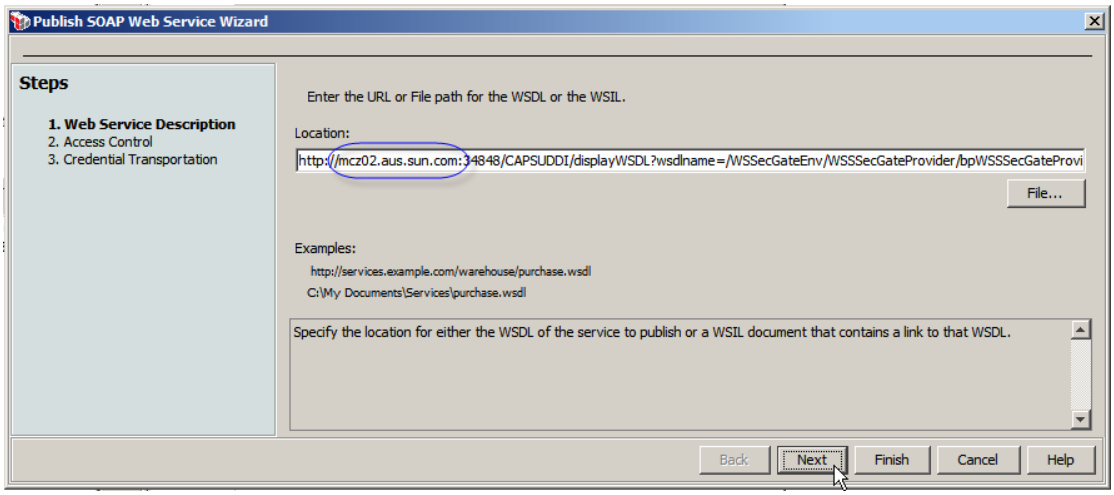

Figure 10-6 Make sure to use the FQDN of the host on which the web service runs

Note the controls for enabling channel encryption, authenticated access and permissions, Figure 10-7. Leave Allow Anonymous Access enabled and click next.

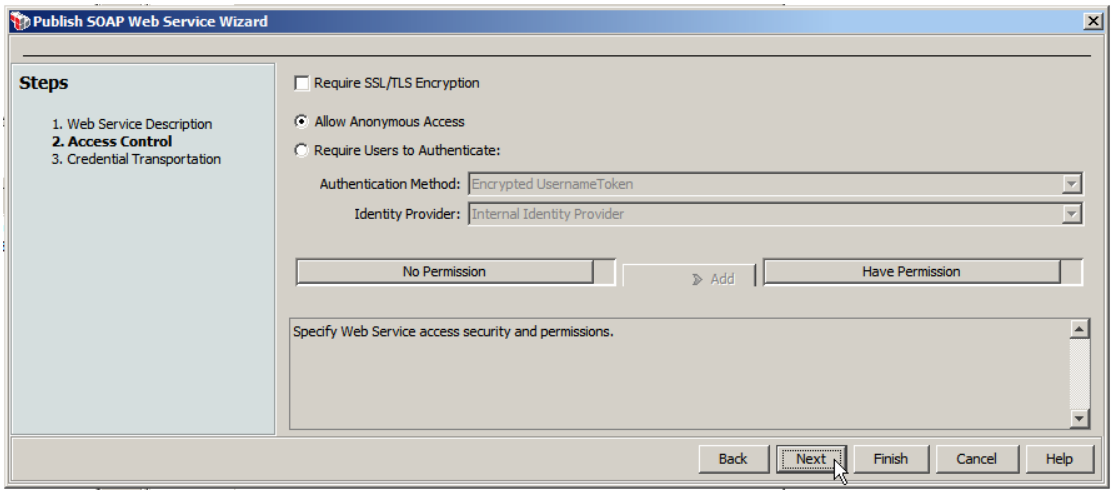

Figure 10-7 Access control controls

Make sure that the web service end point URL is resolvable from the gateway. The original URL used localhost for the host name. It must be changed to the FQDN of the host that hosts the web service, see Figure 10-8. Note, also, that the gateway can provide credentials, if the service requires them.

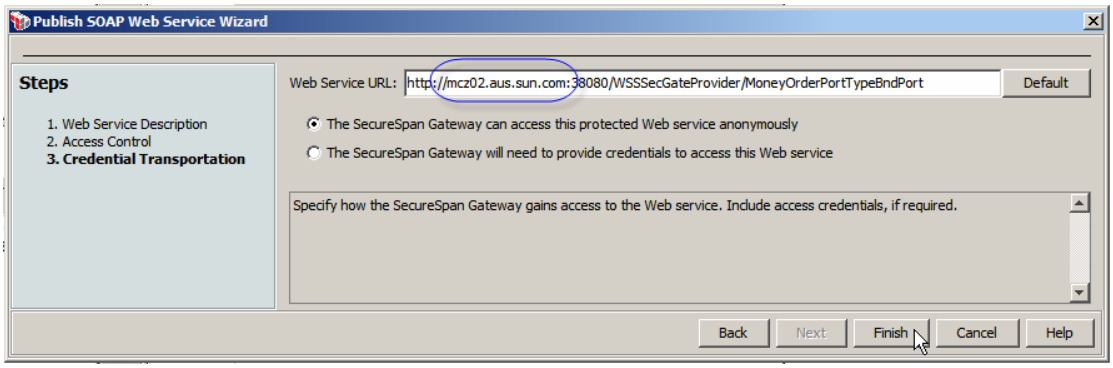

Figure 10-8 Make sure the web service endpoint URL is correct and resolvable by the Gateway

Notice that the service was added, a default policy allowing anonymous access to the service was created and a warning to that effect was produced, Figure 10-9.

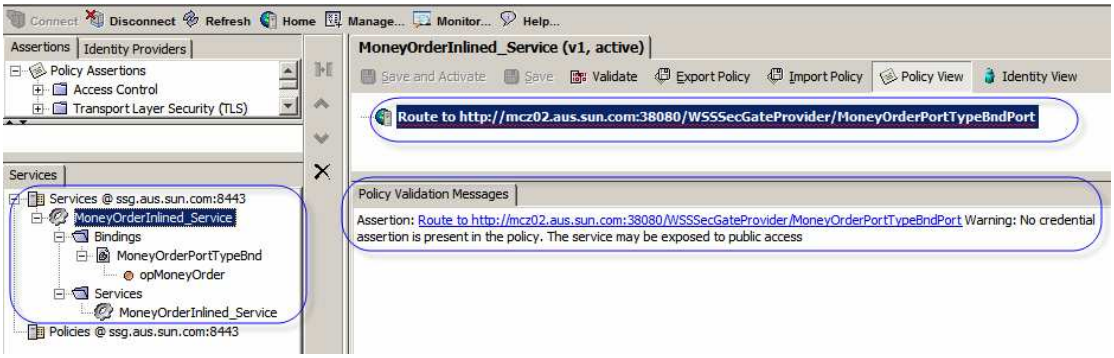

Figure 10-9 Anonymous web service was configured

Note, too, attributes of the service, displayed in the service name Tab – MoneyOrderInlined_Service (v1, active). Just from this string one surmises that the SecureSpan Gateway can a) version services, and b) activate/deactivate service (enable/disable access) per service version.

By default the SSG expects the service request to use the servlet context of /ssg/soap. Clearly, only one of these can be defined at a time. To allow support services with different servlet contexts and additional configuration step is required. In the SSG Manager UI expand the node tree under the Services Tab, Services @ ssg.aus.sun.com:8443 (or the FQDN of your Gateway instance), right-click on the name of the service and choose Service Properties, as shown in Figure 10-10.

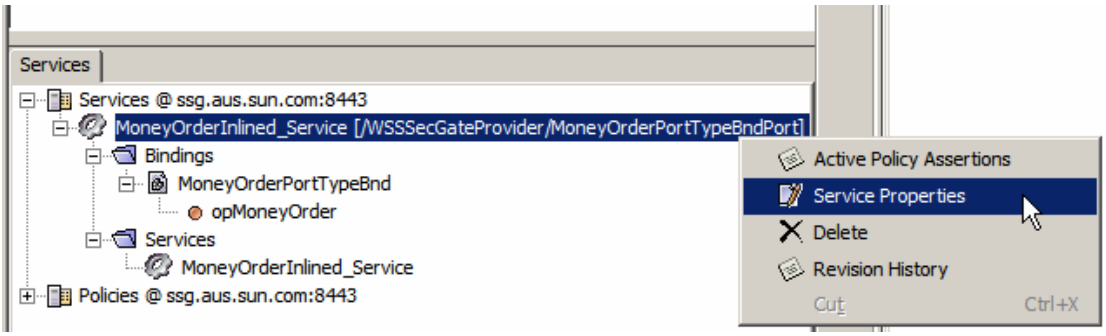

Figure 10-10 Edit service properties for the new service

Click the HTTP Tab, check the "Custom resolution path" radio button and paste the entire servlet context configured in the WSCSecGateConsumer SOAP/HTTP Web Service External System container used to deploy the consumer. Figure 10-11 illustrates the SSG Service Properties HTTP Tab where the Custom resolution path is configured. Figure 10-12 shows the source of that servlet context, for cross reference.

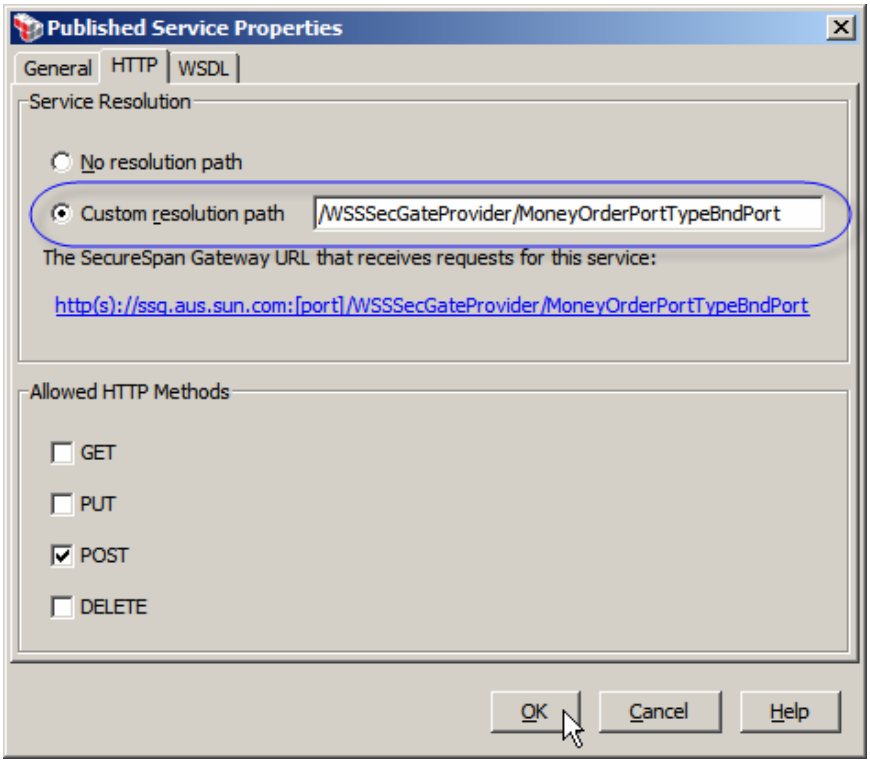

Figure 10-11 Configuring Custom resolution path through the SSG Service Properties

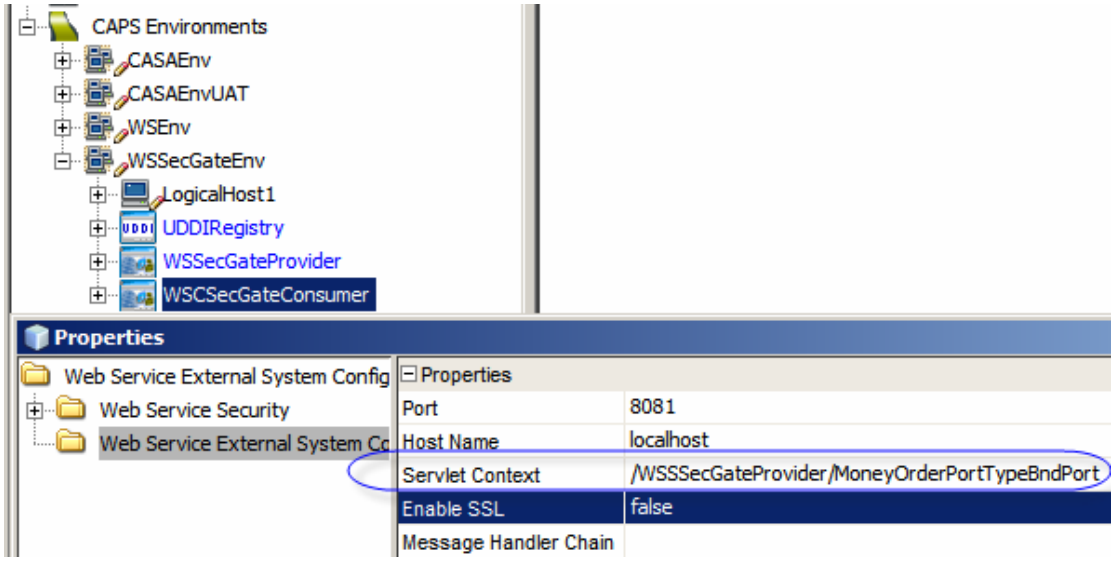

Figure 10-12 Source of the servlet context

Let's switch to TCP Mon and click the "Remove All" button in the TCP Mon UI to clear old requests and responses. The reason for this is that if the HTTP Keep-alive is set, as it seems to be by default in Java CAPS, multiple requests and responses will show up in the same windows, making it hard to figure out which response belongs to which request.

The route to the service has been configured in the gateway. Let's submit a message to the consumer and see message exchange. Figure 10-13 shows it in TCP Mon.

```
POST /WSSSecGateProvider/MoneyOrderPortTypeBndPort HTTP/1.1
Content-Type: text/xml; charset=utf-8
Accept: text/xml, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Length: 1187
SOAPAction: "urn:Sun:Michael:Czapski:WSDL:MoneyOrder/MoneyOrderPortType/opMoneyOrder"
User-Agent: Java/1.6.0_10
Host: ssg.aus.sun.com:8081
Connection: keep-alive

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:enc="http://schema
   <env:Body>
      <ns0:MoneyOrderReq xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
         <tns:OrderDetails>
            <tns:dateTime>2009-02-18T10:42:44.687Z</tns:dateTime>
            <tns:seq>234</tns:seq>
            <tns:total>1000</tns:total>
         </tns:OrderDetails>
```

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Content-Length: 664
Date: Wed, 18 Feb 2009 19:32:21 GMT

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:enc="http://schema
   <env:Body>
      <ns0:MoneyOrderRes xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
         <tns:OrderDetails>
            <tns:dateTime>2009-02-18T10:42:44.687Z</tns:dateTime>
            <tns:seq>234</tns:seq>
            <tns:total>1000</tns:total>
            <tns:orderStatus>true</tns:orderStatus>
         </tns:OrderDetails>
         <tns:SenderDetails>
            <tns:customerName>Jan Kowalski</tns:customerName>
         </tns:SenderDetails>
      </ns0:MoneyOrderRes>
   </env:Body></env:Envelope>
```

Figure 10-13 SOAP Re1quest and Response on-the-wire, with SSG mediation

As we can see the addition of the SecureSpan Gateway to the solution and defintion of a plain routing policy does not add security or cause issues for non-secure service consumers.

Let's add a simple security policy to see what will happen. Let's switch to the SSG Manager, select the MoneyOrederInlined_Service service so that it is displayed in the right-hand pane, in the Assertions Tab expand Policy Assertions->Access Control and drag the Encrypted Username Token policy assertion onto the canvas and drop it above the Route policy. Figure 10-14 illustrates the process.

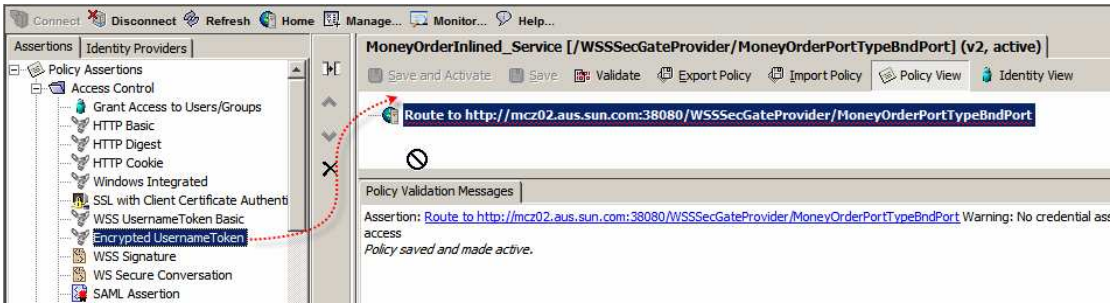

Figure 10-14 Add Encrypted username Token policy assertion

With the assertion added, click the Save and Activate "button" as shown in Figure 10-15. This will propagate changes to the Gateway. Next request, which is submitted to the gateway, will be required to contain the Username Token SOAP Headers.

Figure 10-15 Save and Activate new policy

Let's test the service to see what response we will get, since the consumer does not provide Username Token.

The Gateway returned SOAP Fault saying "Authentication Required" as shown in Figure 10-16. We modified the policy for the service at the Gateway. This policy now requires the consumer to provide a Username Token. We did not have to either modify the service or build and deploy it. The policy was dynamically applied and took effect for the very next request.

```
HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1
L7-Policy-URL: http://ssg.aus.sun.com:8081/ssg/policy/disco?serviceoid=425985
Content-Type: text/xml;charset=utf-8
Content-Length: 604
Date: Wed, 18 Feb 2009 19:42:26 GMT
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
         <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>Policy Falsified</faultstring>
            <faultactor>http://ssg.aus.sun.com:8081/WSSSecGateProvider/MoneyOrderPortTypeBndPort</faultactor>
            <detail>
               <l7:policyResult status="Authentication Required" xmlns:l7="http://www.layer7tech.com/ws/policy/fault"/>
            </detail>
         </soapenv:Fault>
      </soapenv:Body>
   </soapenv:Envelope>
```

Figure 10-16 SOAP Fault with Authentication Required from the Gateway

11. Add a VPN Client to the Mix

Let's add the SecureSpan VPN Client to the mix. The schematic of the solution is shown in Figure 11-1.

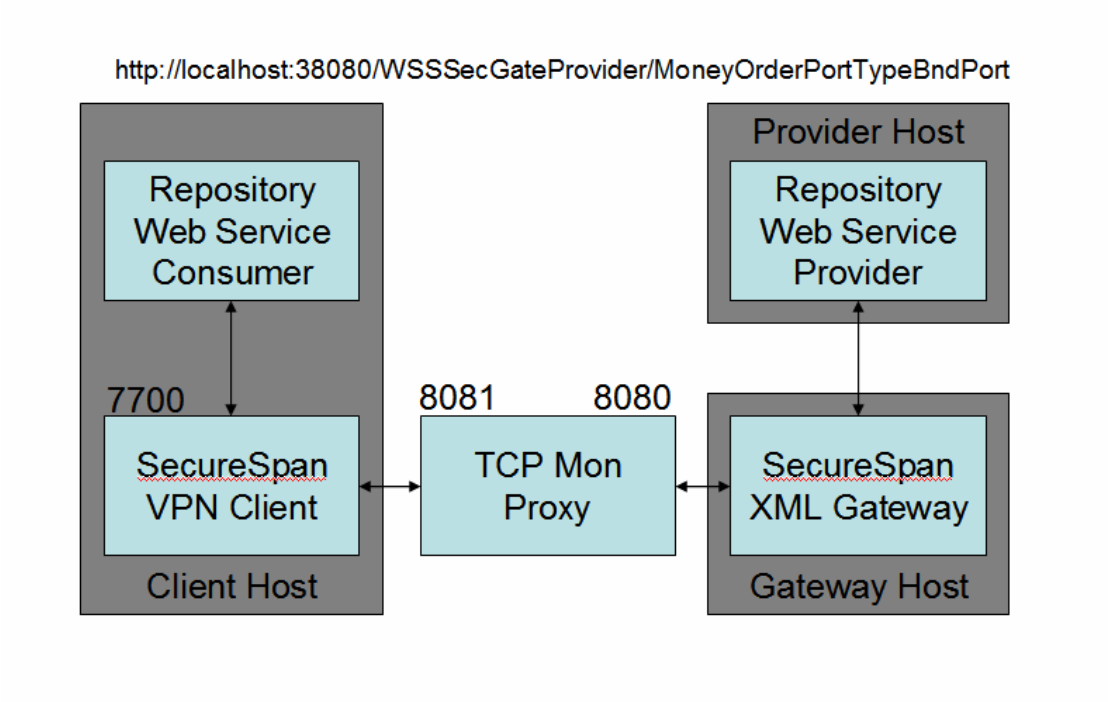

Figure 11-1 Solution with the SecureSpan VPN CLient

We will configure the WSCSecGateConsumer SOAP/HTTP Web Service External System container to forward requests to the VPN Client as though it was the host actually hosting the service. The VPN Client is expected to run on locahost and by default listens on port 7700. The port can be changed. The host can not.

If you have not done so yet, follow the steps in section 15.3, "Install and Configure the SecureSpan VPN Client", before continuing with this section.

We need to start and/or configure the components shown in Figure 11-2, with all but two already configured.

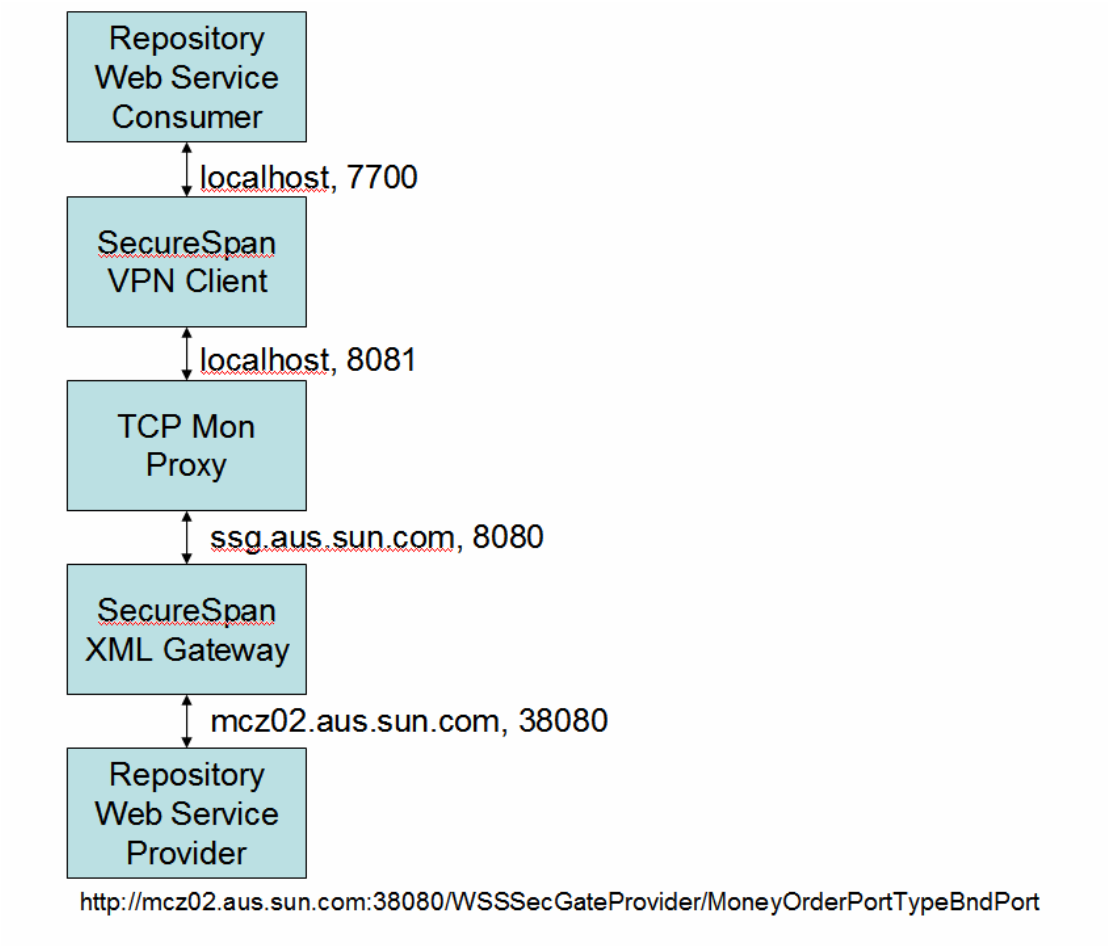

Figure 11-2 Host and port assignments

Let's start the SecureSpan VPN Client, click on the Properties button, switch to Network Tab.

We need to configure the port number to which the VPN Client will connect, the "Gateway custom port" and the "Gateway IP Address". The last two we will modify only because we have the TCP Mon as an intermediary. If the VPN Client was connecting directly to the Gateway, as would normally be the case, we would keep the "Gateway uses standard ports" and "Lookup IP Address in DNS" properties checked – the default.

Check the "Gateway Requires Custom Ports" checkbox and provide the port number on which the TCP Mon is listening, for me 8081, as shown in Figure 11-3. In normal circumstances we would not do this as we would not be running the TCP Mon to snoop on the wire. At any rate, if the SSG was running on a different port this is where we would specify this different port.

If using an intermediary like a TCP Mon, as we are doing in this exercise, we need to tell the VPN Client to which host to forward requests. Click the "Use the following IP addresses" radio button, click Add and enter 127.0.0.1, for localhost. This will redirect requests to localhost:8081, which is where the TCP Mon is listening for requests. If we did not use the TCP Mon we would not need to make this modification.

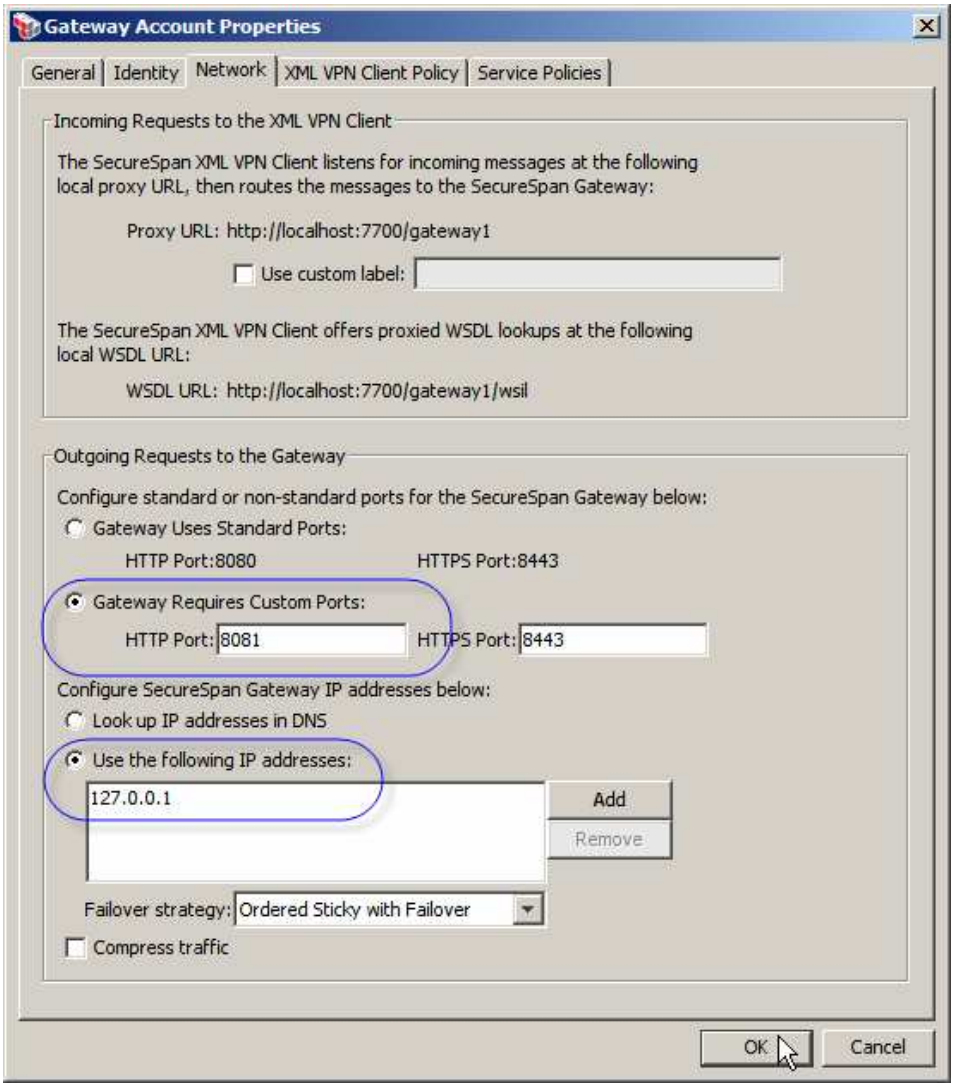

Figure 11-3 Specify custom port and custom label

Let's now modify the WSCSecGateConsumer SOAP/HTTP Web Service External System so that it points to the VPN Client rather then to the TCP Mon, as it currently does if you followed the steps thus far. Figure 11-4 illustrates the properties. Build and deploy the project.

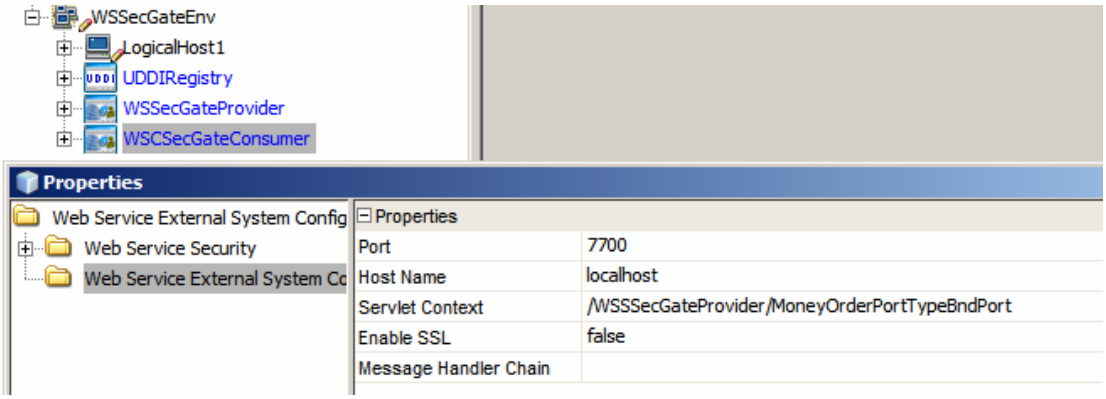

Figure 11-4 Point the consumer at the VPN Client

Let's ask the VPN Client to show what it sees. Open up the SecureSpan VPN Client UI by clicking on the icon in the System Tray, pull down the Windows menu and select the "Recent Message Traffic" option.

Now that all the components of the infrastructure are running, let's Remove All from the TCP Mon, submit a test message to the appropriate JMS Queue using the Enterprise Manager, and observe what is exchanged between the VPN Client and the SSG.

The request, and the response, indicate that all was well, see Figure 11-5.

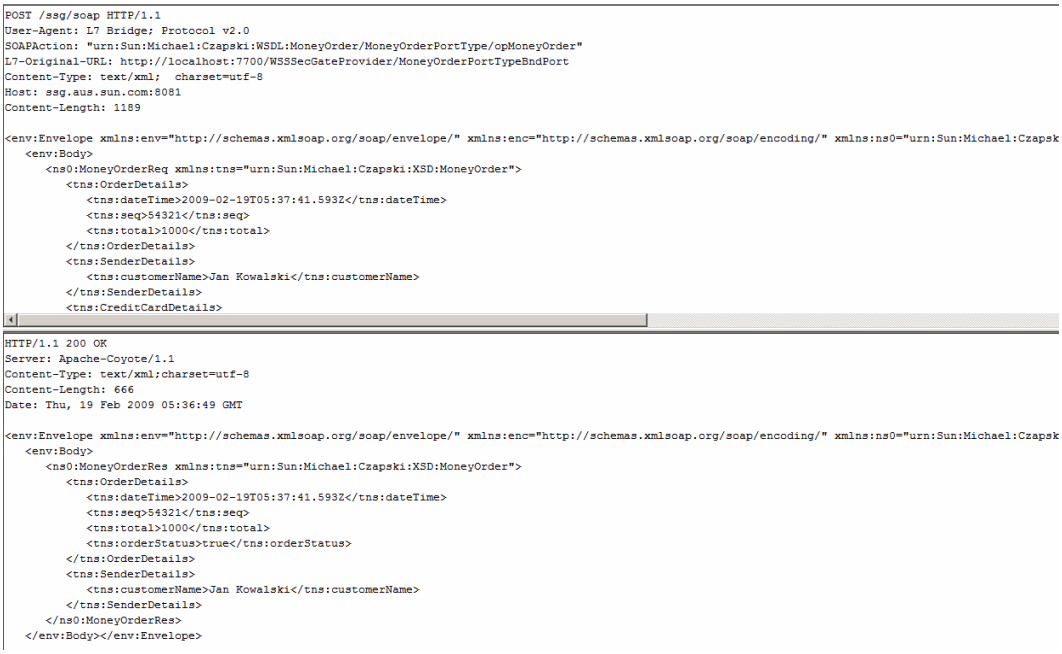

Figure 11-5 Success

Switch to the VPN Client's Recent Message Traffic window and take a look at the messages that were exchanged. Figure 11-6 shows one of the messages in the right-hand window and the message exchange in the left-hand window.

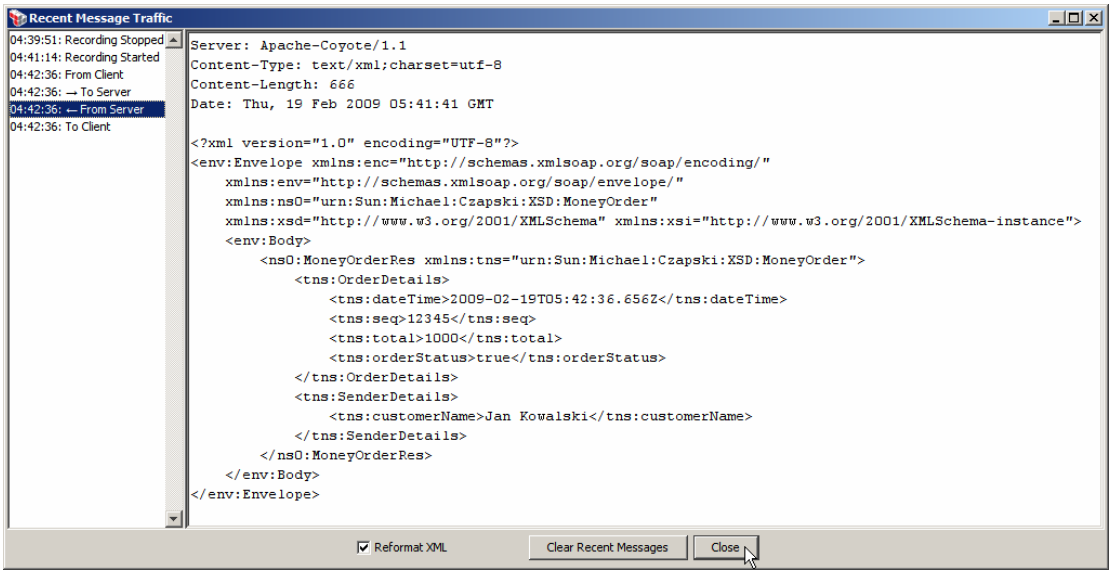

Figure 11-6 One of the messages (from server) in the Recent Message Traffic VPN Client window

We have a functioning configuration with consumer requests being passed off to the SecureSpan VPN Client, from there to the TCP Mon, from there to the SecureSpan XML Gateway (on a separate host), from there to the service provider. If all is well, responses travel in the opposite direction.

Consider what would happen if the service was not defined in the SSG or the consumer's request did not contain security information required by the gateway. As you would expect the Gateway would reject the request and send a SOAP Fault message back to the consumer. We have seen this behaviour earlier, when submitting a request from the consumer directly to the Gateway, when it did not have the service defined.

12. Miscellaneous Gateway Service

12.1 *Logging, Auditing and Alerting*

The SecureSpan Gateway supports logging and auditing, which includes configurable logging sinks and other facilities.

To make it easier for me to see what is going on, and to show it in the Note, I downloaded and installed a Kiwi Syslog Server for Windows as an evaluation – see http://www.kiwisyslog.com/. I configured the Gateway to log all it knows about to that destination. Configuration of logging sinks is accessed through the Manage drop-down, see Figure 12-1.

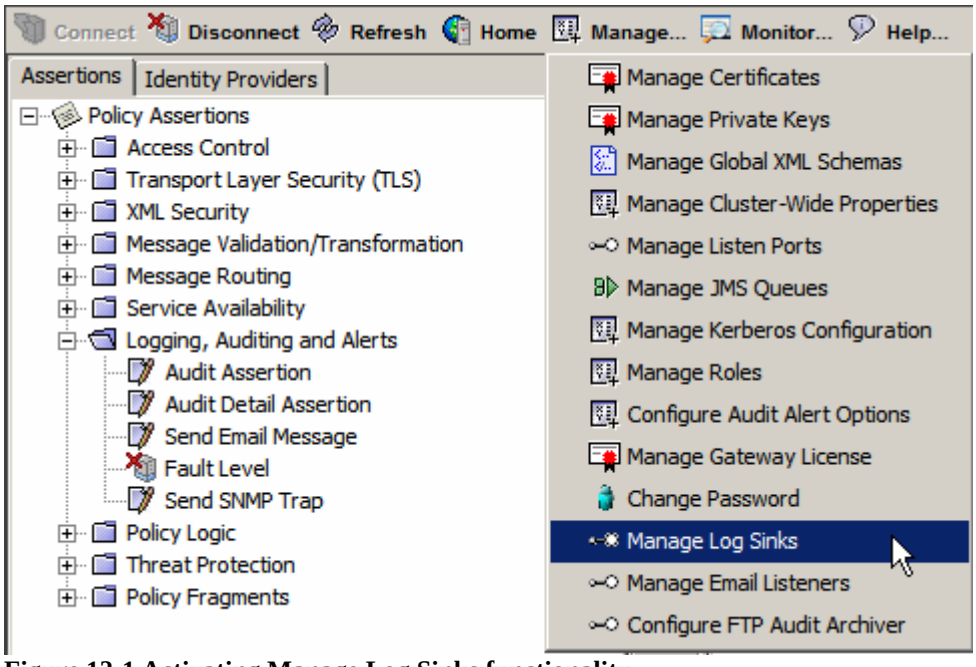

Figure 12-1 Activating Manage Log Sinks functionality

The Syslog configuration I used logs all there is to be logged, see Figure 12-2. Syslog Settings I use are shown in Figure 12-3. Note the FQDN of the Syslog host.

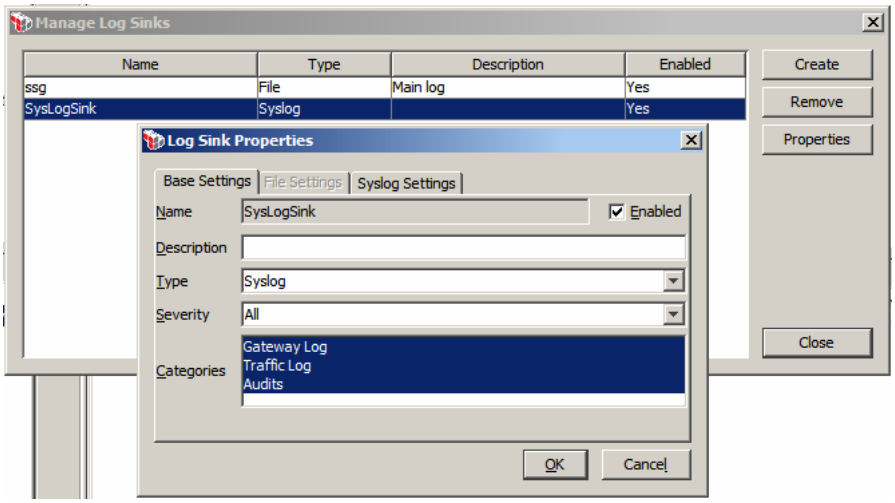

Figure 12-2 Configuration of the Syslog sink, Base Settings

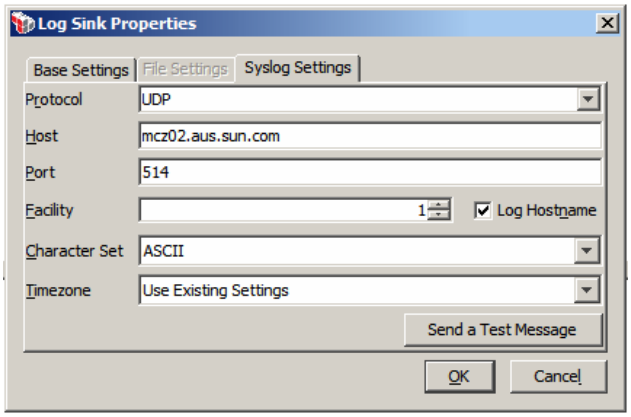

Figure 12-3 Configuration of the Syslog Settings

All this is to enable me to discuss and illustrate logging and auditing directives that can be added to the policies in the Gateway.

Let's drag the Audit Assertion to the Policy editor window then double-click the policy assertion and check the "Save request XML" and "Save response XML", see Figure 12-4. Note other logging and auditing assertions which you can add to the policy.

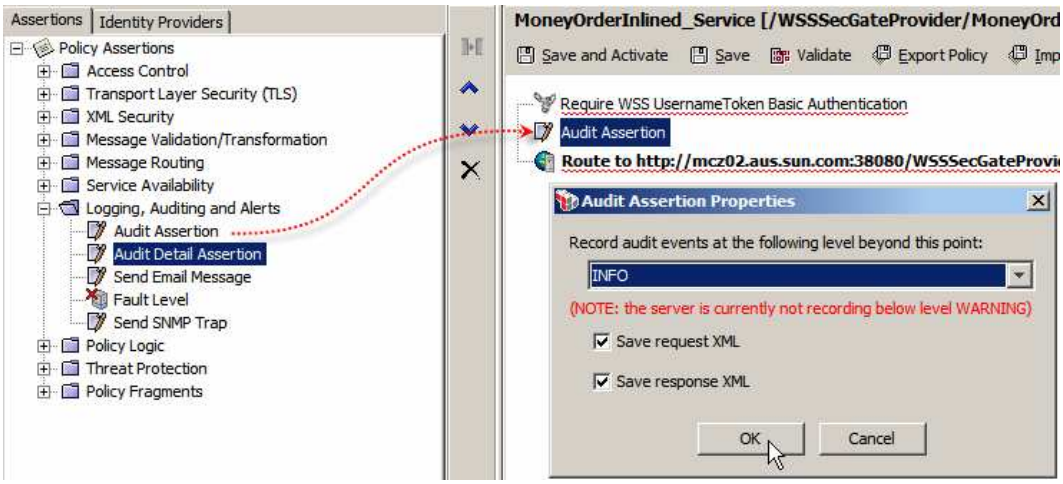

Figure 12-4 Add and configure Audit Assertion

Let's exercise the solution by submitting a JMS message, as we have done before. Figure 12-5 show the Kiwi Syslog Server window with the messages sent out from the Gateway.

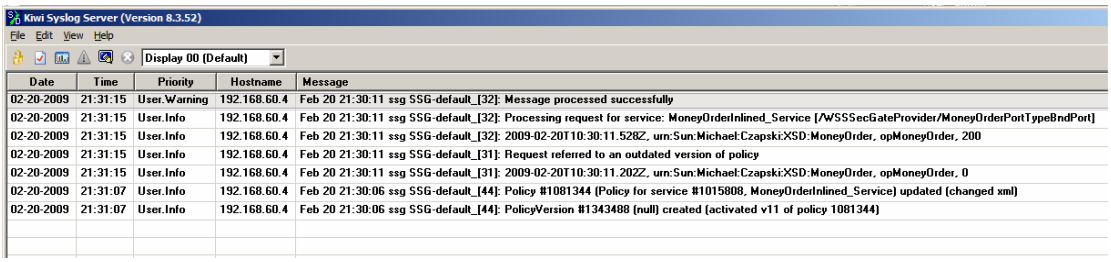

Figure 12-5 Syslog output in the Kiwi Syslog Server window

Figure 12-6 shows the messages in a readable form. Note, starting at the bottom and going toward the top, that a new policy was activated, VPN Client request was rejected due to outdated policy then the re-submitted request was processed.

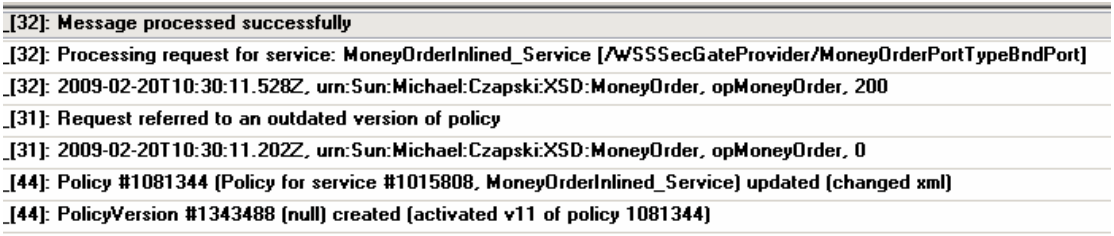

Figure 12-6 Message detail

This is something you can watch to see what is going on in a brief format.

The Audit Assertion, which we added to the policy and configured to save request and response XML, causes audit events to be written to the audit database. To access audit events, drop down the SSG Manager's Monitor … drop down and select Gateway Audit Events, as shown in Figure 12-7. A new window will open.

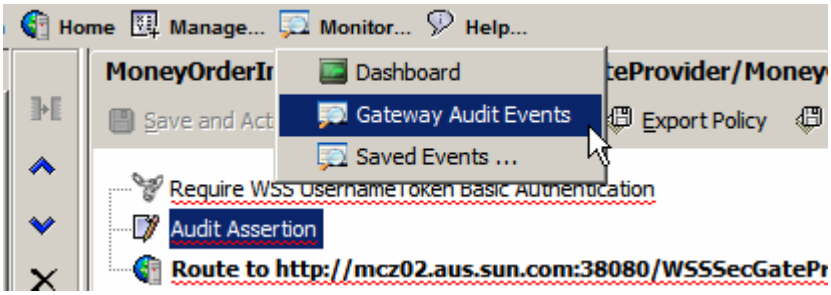

Figure 12-7 Choose Gateway Audit Events

For each policy evaluation that has Audit Assertion, there will be an audit entry. If save request and response XML is selected the request and response will be saved. Figure 12-8 shows the request XML for the service invocation.

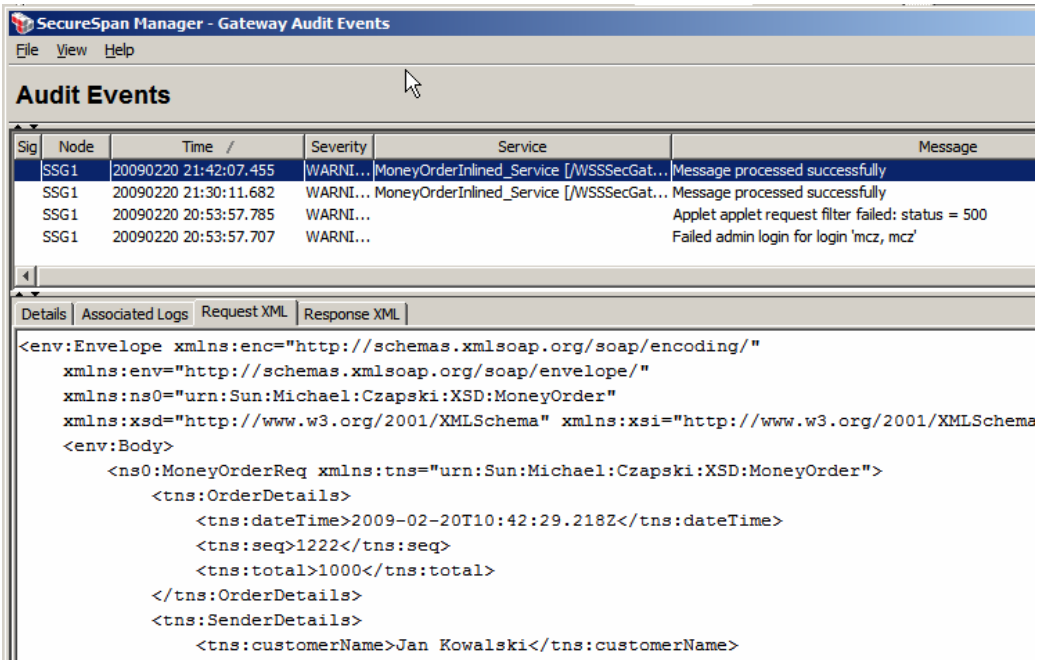

Figure 12-8 Request XML in the Gateway Audit Event window

Let's revert to a policy consisting solely of the "Route to" assertion then add "Fault Level" and "Send Email Message" assertions from the "Logging, Auditing and Alerts" group. The policy looks like that shown in Figure 12-9.

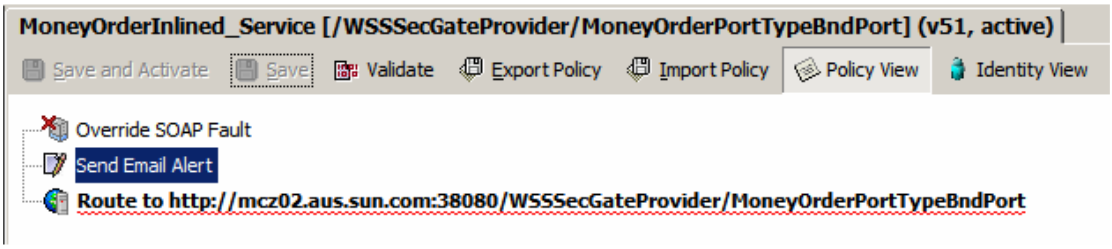

Figure 12-9 Policy with Fault Level and Send Email Alert assertions.

Right-click the Override SOAP Fault assertion and modify its properties to "Full Detail", meaning provide verbose fault information. Right-click the "Send Email Alert" and configure the properties. At minimum specify the host and port for the SMTP Server which will receive email messages from the gateway, the recipient address, and perhaps a constant string in the body of the message. The message body in Figure 12-10 uses a number of context variables to obtain and embed certain information about the request and the environment in which it is being processed.

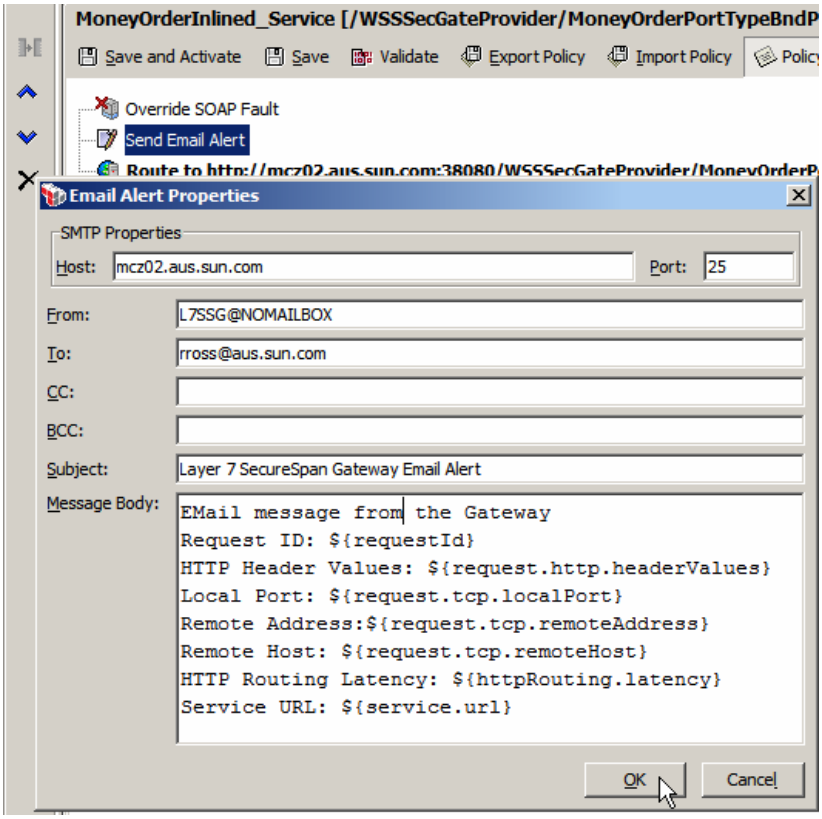

Figure 12-10 Configure email alert properties

"Save and Activate" the policy, submit a JMS message and check the email. An alert message should have been sent. I received an alert shown in Figure 12-11.

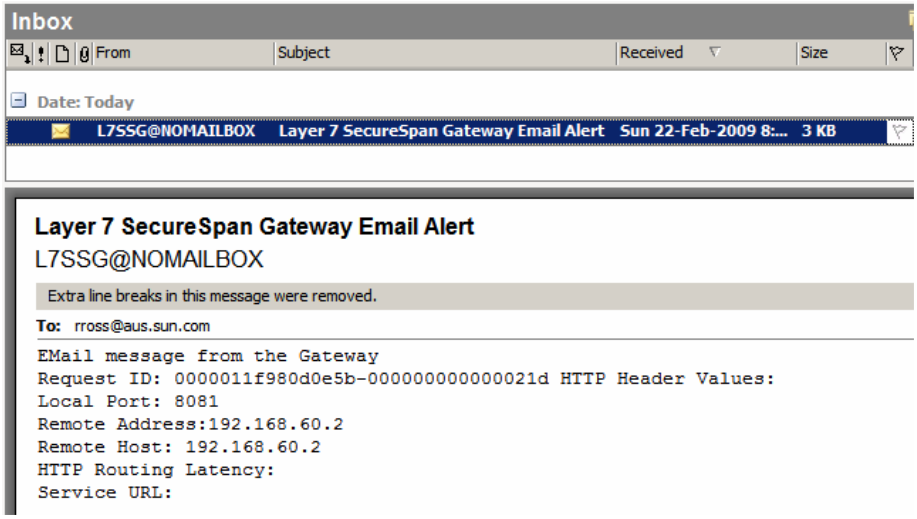

Figure 12-11 Email alert

There are a number of options for logging and auditing of policy evaluation and request and response processing. Email notifications can be sent to alert operational staff of events of interest, both normal and abnormal. Consult the vendor documentation for information on how to configure these facilities. Logging, auditing and alert assertions can be added at various places in a policy to see which policy alternatives are evaluated, to log information about requests and responses, and to provide audit trial of service invocation.

12.2 *XML Schema Validation*

In addition to security services, digital signatures, encryption authentication and suchlike, the Gateway can be employed to deal with certain kinds of XML-borne threats like XML Injection, malformed XML, buffer overflow, deep recursion and similar. One of the means of ensuring that rouge SOAP requests and responses do not make their way too deeply into the infrastructure is XML Schema validation.

Let's revert to the policy version in which only the "Route to …" assertion exists, expand Policy Assertions->Threat Protection, drag the Validate XML Schema assertion to the canvas above the Route To assertion and click the "Extract from WSDL" button in the dialogue box that appears. See Figure 12-12.

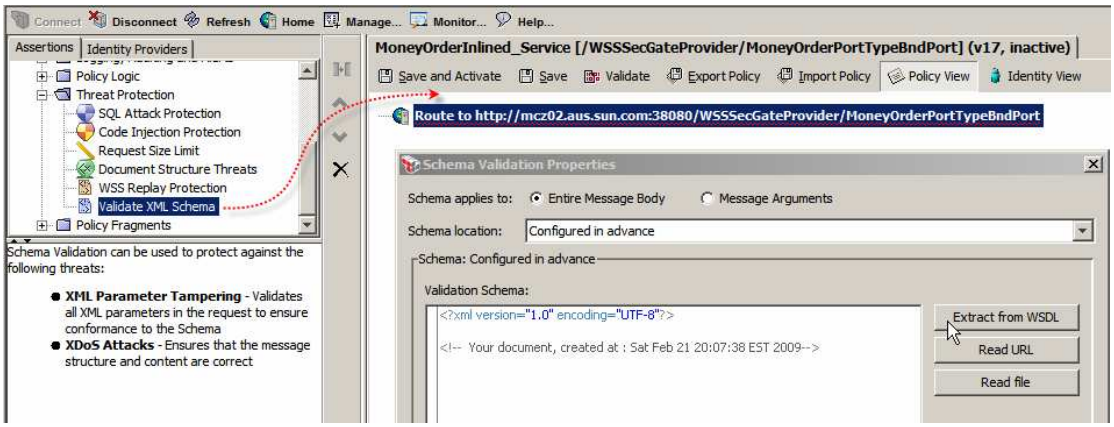

Figure 12-12 Get hold of the XML Schema to use for validation

Choose the entire schema, Figure 12-13, and click OK. We could have chosen the request only or the response only.

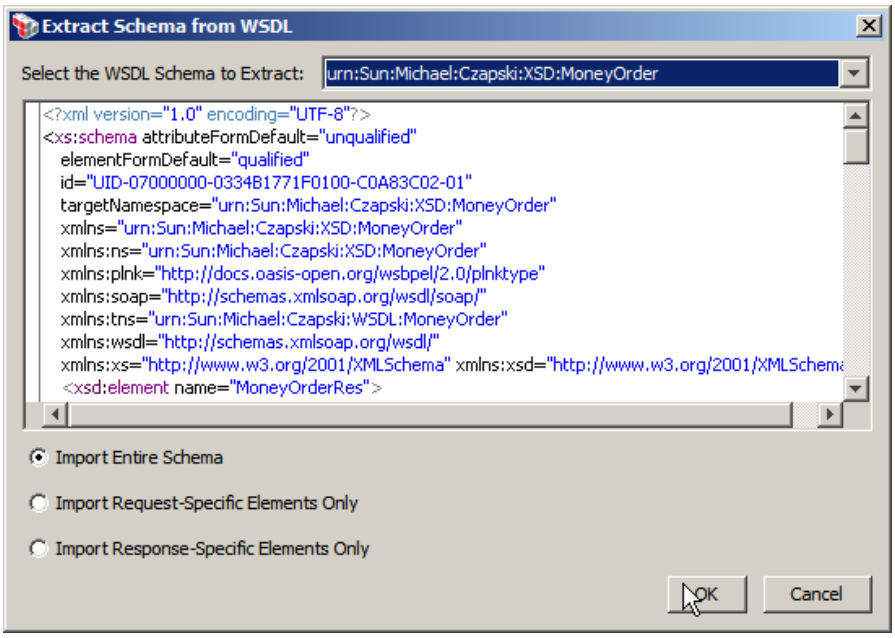

Figure 12-13 Choose entire schema

By placing the Validate XML Schema assertion before the Route to assertion we are getting the gateway to validate the request before it is submitted to the back-end service. Let's also add the Validate XML Schema after the Route to assertion to have

the response validated before it is passed on. Use the same steps as before making sure to place the assertion after the Rout to assertion. Click the “Save and Activate” button, submit the JMS message and observe request and response messages in the TCP Mon, in the VPN Client’s Recent Message Traffic window and in the SSG Manager’s Monitor Audit Events window. Notice that noting was added to the request and nothing was added to the response. The request was processed successfully. No new audit events in the Gateway Audit Events window.

Let’s add a digital signature assertion to break schema conformance. Let’s expand the Policy Assertions->XML Security node tree, drag Sign Request Element to the Policy editor window and drop it above the Validate Message Schema assertion. Double-click the new assertion, click on the Web Service Operations ->opMoneyOrder operation and click on the <ns:SenderDetails> node. This will cause the selected element to be digitally signed. In order to successfully sign a message or a part of a message, a WSS Signature Access Control assertion has to be added before the Sign Request Element assertion. Let’s add that assertion, Figure 12-14. Click the “Save and Activate” button. Submit a JMS message and observe the results.

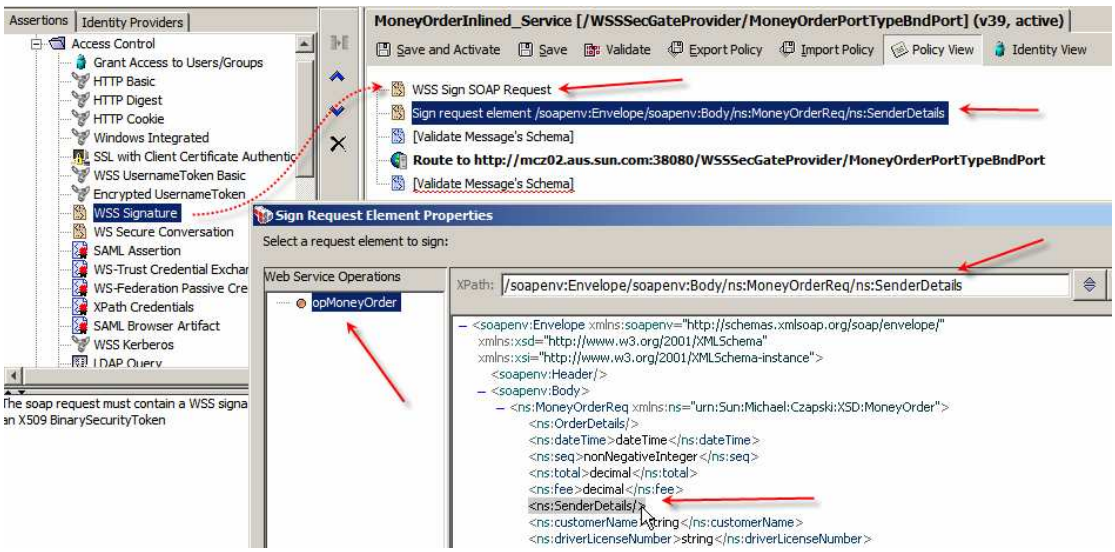

Figure 12-14 Choose request element to sign

Request processing failed. The offending fragment of the request in TCP Mon is highlighted in Figure 12-15.

```
<tns:OrderDetails>
   <tns:dateTime>2009-02-21T09:39:26.562Z</tns:dateTime>
   <tns:seq>121</tns:seq>
   <tns:total>1000</tns:total>
</tns:OrderDetails>
<tns:SenderDetails
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
      wsu:Id="SenderDetails-1-412ec2a6c9f900dc4c11cbc713e4ccd5">
   <tns:customerName>Jan Kowalski</tns:customerName>
</tns:SenderDetails>
<tns:CreditCardDetails>
   <tns:cardType>American Express</tns:cardType>
   <tns:nameOnCard>Jan Kowalski</tns:nameOnCard>
   <tns:cardNumber>1111-111111-111111</tns:cardNumber>
   <tns:securityCode>1111</tns:securityCode>
   <tns:validUntil>10/12</tns:validUntil>
</tns:CreditCardDetails>
<tns:ReceipientDetails>
   <tns:familyName>Smith</tns:familyName>
```

Figure 12-15 Additional attributes on the signed element in the request body

The response, in TCP Mon, says “Bad Request”, see Figure 12-16.

```
<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
         <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>Policy Falsified</faultstring>
            <faultactor>http://127.0.0.1:8081/ssg/soap</faultactor>
            <detail>
               <l7:policyResult status="Bad Request" xmlns:l7="http://www.layer7tech.com/ws/policy/fault"/>
            </detail>
         </soapenv:Fault>
      </soapenv:Body>
   </soapenv:Envelope>
```

Figure 12-16 Bad Request

The message is pretty generic and pretty meaningless. Because I added a sign element assertion I suspect that this is what caused the problem, but was it? Let’s add a “Logging, Auditing and Alerts” -> “Fault Level” assertion at the beginning of the policy, double-click on the assertion, change fault level to “Full Details” from the drop down menu, click OK, click “Save and Activate” and submit another JMS message to test the outcome.

The SOAP Fault in the TCP Mon window, reformatted for better readability, is quite clear in that it says exactly what the issue is - Figure 12-17. Attribute wsu:Id, which is added to the request body to identify the part of the message which was signed, was not catered for by the original request schema. Whatever the reason, the request does not conform to the schema and is rejected with a SOAP Fault being returned to the sender.

```
<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
         <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>Policy Falsified</faultstring>
            <faultactor>http://127.0.0.1:8081/ssg/soap</faultactor>
            <detail>
               <l7:policyResult status="Bad Request"
                        xmlns:l7="http://www.layer7tech.com/ws/policy/fault"
                        xmlns:l7p="http://www.layer7tech.com/ws/policy">
                  <l7:assertionResult assertion="l7p:FaultLevel" status="No Error"/>
                  <l7:assertionResult assertion="l7p:Integrity" status="No Error"/>
                  <l7:assertionResult
                        assertion="l7p:RequestWssX509Cert" status="No Error">
                     <l7:detailMessage id="4805">Certificate loaded as principal credential
                        for CN:ssgcli</l7:detailMessage>
                  </l7:assertionResult>
                  <l7:assertionResult assertion="l7p:SchemaValidation" status="Bad Request">
                     <l7:detailMessage id="5600">Validating request document</l7:detailMessage>
                     <l7:detailMessage id="5604">Schema validation failure: org.xml.sax.SAXParseException:
                              cvc-complex-type.3.2.2: Attribute 'wsu:Id' is not allowed to appear
                              in element 'tns:SenderDetails'.</l7:detailMessage>
                  </l7:assertionResult>
               </l7:policyResult>
            </detail>
         </soapenv:Fault>
      </soapenv:Body>
   </soapenv:Envelope>
```

Figure 12-17 Schema validation error

Schema validation can be added for requests and responses. Of specific note is the Fault Level assertion, which help identify the issues by providing greater level of detail in the SOAP Fault returned to the consumer.

12.3 *Policy Versioning*

Let's briefly explore policy versioning. The policy consists of a single Route to assertion, as shown in Figure 12-18. In my case the policy version is v43. I modified the policy a fair bit as I was working my way through various examples.

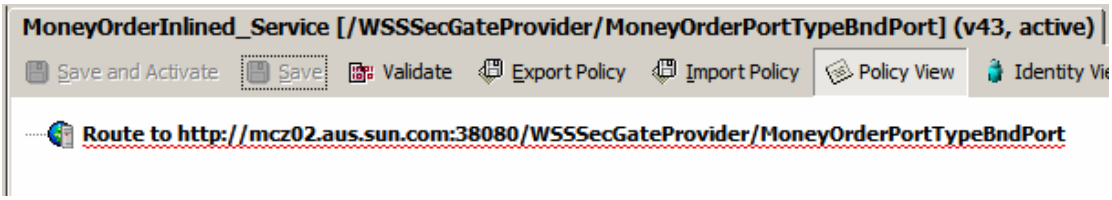

Figure 12-18 Policy with the Route to assertion

Let's add an "Encrypted Username Token" "Access Control" assertion, and a "Fault Level" "Logging, Auditing and Alerts" assertion above the "Route to" policy assertion, then click "Save and Activate". Figure 12-19 shows the new policy. Policy version went up to 44.

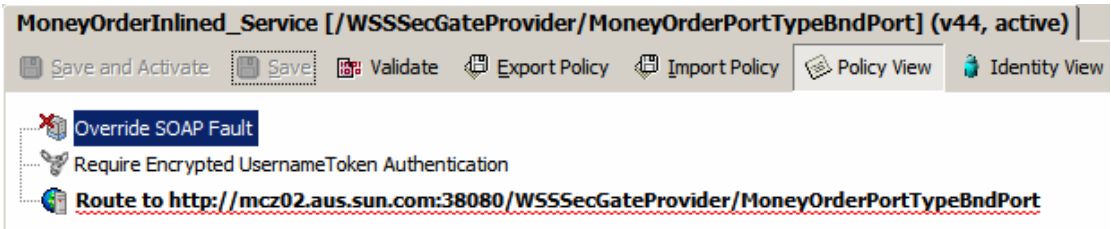

Figure 12-19 Modified policy

Right click on the service name in the Services Tab and choose Revision History, as shown in Figure 12-20.

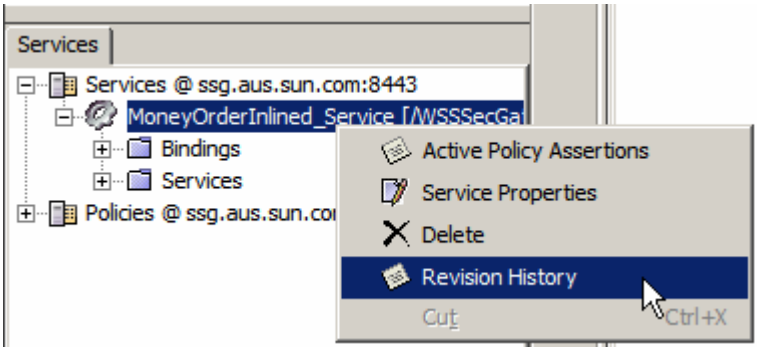

Figure 12-20 Choose Revision History

Policy revision 44, shown in Revision History, Figure 12-21, is active and looks like the policy that we just configured. If we choose version 43 of the policy, Figure 12-22, and click "Set Active", the old policy will be activated. Note that this policy does not require "Encrypted Username Token".

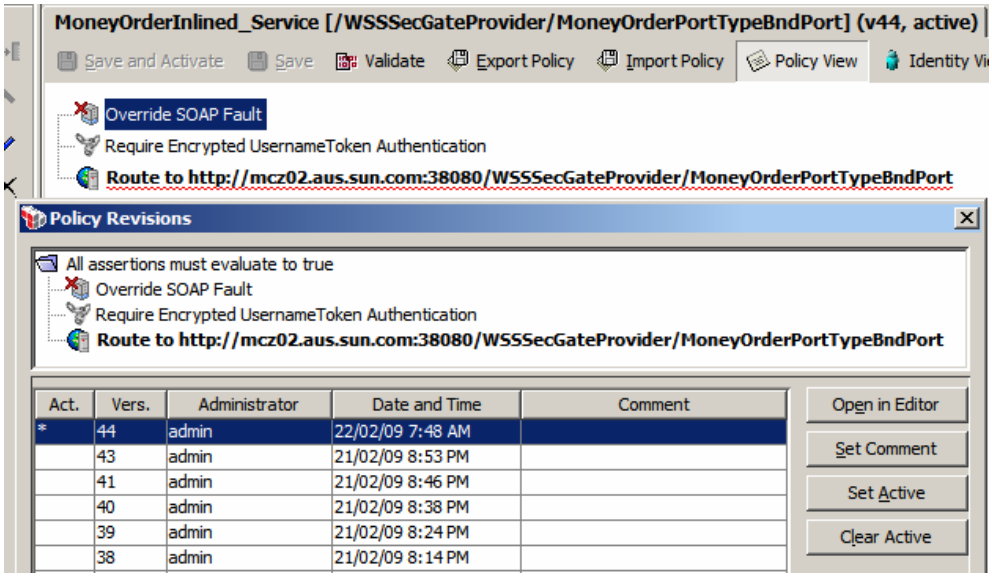

Figure 12-21 Version 44, Active policy

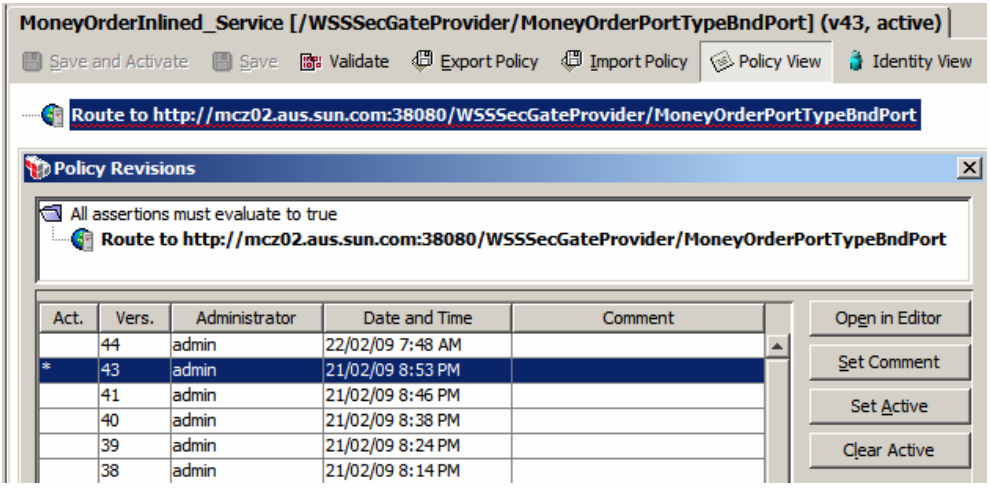

Figure 12-22 Version 43 policy activated

Clicking Clear Active, when the active policy is enabled, disables the policy completely, see Figure 12-23. Attempt at invocation of the service with no active policy will fail with a SOAP Fault, Figure 12-24, being returned to the consumer.

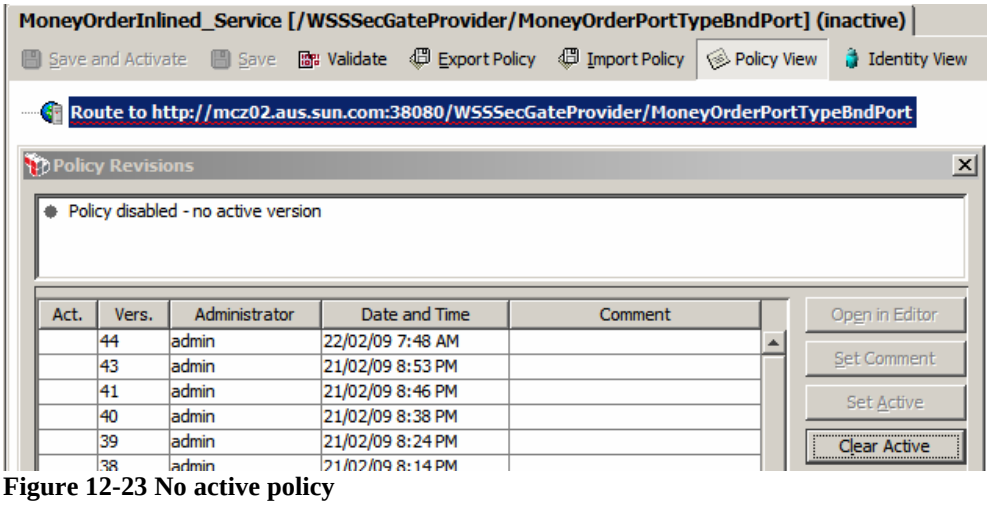

Figure 12-23 No active policy

```
HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Content-Length: 564
Date: Sat, 21 Feb 2009 20:56:00 GMT
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
         <soapenv:Fault>
            <faultcode>soapenv:Server</faultcode>
            <faultstring>Policy Falsified</faultstring>
            <faultactor>http://ssg.aus.sun.com:8081/ssg/soap</faultactor>
            <detail>
               <l7:policyResult
                        status="Assertion Falsified"
                        xmlns:l7="http://www.layer7tech.com/ws/policy/fault"/>
            </detail>
         </soapenv:Fault>
      </soapenv:Body>
   </soapenv:Envelope>
```

Figure 12-24 SOAP Fault resulting from an attempt to invoke a service with no active policy

Let's activate policy version 44, "Route to …." assertion only, for future experiments.

Policies can be developed over time. Different versions of the policy can be active at different times. Services can be disabled by making sure that no policy is active for the service. This is a part of what some call "Runtime Service Governance".

12.4 *Service Availability Policies*

Service Availability assertions provide for specification of such Quality of Service and Availability policies as Time/Day Availability, IP Address Range, Throughput Quotas and Rate Limits, Figure 12-25. Judicious use of one or more of these assertions, for all services the Gateway handles, can provide for enforcement of Service Level Agreements. It can do so by preventing access to specific services from specific addresses and address ranges, limiting access to non-essential services, throttling request processing for non-premier services, regulating bandwidth use by heavy bandwidth users and ensuring higher-priority, premium services receive appropriate resource allocation.

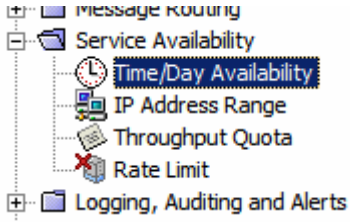

Figure 12-25 Service availability assertions

12.5 *Threat protection*

Supplementing service availability assertions are the Threat Protection assertions. Most notable of these, from the web service security perspective, are Request Size Limit (prevent buffer overflow and denial of service attacks arising out of attempts to process excessively large requests), Document Structure Threats (preventing deliberately malformed XML from overwhelming XM processing machinery and

potentially causing rouge XML to be processed), WSS Reply Protection (preventing intercepted messages from being re-submitted by rogue senders to cause repeated submission of given data (withdraw the same amount from the same account several times, for example) and Validate XML Schema (preventing submission of non-conformant messages potentially breaking request processing machinery).

Figure 12-26 Threat Protection Assertions

13. Using the Gateway to Secure Web Services

In the next few sections we will explore application of various security policies, interaction between the SecureSpan VPN Client and the SecureSpan XML Gateway and SOAP Headers used to convey the security information called for by the specific policies. We will use the infrastructure which we finally have deployed. We will not be modifying either the consumer or the provider. We will be working with the Gateway and will be applying policy changes dynamically. This is what the whole thing is about - being able to secure services without having to refer to a developer or to re-deploy the service provider or the service consumer.

Note that while we have a Java CAPS Repository-based Web Service consumer and Web Service Provider, we could equally well have a JBI-based consumer and provider, an EJB-based consumer and provider or a combination of both. The gateway does not care. As mentioned, the reason I chose Repository-based technology was to demonstrate that sophisticated, secure web services solutions can be built, with the aid of a Gateway, regardless of the capabilities of the technology used to develop providers and consumers.

13.7 *Propagation of Polices to the VPN Client*

The SecureSpan VPN Client can be statically configured with the security policy applicable to a web service protected by the SecureSpan Gateway. The policy can be exported using the SSG Manager and imported into the VPN Client. Naturally, changes to the policy require it to be exported, distributed to the consumers and imported into the VPN Client for use. There are good reasons to use static policies. I will not go into them here.

The SecureSpan VPN Client can also be configured to dynamically obtain the security policy of a service protected by the SecureSpan XML Gateway. This makes policy propagation automatic and transparent to the service consumer. This is what I find so attractive about the VPN Client.

Let’s look at a scenario that demonstrates dynamic policy propagation from the Gateway to the VPN Client.

Let's switch to the SecureSpan VPN Client UI, assumed to be running. Let's open the properties and switch to the Service Policies Tab. If the steps were followed as given in this Note, there should be no policies being cached, see Figure 13-1.

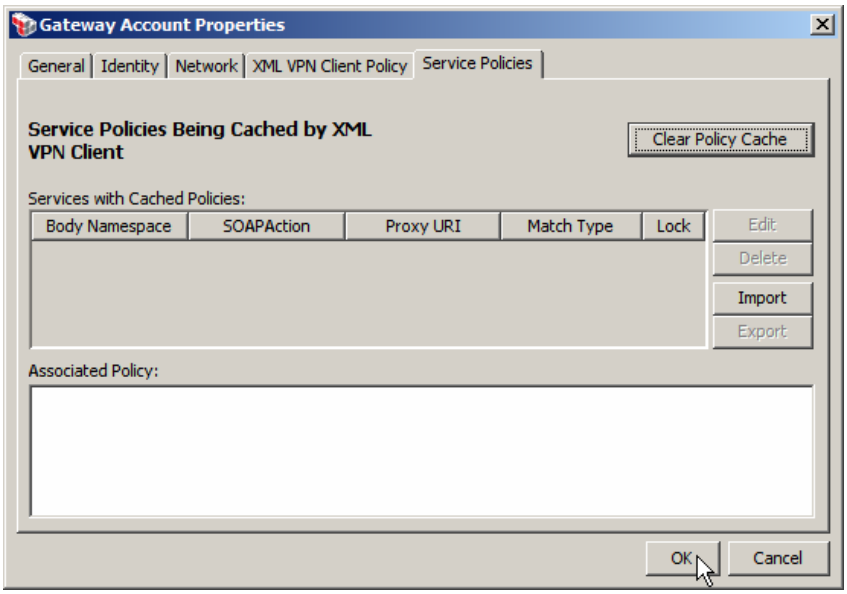

Figure 13-1 No policies cached by the Client

Because the policy defined for the WSSSecGateProvider service protected by the Gateway only contains the routing rule, Figure 13-2, there is no requirement for the client to deal with security decorations either on requests or on responses. The Gateway merely relays requests and responses.

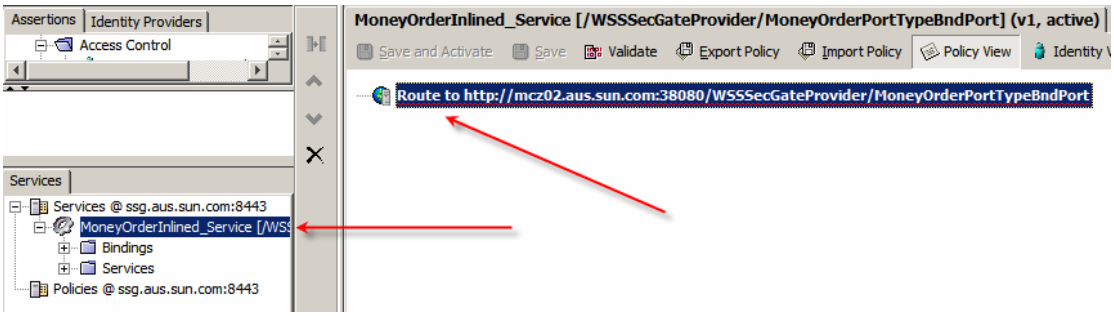

Figure 13-2 Policy with just the routing rule

Let's switch to the SSG Manager and select our service so the policy appears in the right hand pane, as shown in Figure 13-2.

In the top left pane let's expand the Access Control node and look at the various ways in which access control can be imposed over the service. There are quite a number of methods, as shown in Figure 13-3. Let's start simple; drag the "HTTP Basic" assertion form the list onto the right-hand pane and drop it above the route rule. This is shown in Figure 13-3.

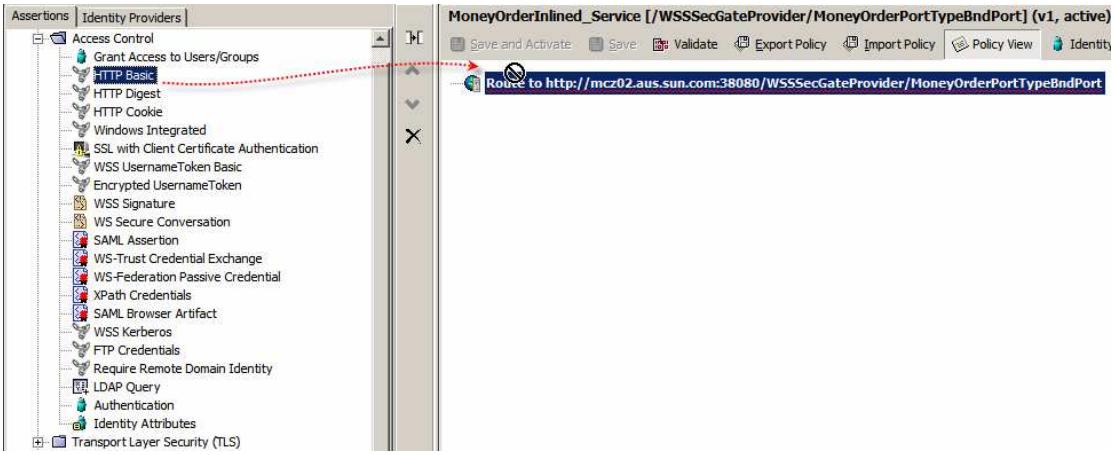

Figure 13-3 Adding HTTP Basic Access Control Assertion to the policy

Let's now "Save and Activate" the changed policy, as illustrated in Figure 13-4.

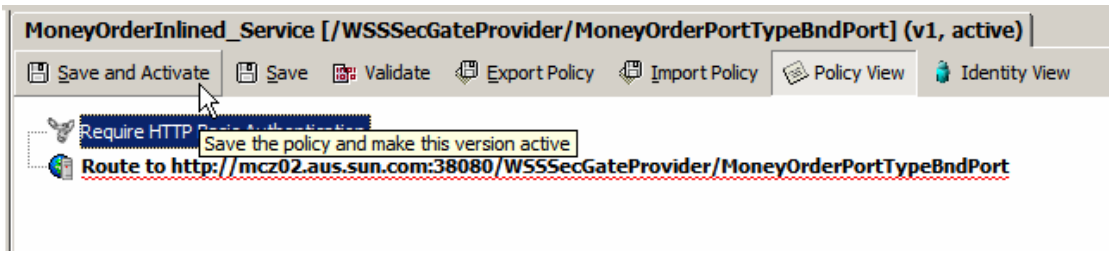

Figure 13-4 Save and Activate the policy

Note that the new version of the policy was created and that a message is warning us that the credentials are collected but not validated - Figure 13-5. I will not go into the identity providers and similar matters. The SecureSpan documentation, Layer 7 personnel and Layer 7 partners can help you configure these things if you decide to buy the product. My purpose here is to demonstrate how dynamic policy change is propagated to the VPN Client and used by the client to invoke the protected service.

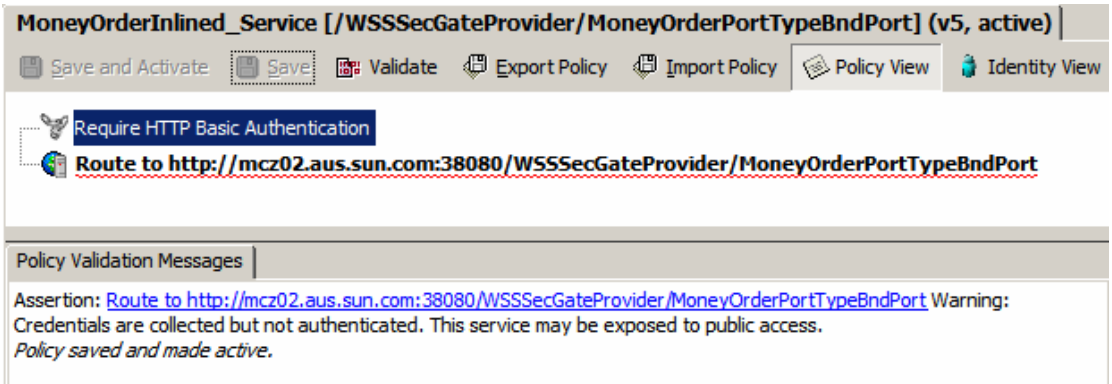

Figure 13-5 Warning about a potential issue with the policy

Let's switch to the TCP Mon and Remove All to clear the deck, as it were.

Let's now switch to the VPN Client, open Properties, switch to Service Policies Tab and confirm that no policy is cached there, then click OK to dismiss the Properties window.

Let's now submit a message to the inbound JMS Queue, much as we have done before, confirm that there is a response in the outbound queue, then switch to TCP Mon and look at the message exchange. Because the services have the HTTP Keep-alive turned on multiple requests and responses appear in the same windows. Look carefully at the requests window to identify where the first request ends and the next begins. The first request, extracted form the TCP Mon window, is shown in Figure 13-6. This is a policy discovery request. The VPN Client asked the Gateway for the policy to use to decorate the actual web service request.

Figure 13-6 First Request – policy discovery

```
POST /ssg/policy/disco HTTP/1.1
Content-Type: text/xml;  charset=utf-8
User-Agent: Jakarta Commons-HttpClient/3.0.1
Host: ssg.aus.sun.com:8081
Transfer-Encoding: chunked

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      <L7a:MessageID
xmlns:L7a="http://www.layer7tech.com/ws/addr">http://www.layer7tech.com/uuid/41dadd6ab
e75aa610f1bf730729b6e17</L7a:MessageID>
      <L7a:ServiceId
xmlns:L7a="http://www.layer7tech.com/ws/addr">1015808</L7a:ServiceId>
   </soap:Header>
   <soap:Body>
      <wsx:GetPolicy
xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/03/mex"></wsx:GetPolicy>
   </soap:Body></soap:Envelope>
```

To make it easy to follow I am showing requests and responses in the order they have been sent. The response to the policy discovery request is shown in Figure 13-7.

Figure 13-7 Policy discovery response – the policy

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml
Transfer-Encoding: chunked
Date: Thu, 19 Feb 2009 08:57:34 GMT

12dc
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" actor="secure_span"
soap:mustUnderstand="1">

         <wsu:Timestamp wsu:Id="Timestamp-4-9863b4d75fb6db844260e7465f132b36">
            <wsu:Created>2009-02-19T08:57:34.701756387Z</wsu:Created>
            <wsu:Expires>2009-02-19T09:02:34.701Z</wsu:Expires>
         </wsu:Timestamp>
         <wsse:BinarySecurityToken EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3" wsu:Id="BinarySecurityToken-0-
b09596a4ffadd42ff0e4d37cb5312889">MIICFjCCAX+gAwIBAgIIAIhey9DpmxQwDQYJKoZIhvcNAQEFBQAw
HzEdMBsGA1UEAwwUcm9vdC5zc2cuYXVzLnN1bi5jb20wHhcNMDkwMjE3MTQzNDM0WhcNMTEwMjE3MTQ0NDM0Wj
AaMRgwFgYDVQQDDA9zc2cuYXVzLnN1bi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJMsd30S24w+
IKgDIpA3liwG+gjwKlV5I5PyvzbtoELVu8jFdhGUoeqv6/i5NjYkFZdhkXrV8J4pCWJJ6yF+jaicoBH3AhXjEf
Ul7xcFKMrXtTq5AFt74ksFVn2Aet8Tbt49ctvzFLnHgDNqPUhtrSSmINXKRVARQ/V9An+4WgovAgMBAAGjYDBe
MAwGA1UdEwEB/wQCMAAwDgYDVR0PAQH/BAQDAgXgMB0GA1UdDgQWBBQjMiqQNEcOo7Jxbpjl8vekmL/pTjAfBg
NVHSMEGDAWgBTLSXPwBjWtZ+uHPxe1vZoL9rospjANBgkqhkiG9w0BAQUFAAOBgQB7rkS45X6rwC8IRAQK9try
+by4C6XID2ZsUX+KXYCcUpWwEHCSVo/Z9+JBHdIkPaeWBMB1zilIXguXYwZSNpSmbMFXivBzNDnOqoyhfBK3Tn
5/LdKGNdBJrg4lfZH1Ww4k814fsZ+LZPsNCv5DjgxfPSs/kNWPFc7WkaiqWylxbA==</wsse:BinarySecurit
yToken>
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"></ds:CanonicalizationMethod>
```

```
               <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"></ds:SignatureMethod>
               <ds:Reference URI="#Body-1-ab8bf4d52e02a26b8b1b2ee414ca8d6a">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>+d1QRlnWih40e8cBs7Pu8bnqz/E=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#PolicyVersion-2-87788f709f148454caa6ffbdcbb38cf5">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>78Nn7ftRAJ25eJKQxutN/dnLM60=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#RelatesTo-3-af2a924c154c6a1d492cd1a13caf2c9e">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>AA65CE8TcLpsQbV0mkqb7xwgOBI=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#Timestamp-4-9863b4d75fb6db844260e7465f132b36">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>o7QuF4MTGn7x7LCB2wSJiCFdE64=</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>

<ds:SignatureValue>b5hlacocWWiiatVzGd7DoaRdbd+3RusYbsYHS8kKGUK2TQUvCsR+kpE+JIo6+eb6Vcs
kn0c1KdPIdBDbbFdHHUAQOMtYpjyp0uTboehVGhMUlhm/Qfvtj26V6SjDs+ZY7Cs6EXVa5jJaZo4aUzsRyXZ76
2ISmrKRzF5r7b8cRaA=</ds:SignatureValue>
            <ds:KeyInfo>
               <wsse:SecurityTokenReference>
                  <wsse:Reference URI=***"#BinarySecurityToken-0-
b09596a4ffadd42ff0e4d37cb5312889***" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"></wsse:Reference>
               </wsse:SecurityTokenReference>
            </ds:KeyInfo>
         </ds:Signature>
      </wsse:Security>
      <L7a:RelatesTo xmlns:L7a="http://www.layer7tech.com/ws/addr"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" wsu:Id="RelatesTo-3-
af2a924c154c6a1d492cd1a13caf2c9e">http://www.layer7tech.com/uuid/41dadd6abe75aa610f1bf
730729b6e17</L7a:RelatesTo>
      <L7a:PolicyVersion xmlns:L7a="http://www.layer7tech.com/ws/addr"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" wsu:Id="PolicyVersion-2-
87788f709f148454caa6ffbdcbb38cf5">1015808|4244eb4ed3642eeb58b5a3372e5ec398</L7a:Policy
Version>
   </soap:Header>
   <soap:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" wsu:Id="Body-1-ab8bf4d52e02a26b8b1b2ee414ca8d6a">
      <wsx:GetPolicyResponse xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/03/mex">
         <wsp:Policy xmlns:L7p="http://www.layer7tech.com/ws/policy"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
            <wsp:All wsp:Usage="Required">
               <L7p:HttpBasic></L7p:HttpBasic>
            </wsp:All>
         </wsp:Policy>
      </wsx:GetPolicyResponse>
   </soap:Body></soap:Envelope>
```

There is a great deal to be said about the response, much more then I am prepared to write about. The high points are:

- SOAP Body carries the actual policy (Useage=Required, HttpBasic)
- SOAP Body is signed using the XML Digital Signature
- RSA SHA1 digest algorithm is used to get the checksums of the parts signed
- Signature covers: soap:body, wsu:Timestamp, wsse:BinarySecurityToken,
- X.509 Certificate is embedded in the header (binary security token)

Because this is a security policy the gateway digitally signs the critical components of the policy to ensure that if it is tampered with in transit it will be obviously invalid and will be rejected. The two messages were exchanged between the VPN Client and the Gateway transparently to the service consumer and the service provider.

Once the VPN Client obtained the policy it was able to modify the original request so that it satisfies the policy expected by the gateway. Figure 13-8 shows the request with the HTTP Basic Authentication header added. The Base64-encoded string contains the username:password combination obtained from the VPN Client configuration – admin:L7.Ap4y0u&me. Connect to http://base64-decoder-online.ewebdev.com/, or another online Base64 decode service, and convince yourself that this is indeed the case.

Figure 13-8 Actual service request decorated according to policy

```
POST /ssg/soap HTTP/1.1
User-Agent: L7 Bridge; Protocol v2.0
SOAPAction: "urn:Sun:Michael:Czapski:WSDL:MoneyOrder/MoneyOrderPortType/opMoneyOrder"
L7-Original-URL: http://localhost:7700/WSSSecGateProvider/MoneyOrderPortTypeBndPort
L7-policy-version: 1015808|4244eb4ed3642eeb58b5a3372e5ec398
Content-Type: text/xml;  charset=utf-8
Authorization: Basic YWRtaW46TDcuQXA0eTB1Jm1l
Host: 127.0.0.1:8081
Content-Length: 1188

<env:Envelope xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Body>
      <ns0:MoneyOrderReq xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
         <tns:OrderDetails>
            <tns:dateTime>2009-02-19T08:58:09.218Z</tns:dateTime>
            <tns:seq>5432</tns:seq>
            <tns:total>1000</tns:total>
         </tns:OrderDetails>
         <tns:SenderDetails>
            <tns:customerName>Jan Kowalski</tns:customerName>
         </tns:SenderDetails>
         <tns:CreditCardDetails>
            <tns:cardType>American Express</tns:cardType>
            <tns:nameOnCard>Jan Kowalski</tns:nameOnCard>
            <tns:cardNumber>1111-111111-111111</tns:cardNumber>
            <tns:securityCode>1111</tns:securityCode>
            <tns:validUntil>10/12</tns:validUntil>
         </tns:CreditCardDetails>
         <tns:ReceipientDetails>
            <tns:familyName>Smith</tns:familyName>
            <tns:streetAddress>33 Berry Street</tns:streetAddress>
            <tns:cityTown>North Sydney</tns:cityTown>
            <tns:stateProvince>NSW</tns:stateProvince>
            <tns:postalCode>2060</tns:postalCode>
            <tns:country>Australia</tns:country>
         </tns:ReceipientDetails>
      </ns0:MoneyOrderReq>
   </env:Body></env:Envelope
```

Finally, the response from the service provider is the same as we would have gotten with the original policy in place, see Figure 13-9.

Figure 13-9 Service response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Content-Length: 665
Date: Thu, 19 Feb 2009 08:57:34 GMT

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns0="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Body>
      <ns0:MoneyOrderRes xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
         <tns:OrderDetails>
            <tns:dateTime>2009-02-19T08:58:09.218Z</tns:dateTime>
            <tns:seq>5432</tns:seq>
            <tns:total>1000</tns:total>
            <tns:orderStatus>true</tns:orderStatus>
         </tns:OrderDetails>
         <tns:SenderDetails>
            <tns:customerName>Jan Kowalski</tns:customerName>
         </tns:SenderDetails>
      </ns0:MoneyOrderRes>
   </env:Body></env:Envelope>
```

Let's now turn out attention to the VPN Client's Service Policy cache, accessible through Properties->Service Policy Tab, see Figure 13-10.

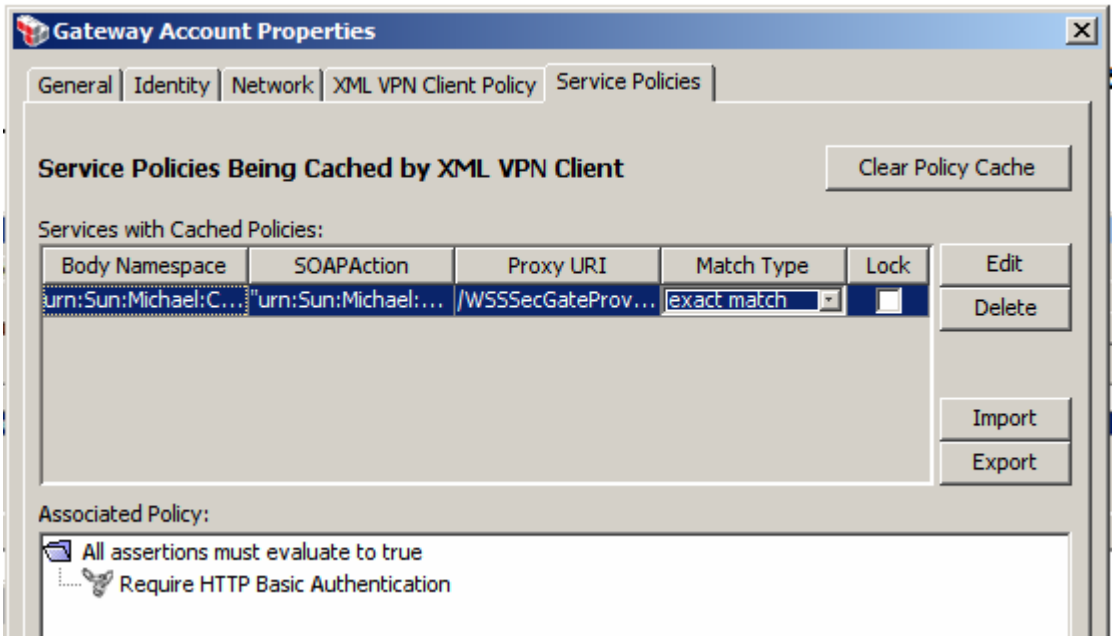

Figure 13-10 Cached Service Policy

Selecting the policy in the "Services with Cached Policies:" list shows the policy in the Associated Policy pane. As is clearly seen the "Require HTTP Basic Authentication" policy, which we originally set for this service in the Gateway, has been propagated to the VPN Client. Note that checking the checkbox in the Lock column will prevent this policy form being automatically updated.

Let’s click the Edit button to see the parts of the request that are used to search the policy cache for the applicable policy, Figure 13-11.

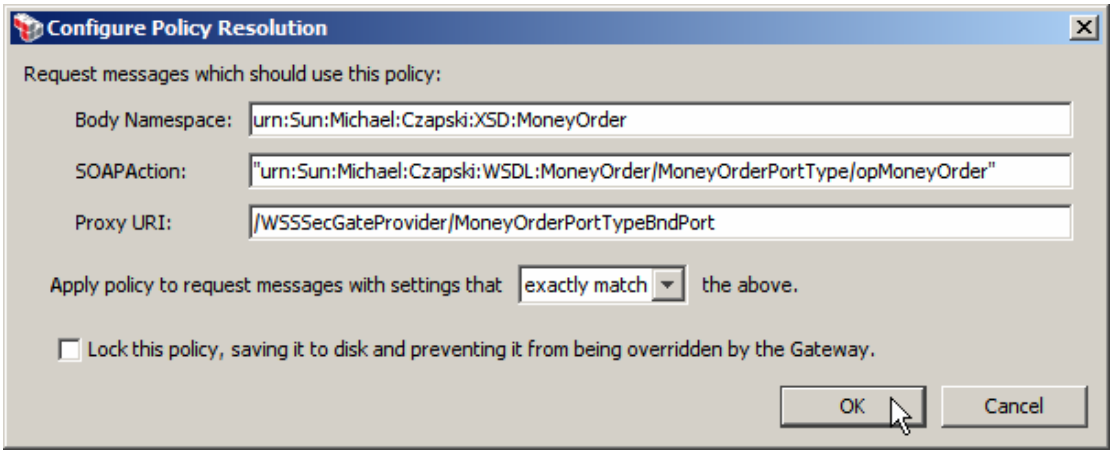

Figure 13-11 Request components used to identify applicable policy

Let’s turn our attention to the VPN Client’s Recent Message Traffic window, which should have been open since the previous test. It will show something similar to what appears in Figure 13-12. Exploring the messages leads us to discover that the VPN Client submitted an undecorated request (it was not aware of a security policy being enforced by the gateway), the gateway returned the policy, the VPN Client cached it in the cache, decorated the request according to the requirement of the policy and submitted it again. The Gateway, having sent the request to the provider and having obtained the response, returned the provider’s response, which the VPN Client returned to the consumer.

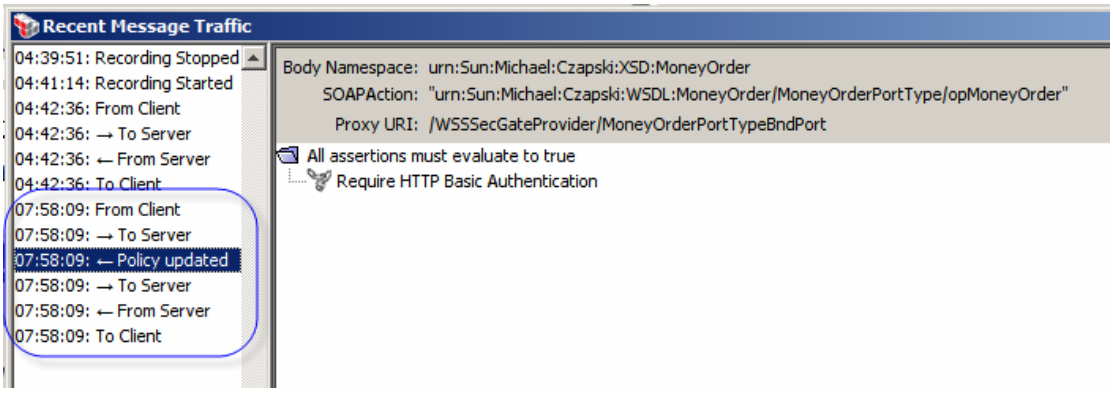

Figure 13-12 Messages exchanged between the VPN Client and the Gateway

No modification to either consumer or provider was required to add a security policy and have it take effect. The SecureSpan VPN Client was automatically provided with the security policy, having tried and having failed to submit the original request, used it to decorate the request and resubmitted the request, all without either the consumer or the provider being aware of the interactions.

To make it clearer, let’s switch to the SSG Manager and change the policy so it requires WS Security username Token instead of the HTTP Basic Authentication header.

Let's add WSS Username Token Assertion from the Access Control Assertions. Figure 13-13 shows the new policy in the SSG Manager window.

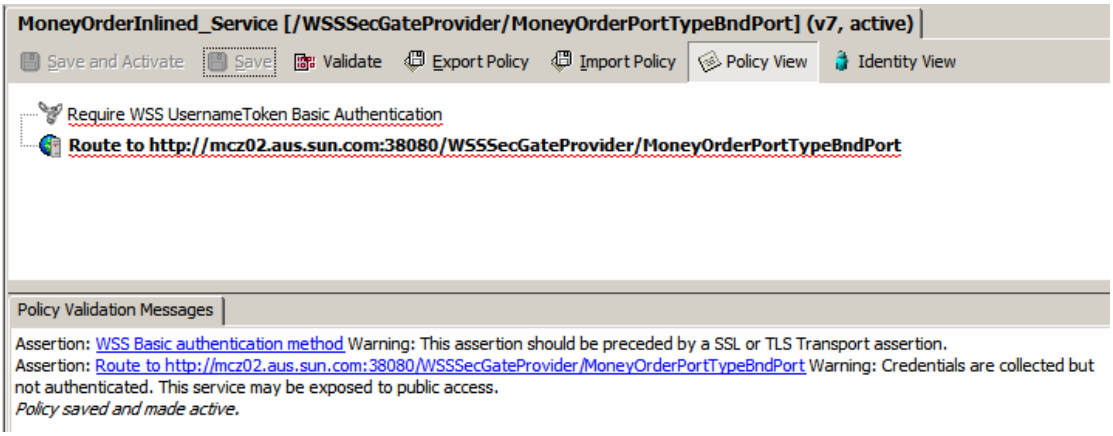

Figure 13-13 New policy assertion

"Save and Activate" the policy, clear the TCP Mon window and submit a message to the inbound JMS Queue, as before. Observe the response in the outbound JMS Queue. Take a look at the message exchange in the TCP Mon window. Take a look at the Service Policy Cache at the VPN Client. Take a look at the VPN Client Recent Message Traffic window.

In VPN Client Properties -> Server Policies observe the updated policy. Figure 13-14 shows the updated policy that requires WS Token Basic Authentication decoration.

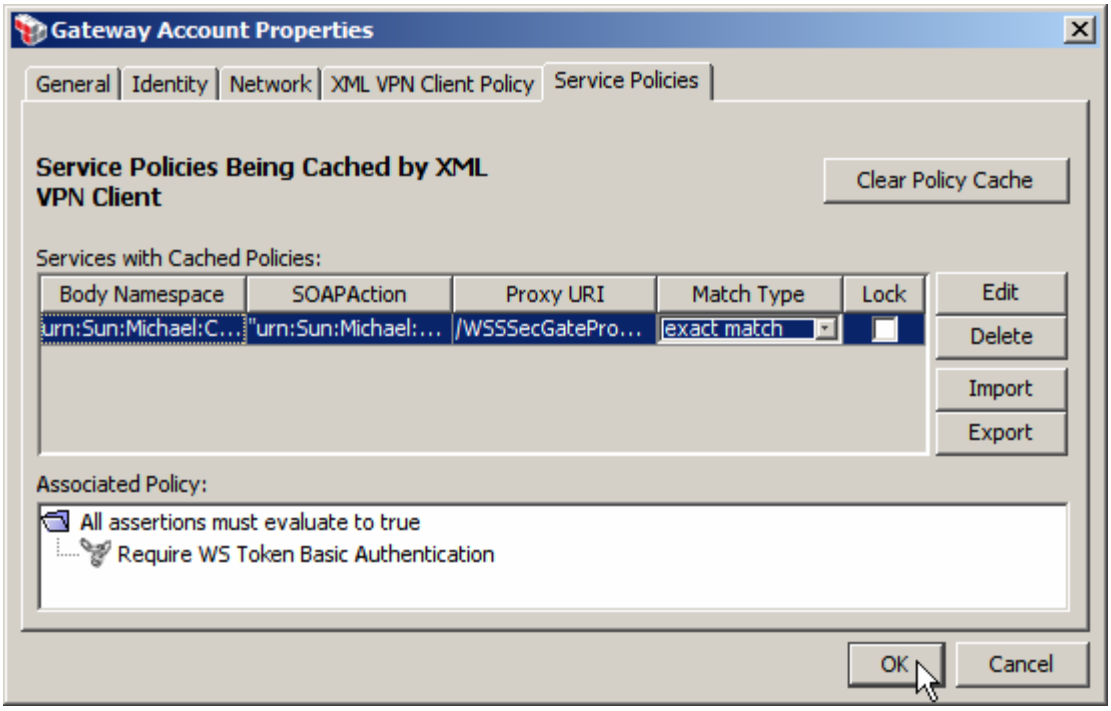

Figure 13-14 WS Token Basic Authentication policy was propagated to the VPN Client

As before, the VPN Client sent a request to the Gateway, the Gateway - having concluded that the message does not comply with the policy - rejected it and sent the new policy. The VPN Client re-decorated the request in compliance with the new policy and sent it again to the Gatewy. This time the Gateway accepted the request,

obtained the response and returned the response to the consumer. Figure 13-15 show the message exchange as seen by the VPN Client in the left hand pane and the re-decorated request in the right-hand pane. The request now carries a SOAP:Header with the wsse:Security section that provides the wsu:Timestamp and wsse:UsernameToken stanzas in accordance with WS-Security standard and the dictates of the security policy.

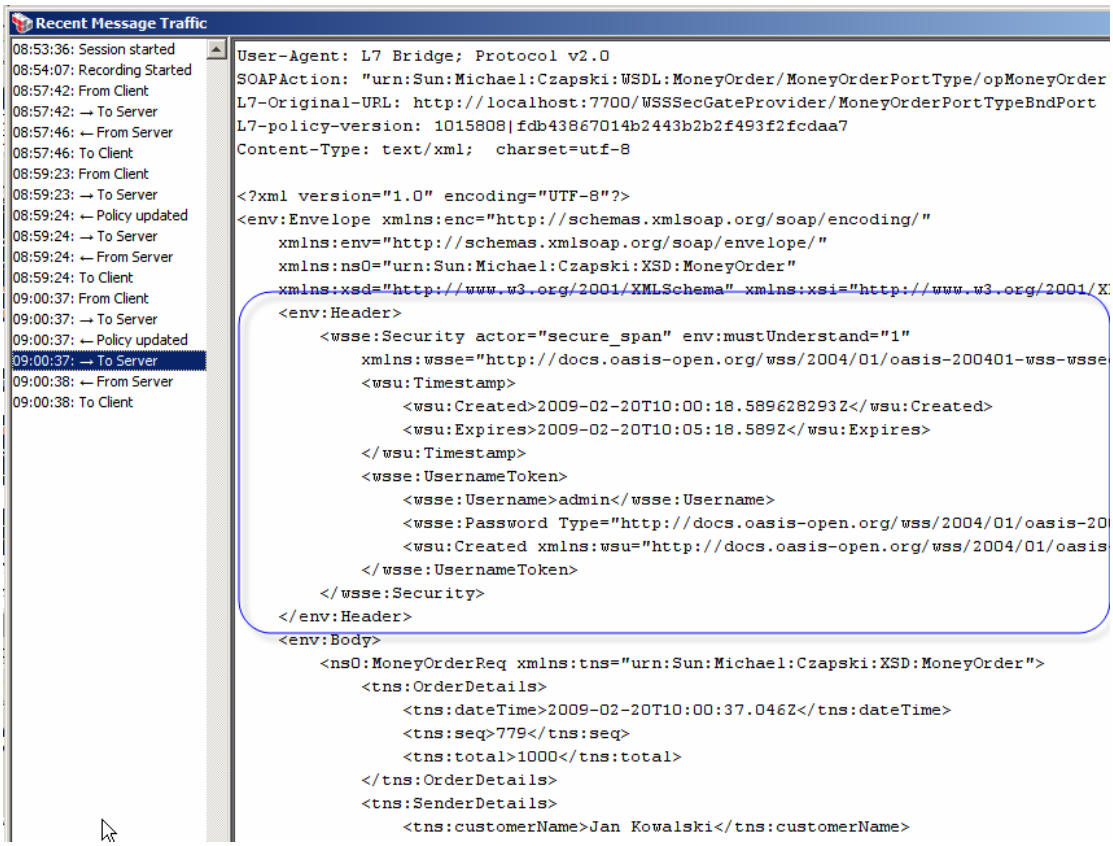

Figure 13-15 SOAP Header with wsse:Security section

As before, no modification to either the consumer or the provider was required to add a security policy and have it take effect. The SecureSpan VPN Client was automatically provided with the updated security policy. Having tried and having failed to submit the original request, the VPN Client used the new policy to decorate the request and resubmitted the request, all without either the consumer or the provider being aware of the interactions.

13.8 *Digital Signing*

If you have not already installed private keys and certificates at the Gateway and the VPN Client, branch out to Section 15.4, “Install certificates”, before resuming here.

Let’s add a requirement to digitally sign the body of the message. Switch to the SSG Manager, select the service so the policy is shown in the right-hand pane. Remove any assertions you may have except for the “Route to” assertion, or revert to the version of the policy which consists of just that one assertion.
Add “Fault Level” assertion from the “Logging, Auditing and Alerts” assertion group and configure it to log at “Full Detail”. Expand the “Authorization Assertions” group and drag the “Encrypted Username Token” to above the “Route to” assertion.

Expand the “Policy Assertions”->”XML Security” assertions group. Drag the “Sign Request Element” assertion onto the Policy canvas, dropping it before the “Route to …” assertion, Figure 13-16.

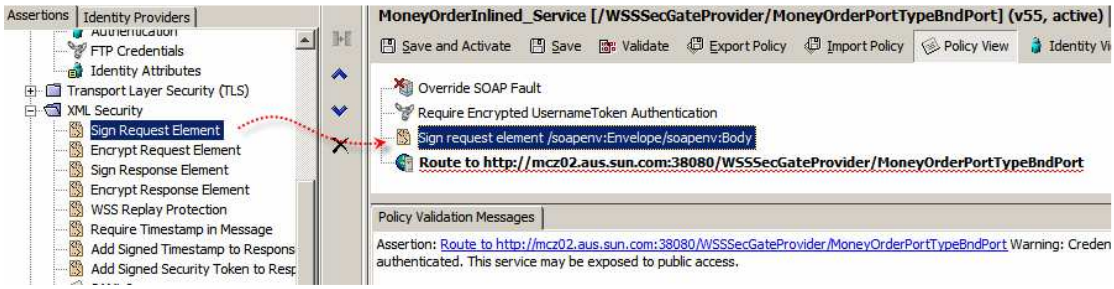

Figure 13-16 Add Sign Request Element assertion

Right-click the policy assertion, select “Sign XML Element” Properties option, choose the soap:Body node (the default) and click OK, Figure 13-17. The assertions will be processed in order, so the Username Token will be added to the soap:Header then the soap:Body will be signed.

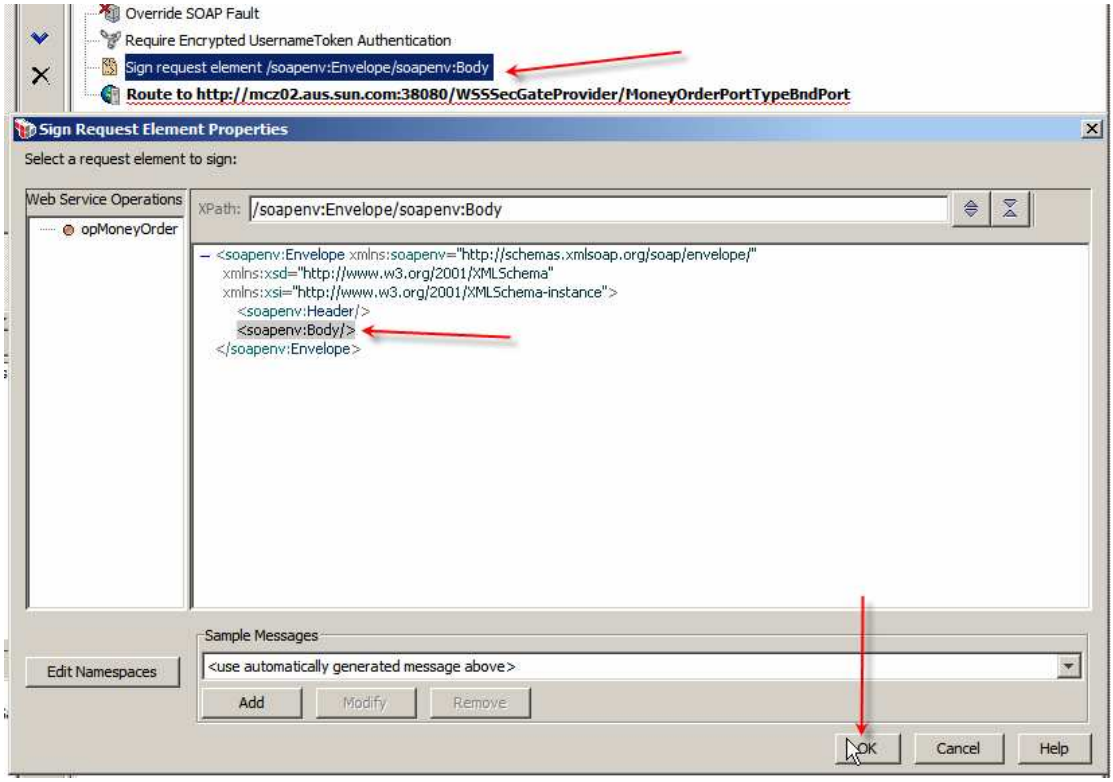

Figure 13-17 Choose soap:Body to sign

“Save and Activate” the policy, submit a JMS Message to trigger the solution, then observe the messages in the VPN Client’s Recent Message Traffic windows and the TCP Mon window.

The VPN Client’s Recent Message Traffic windows shows that the policy was updated, Figure 13-18, as we expect.

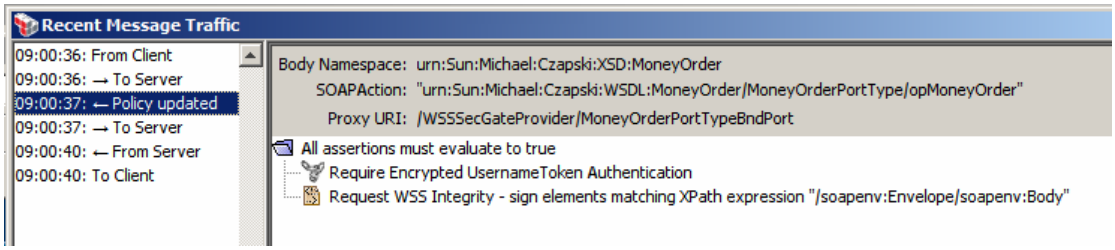

Figure 13-18 Policy was updated

The SOAP Request, sent to the Gateway, has a great deal of XML markup added, Figure 13-19. The soap:Header is particularly large and complex.

Figure 13-19 SOAP Request, decorated to satisfy policy requirements

```
POST /ssg/soap HTTP/1.1
User-Agent: L7 Bridge; Protocol v2.0
SOAPAction: "urn:Sun:Michael:Czapski:WSDL:MoneyOrder/MoneyOrderPortType/opMoneyOrder"
L7-Original-URL: http://localhost:7700/WSSSecGateProvider/MoneyOrderPortTypeBndPort
L7-policy-version: 1015808|3b0be71db26c6296e1a20f1279caf93c
Content-Type: text/xml;  charset=utf-8
Host: 127.0.0.1:8081
Content-Length: 7045

<env:Envelope xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" actor="secure_span"
env:mustUnderstand="1">
         <wsu:Timestamp wsu:Id="Timestamp-3-734e71731195ce05d607e2234c873b14">
            <wsu:Created>2009-02-21T21:59:26.825790865Z</wsu:Created>
            <wsu:Expires>2009-02-21T22:04:26.825Z</wsu:Expires>
         </wsu:Timestamp>
         <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
Id="EncryptedKey-0-002e2f086b947eaf9efa52ef0a9a9123">
            <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
1_5"></xenc:EncryptionMethod>
            <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
               <wsse:SecurityTokenReference>
                  <wsse:KeyIdentifier EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509SubjectKeyIdentifier">IzIqkDRHDqOycW6Y5fL3pJi/6U4=</wsse:KeyIdentifier>
               </wsse:SecurityTokenReference>
            </dsig:KeyInfo>
            <xenc:CipherData>

<xenc:CipherValue>HWOUNnjV3Y9iUQZFrBNu8FTCiHMazG3nSdMTpF7fJdcru6re8bI7W8+RNVT+IrLxtVAN
fbQESGW7uuIYZta426r6KgOW/B8CKf6AMfIDHd/Q5DyYilhnmnY8x3bN0ekMy4yuQkB4/knlPi2v77taYCs5wZ
b/hBglYtydDKVK9ZE=</xenc:CipherValue>
            </xenc:CipherData>
         </xenc:EncryptedKey>
         <wssc:DerivedKeyToken xmlns:wssc="http://schemas.xmlsoap.org/ws/2004/04/sc"
wsu:Id="DerivedKey-Sig-1-aea7c649c407516859d9ec35501478a1"
wssc:Algorithm="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk/p_sha1">
            <wsse:SecurityTokenReference>
               <wsse:Reference URI="#EncryptedKey-0-002e2f086b947eaf9efa52ef0a9a9123"
ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-
1.1#EncryptedKeySHA1"></wsse:Reference>
            </wsse:SecurityTokenReference>
            <wssc:Generation>0</wssc:Generation>
            <wssc:Length>16</wssc:Length>
            <wssc:Label>DerivedKey</wssc:Label>
            <wsse:Nonce>4weOHai+hmn2v1iLfVGYFg==</wsse:Nonce>
         </wssc:DerivedKeyToken>
         <wssc:DerivedKeyToken xmlns:wssc="http://schemas.xmlsoap.org/ws/2004/04/sc"
wsu:Id="DerivedKey-Enc-5-fda45802545986ae65fcb7582a19595c"
wssc:Algorithm="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk/p_sha1">
```

```
            <wsse:SecurityTokenReference>
               <wsse:Reference URI="#EncryptedKey-0-002e2f086b947eaf9efa52ef0a9a9123"
ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-
1.1#EncryptedKey"></wsse:Reference>
            </wsse:SecurityTokenReference>
            <wssc:Generation>0</wssc:Generation>
            <wssc:Length>16</wssc:Length>
            <wssc:Label>DerivedKey</wssc:Label>
            <wsse:Nonce>SRMA5w6cUN+RZh0mhcoeqw==</wsse:Nonce>
         </wssc:DerivedKeyToken>
         <xenc:ReferenceList xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            <xenc:DataReference URI="#EncryptedUsernameToken-6-
9b15dab0c1c95e713006a3238cefd733"></xenc:DataReference>
         </xenc:ReferenceList>
         <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
Id="EncryptedUsernameToken-6-9b15dab0c1c95e713006a3238cefd733"
Type="http://www.w3.org/2001/04/xmlenc#Content">
            <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
cbc"></EncryptionMethod>
            <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
               <wsse:SecurityTokenReference>
                  <wsse:Reference URI="#DerivedKey-Enc-5-
fda45802545986ae65fcb7582a19595c"
ValueType="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk"></wsse:Reference>
               </wsse:SecurityTokenReference>
            </dsig:KeyInfo>
            <CipherData>

<CipherValue>tWJGsQ0cXbtXK4Jb3xsuuBxxjQAD3VS2153OTXgj+usoyAv8P1eAbSp4n7z8t8mjwJyHsVBse
TdXuYDRL5Sv67yraNKRllBgXzGDblRbUhlKfbHjiUbHw7yQyr1tHj3LXVkeN40iEDGO8oF76COlWoIq/OvWzrO
7a3tmJ83W3bZgUDRmYzHZ4BtRItd7H6HelUDc43EB68vZ5dKobmqtmtZWYzQ69Dzo42p+RPOpS0SnC/t+A3Bx6
LSEwutwshtFSIQ7OmGDifIDLU3Kvcy19s5sf2y+9w5QOjhN2dSpw5Y7d5krsSnMBx67mh4liLVal0awAqOTrRK
aARw7ekEIF6aVE/vgelKVAmt+FL7TEpu9sp8l5ki/+pFerlTJ+N3DytPWNQLbDnUBiQiwdg3A0bwLcPnOU2ysu
BMmoceKjjYIuMNzcWAs6H3Rm7ERwl88EoVhNakBlSASlGg32Zd0br+9xqFPwnOTkh8e5j+pOP4BFPtYs0rL79T
s4Gs/5CkoRSDV4/Y144LaZ2HTXsvYH+6D0mSWSZ4vElI7UwgPDIe6bEnZHdLwHFt02PG4Rl0WfjsZ6/WyK45rb
Znd534So76fehGeUinQH0W1PRQYIqE=</CipherValue>
            </CipherData>
         </EncryptedData>
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"></ds:CanonicalizationMethod>
               <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-
sha1"></ds:SignatureMethod>
               <ds:Reference URI="#Body-2-5db1a984d58eeca6d5a5ffa38f66665b">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>v6Y+HypefqIX6+t+GP0ByhfgUWE=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#Timestamp-3-734e71731195ce05d607e2234c873b14">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>5NfzRwAQ/JSaU4WJfc5QROillL0=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#UsernameToken-4-88a798d55fc9ed6fccf5819f658edf6e">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>4sY85e+nxse6LVM9O8jHz6mx010=</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>oLOm80C+kANcANkDAeasYlKv/38=</ds:SignatureValue>
            <ds:KeyInfo>
               <wsse:SecurityTokenReference>
```

```
                  <wsse:Reference URI="#DerivedKey-Sig-1-
aea7c649c407516859d9ec35501478a1"
ValueType="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk"></wsse:Reference>
               </wsse:SecurityTokenReference>
            </ds:KeyInfo>
         </ds:Signature>
      </wsse:Security>
   </env:Header>
   <env:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" wsu:Id="Body-2-5db1a984d58eeca6d5a5ffa38f66665b">
      <ns0:MoneyOrderReq xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
         <tns:OrderDetails>
            <tns:dateTime>2009-02-21T22:00:36.812Z</tns:dateTime>
            <tns:seq>11</tns:seq>
            <tns:total>1000</tns:total>
         </tns:OrderDetails>
         <tns:SenderDetails>
            <tns:customerName>Jan Kowalski</tns:customerName>
         </tns:SenderDetails>
         <tns:CreditCardDetails>
            <tns:cardType>American Express</tns:cardType>
            <tns:nameOnCard>Jan Kowalski</tns:nameOnCard>
            <tns:cardNumber>1111-111111-111111</tns:cardNumber>
            <tns:securityCode>1111</tns:securityCode>
            <tns:validUntil>10/12</tns:validUntil>
         </tns:CreditCardDetails>
         <tns:ReceipientDetails>
            <tns:familyName>Smith</tns:familyName>
            <tns:streetAddress>33 Berry Street</tns:streetAddress>
            <tns:cityTown>North Sydney</tns:cityTown>
            <tns:stateProvince>NSW</tns:stateProvince>
            <tns:postalCode>2060</tns:postalCode>
            <tns:country>Australia</tns:country>
         </tns:ReceipientDetails>
      </ns0:MoneyOrderReq>
   </env:Body></env:Envelope
```

There is a great deal to this markup. I will not go into the details. There is material available on XML Encryption and XML Digital Signatures, not to mention the OASIS standards which dictate how the WS-Security markup is constructed and how it looks like. Notable, in bold and italics, are references to the parts of the SOAP Request which receive “special attention”. For example the soap:body has a wsu:Id attribute added. This attribute associates a unique ID with this part of the message. Scanning up from that spot in the listing we get to the “`ds:Reference URI="#Body-2-5db1a984d58eeca6d5a5ffa38f66665b"`”, nested within the “ds:Signature” tag. This particular tag specifies that the document element identified with this identifier (soap:body in this case) is one of the elements that has been signed. Scanning through ds:Reference tags allows us to discern that Timestamp Token and Username Token are also signed. Notice that where we can see the Timestamp Token further up in the message we cannot see the Username Token, This is because it is encrypted as dictated by the “Require Encrypted UsernameToken Authentication” policy assertion. Rest easy, it dwells somewhere inside the `<EncryptedData>…</EncryptedData >` tag. Note that the soap:body, and all it contains, is plainly visible. We only require signature over the body, not encryption. Note, too, that soap:body markup was modified by the addition of wsu:Id attribute and the related namespace definition.

Let’s briefly look at the SOAP Response returned from the Gateway. Figure 13-20 shows the response.

Figure 13-20 Service response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Content-Length: 1091
```

```
Date: Sat, 21 Feb 2009 21:59:30 GMT

<env:Envelope xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" actor="secure_span"
env:mustUnderstand="1">
         <wsu:Timestamp>
            <wsu:Created>2009-02-21T21:59:30.157221331Z</wsu:Created>
            <wsu:Expires>2009-02-21T22:04:30.157Z</wsu:Expires>
         </wsu:Timestamp>
      </wsse:Security>
   </env:Header>
   <env:Body>
      <ns0:MoneyOrderRes xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
         <tns:OrderDetails>
            <tns:dateTime>2009-02-21T22:00:36.812Z</tns:dateTime>
            <tns:seq>11</tns:seq>
            <tns:total>1000</tns:total>
            <tns:orderStatus>true</tns:orderStatus>
         </tns:OrderDetails>
         <tns:SenderDetails>
            <tns:customerName>Jan Kowalski</tns:customerName>
         </tns:SenderDetails>
      </ns0:MoneyOrderRes>
   </env:Body></env:Envelope>
```

Note that a soap:header with an unsigned wsu:Timestamp was added by the Gateway. We could look at that header to see if the message is a reply message (timestamp expired) but without a valid digital signature we cannot tell whether the timestamp was tampered with, so we cannot trust the timestamp anyway.

So much for the Request-modifying policy. Let's now look at the response, which may be just as important as the request and my require signing. Let's add a "Sign Response Element" assertion after the "Route to" assertion, that is after the request was sent to the back-end service and the response is received from it. We would like to sign the Timestamp token to detect reply attacks and to sign the whole response. Let's drag the "Add Signed Timestamp to Response" and the "Sign Response Element" assertions from the "XML Security" node tree onto the Policy editor canvas, then click the "Save and Activate" button. Figure 13-21 illustrates the policy.

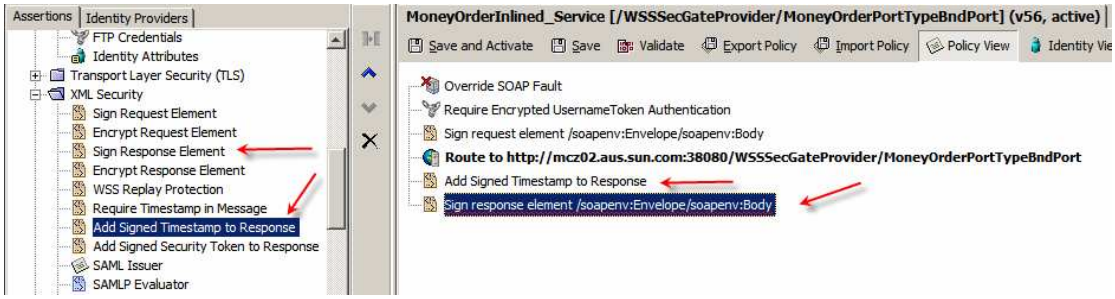

Figure 13-21 Signing assertions for the response

Let's submit a JMS message and observe the request and the response in the TCP Mon window and in the Recent Message Traffic window.

Note the policy update in the Recent Message Traffic window, Figure 13-22.

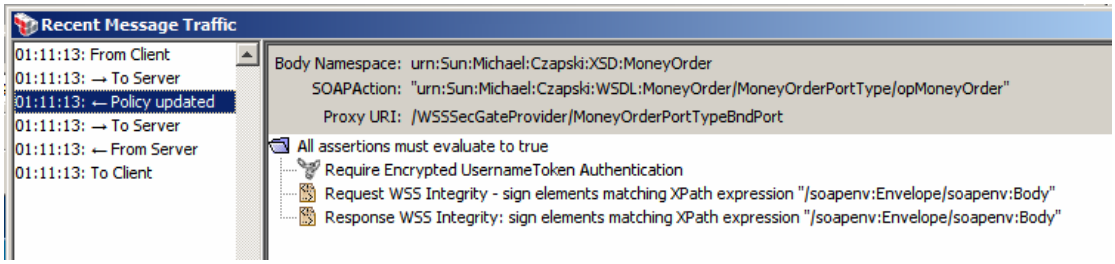

Figure 13-22 Updated policy at the VPN Client

Note the signed response which includes the signed Timestamp Token, Figure 13-22.

Figure 13-23 Signed response from the VPN Client Recent Message Traffic window

```
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Content-Length: 4862
Date: Sun, 22 Feb 2009 02:10:58 GMT

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns0="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <env:Header>
        <wsse:Security actor="secure_span" env:mustUnderstand="1"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd">
            <wsu:Timestamp wsu:Id="Timestamp-3-d39ecea4b4999317d2f906b66dcdf373">
                <wsu:Created>2009-02-22T02:10:58.816630754Z</wsu:Created>
                <wsu:Expires>2009-02-22T02:15:58.816Z</wsu:Expires>
            </wsu:Timestamp>
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3" wsu:Id="BinarySecurityToken-0-
8b6a5876962159 83bef66b2181607998">MIICFjCCAX+gAwIBAgIIAIhey9DpmxQwDQYJKoZIhvcNAQEFBQAw
HzEdMBsGA1UEAwwUcm9vdC5zc2cuYXVzLnN1bi5jb20wHhcNMDkwMjE3MTQzNDM0WhcNMTEwMjE3MTQ0NDM0Wj
AaMRgwFgYDVQQDDA9zc2cuYXVzLnN1bi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJMsd30S24w+
IKgDIpA3liwG+gjwKlV5I5PyvzbtoELVu8jFdhGUoeqv6/i5NjYkFZdhkXrV8J4pCWJJ6yF+jaicoBH3AhXjEf
Ul7xcFKMrXtTq5AFt74ksFVn2Aet8Tbt49ctvzFLnHgDNqPUhtrSSmINXKRVARQ/V9An+4WgovAgMBAAGjYDBe
MAwGA1UdEwEB/wQCMAAwDgYDVR0PAQH/BAQDAgXgMB0GA1UdDgQWBBQjMiqQNEcOo7Jxbpjl8vekmL/pTjAfBg
NVHSMEGDAWgBTLSXPwBjWtZ+uHPxe1vZoL9rospjANBgkqhkiG9w0BAQUFAAOBgQB7rkS45X6rwC8IRAQK9try
+by4C6XID2ZsUX+KXYCcUpWwEHCSVo/Z9+JBHdIkPaeWBMB1zilIXguXYwZSNpSmbMFXivBzNDnOqoyhfBK3Tn
5/LdKGNdBJrg4lfZH1Ww4k814fsZ+LZPsNCv5DjgxfPSs/kNWPFc7WkaiqWylxbA==</wsse:BinarySecurit
yToken>
            <wssc:DerivedKeyToken

wssc:Algorithm="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk/p_sha1"
                wsu:Id="DerivedKey-Sig-1-ff0125610c14fba9d6cb029757007283"
xmlns:wssc="http://schemas.xmlsoap.org/ws/2004/04/sc">
                <wsse:SecurityTokenReference>
                    <wsse:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-soap-message-security-
1.1#EncryptedKeySHA1">ltjdYTuqK36yOwLJjtcGTpYeQq8=</wsse:KeyIdentifier>
                </wsse:SecurityTokenReference>
                <wssc:Generation>0</wssc:Generation>
                <wssc:Length>16</wssc:Length>
                <wssc:Label>DerivedKey</wssc:Label>
                <wsse:Nonce>Zg6TEFsTmwiig7Cxj+/ctA==</wsse:Nonce>
            </wssc:DerivedKeyToken>
            <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                    <ds:Reference URI="#Body-2-e73e82aaf3dfff2bdb9555ee958d5bd5">
                        <ds:Transforms>
```

```
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>o2eT8i8BoKbdrHyt2nXYIfGv5ck=</ds:DigestValue>
                    </ds:Reference>
                    <ds:Reference URI="#Timestamp-3-d39ecea4b4999317d2f906b66dcdf373">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>69tXhPwfJLLG9znuyFUAxJV6hEg=</ds:DigestValue>
                    </ds:Reference>
                    <ds:Reference URI="#RelatesTo-4-13cab008f1a75678b3f4465b1d3b222f">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>wXIrqag1sHXDP32cOXuERXpTJAQ=</ds:DigestValue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>2znrlFjqL5KS6+kvK9D7eOCNoTY=</ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference
                            URI="#DerivedKey-Sig-1-ff0125610c14fba9d6cb029757007283"
ValueType="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
        <L7a:RelatesTo
            wsu:Id="RelatesTo-4-13cab008f1a75678b3f4465b1d3b222f"
            xmlns:L7a="http://www.layer7tech.com/ws/addr"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">http://www.layer7tech.com/uuid/2fc26406f98914b9fdb2738010a4e3d7</L7a:RelatesT
o>
    </env:Header>
    <env:Body wsu:Id="Body-2-e73e82aaf3dfff2bdb9555ee958d5bd5"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
        <ns0:MoneyOrderRes xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
            <tns:OrderDetails>
                <tns:dateTime>2009-02-22T02:11:12.937Z</tns:dateTime>
                <tns:seq>44</tns:seq>
                <tns:total>1000</tns:total>
                <tns:orderStatus>true</tns:orderStatus>
            </tns:OrderDetails>
            <tns:SenderDetails>
                <tns:customerName>Jan Kowalski</tns:customerName>
            </tns:SenderDetails>
        </ns0:MoneyOrderRes>
    </env:Body>
</env:Envelope>
```

Note the ds:Reference URI values showing the parts of the message that were included in the signature, and the wsu:Id attributes that attach unique IDs to these parts of the message. Now the Timestamp Token can be trusted if the digital signature is valid.

Note that both the request and the response are conveyed as plaintext. Anyone, snooping on the wire, can see what the credit card information is and what the order status is. We have not been using explicit encryption of the message body up to this point.

Note how much markup gets added to each message to satisfy these kinds of non-functional requirements.

13.9 *Encryption*

Digital signatures are there to protect message integrity (make sure the message has not been altered in transit) and to convey authenticity of the sender, assumed to be the same as the signer of the message (only the specific signer could have signed the message).

As noted before, the message body is still transmitted in the clear. Given that the request conveys credit card information it would probably be a good idea to protect this information from eavesdroppers. To do this we can encrypt the message.

Let's add "Encrypt Request Element" to the policy just before the "Route to" assertion and "Encrypt Response Element" at the end of the policy, Figure 13-24.

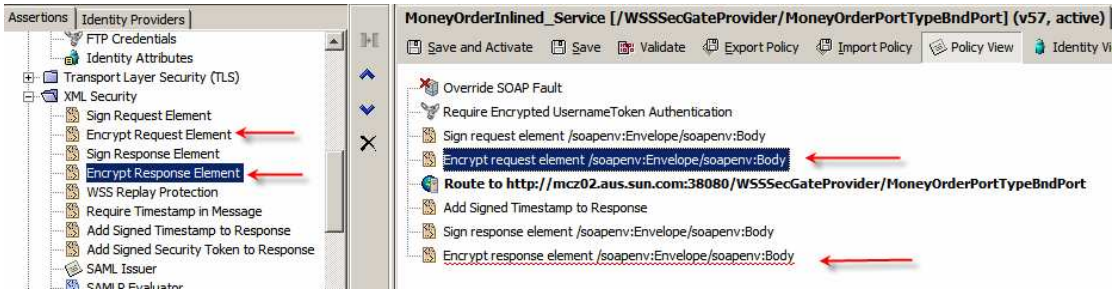

Figure 13-24 Add Encryption policy assertions

The reason we are adding encryption after signing is because the assertions are processed in order and their effect is cumulative. It makes sense to encrypt signed message rather then the sign encrypted message, though there may be circumstances where signing after encryption is appropriate. Let's "Save and Activate", submit a test message and observe the messages exchanged between the VPN Client, which applies security to the request and strips security form the response, and the Gateway, which processes security of the request and decorates the response according to the security policy.

As before, the policy was updated at the VPN Client, Figure 13-25. The client now knows how to decorate the request and how to process the response.

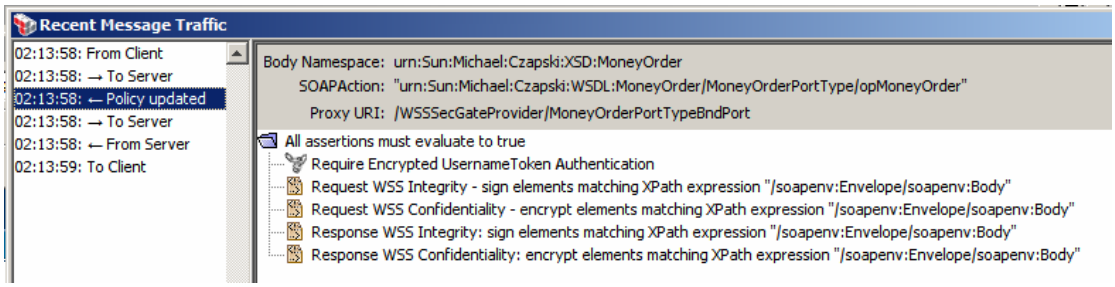

Figure 13-25 Updated VPN Client policy

The request, leaving the client and arriving at the Gateway, now has encrypted sosap:body, Figure 13-26.

Figure 13-26 Signed and Encrypted Request

```
<env:Envelope xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" actor="secure_span"
env:mustUnderstand="1">
         <wsu:Timestamp wsu:Id="Timestamp-3-a8029d1e21ee34605c3ca4404f239504">
            <wsu:Created>2009-02-22T03:12:57.869403078Z</wsu:Created>
            <wsu:Expires>2009-02-22T03:17:57.869Z</wsu:Expires>
         </wsu:Timestamp>
         <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
Id="EncryptedKey-0-e3c890e9f648982de5f6d375967129c7">
            <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
1_5"></xenc:EncryptionMethod>
            <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
               <wsse:SecurityTokenReference>
                  <wsse:KeyIdentifier EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509SubjectKeyIdentifier">IzIqkDRHDqOycW6Y5fL3pJi/6U4=</wsse:KeyIdentifier>
               </wsse:SecurityTokenReference>
            </dsig:KeyInfo>
            <xenc:CipherData>

<xenc:CipherValue>aGsrdYSj6E1LLhQ9oZO1uphxZ9v06fdLnLb3GwyJIApYE5hqjMiv63T3aZ633KLU28Av
pk1m6A8btUqLBOm5wB0+jpzrbqT1+tJjDzkCEBnNaG6tKOYO8sPYgA3YjzpwrTaDGXE1BRiunLZL2QqjgCeqRL
mutDb+8j4KwjatTjQ=</xenc:CipherValue>
            </xenc:CipherData>
         </xenc:EncryptedKey>
         <wssc:DerivedKeyToken xmlns:wssc="http://schemas.xmlsoap.org/ws/2004/04/sc"
wsu:Id="DerivedKey-Sig-1-2fee7bc234c4140a6305d0c0fe89e115"
wssc:Algorithm="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk/p_sha1">
            <wsse:SecurityTokenReference>
               <wsse:Reference URI="#EncryptedKey-0-e3c890e9f648982de5f6d375967129c7"
ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-
1.1#EncryptedKeySHA1"></wsse:Reference>
            </wsse:SecurityTokenReference>
            <wssc:Generation>0</wssc:Generation>
            <wssc:Length>16</wssc:Length>
            <wssc:Label>DerivedKey</wssc:Label>
            <wsse:Nonce>INay0bvIBPu8OaxkIfHEFg==</wsse:Nonce>
         </wssc:DerivedKeyToken>
         <wssc:DerivedKeyToken xmlns:wssc="http://schemas.xmlsoap.org/ws/2004/04/sc"
wsu:Id="DerivedKey-Enc-6-4c458bc9cc36bdb7a2583f0cc5134edc"
wssc:Algorithm="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk/p_sha1">
            <wsse:SecurityTokenReference>
               <wsse:Reference URI="#EncryptedKey-0-e3c890e9f648982de5f6d375967129c7"
ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-
1.1#EncryptedKey"></wsse:Reference>
            </wsse:SecurityTokenReference>
            <wssc:Generation>0</wssc:Generation>
            <wssc:Length>16</wssc:Length>
            <wssc:Label>DerivedKey</wssc:Label>
            <wsse:Nonce>djCSWD1zEoe8piE4EL0bzw==</wsse:Nonce>
         </wssc:DerivedKeyToken>
         <xenc:ReferenceList xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            <xenc:DataReference URI="#Body-7-
407af680717c88917a1e521995f9ff63"></xenc:DataReference>
            <xenc:DataReference URI="#EncryptedUsernameToken-8-
a56260c6ae446d13cd9617b934208f5f"></xenc:DataReference>
         </xenc:ReferenceList>
         <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
Id="EncryptedUsernameToken-8-a56260c6ae446d13cd9617b934208f5f"
Type="http://www.w3.org/2001/04/xmlenc#Content">
            <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
cbc"></EncryptionMethod>
            <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
               <wsse:SecurityTokenReference>
                  <wsse:Reference URI="#DerivedKey-Enc-6-
4c458bc9cc36bdb7a2583f0cc5134edc"
ValueType="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk"></wsse:Reference>
```

```
               </wsse:SecurityTokenReference>
            </dsig:KeyInfo>
            <CipherData>

<CipherValue>YLR5o1mfxo1rgHopdObPs3HgU0WLL3/dAX7bNSm/ICXp3nxyIzqQr2jQUO+3FH20YFxE20KIm
YlajlxSZqknqlxsJwIKd53PbdRxR5uvcYzC+/D0iRRlE8lR3o+lSdZaW+k3eXrWLM84cx4roLiZzSvYln35GHD
w/NLDftD6Id+ryvUzrN4zs6KpWvAbeT2oQxdrmHOZrDT4sMAQ1lptQv8Uz9u4iyQazWaumuJPNfLYz8si0lQW8
6N4zcvvAKzw6W/VWFkbA1ztHqvdxAkeVtzhtNTCu65Lwy9aHGg2EWPOFQ0hdZy7AhV5RvM/h24xxfoo5VI7pp/
rK5SJYdBMg0ymY4UInfD55Lh27xKRpbbCGA0JipR56trpY6YZRtH+f1EGR/E6t3BHikfdqBuMLcBOkTfo8pXXN
Lh+MztAHcsqBVfFIzY4iUjlyPO/C1cUeqoHCWp1BtfvWest9jfYvz9g2eEMJ21RwQIYeQUE/8pnZL2X/vX9x5C
2xSFF93cH6SDKBv7ysfP+T2KD23uqMIyXJPBQJZFu1nuB2sRxpghBtDD4Aif5TbFqH3lXbs/I6qgjXjX+eZnSl
tfW6TsnQ2W4i9aIZSpWl5maLnKi7TE=</CipherValue>
            </CipherData>
         </EncryptedData>
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"></ds:CanonicalizationMethod>
               <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-
sha1"></ds:SignatureMethod>
               <ds:Reference URI="#Body-2-1a75576839089d6f506d869e3e2795ad">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>ZLlSXFCjztNK3+gKj8oGzPtfIIM=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#Timestamp-3-a8029d1e21ee34605c3ca4404f239504">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>mLqHRZVnxm5kmJqnxgCA8dfRExk=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#UsernameToken-4-b20548729ac444a733c9fcab804e2b4c">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>TBNbt7YG7yAJVhF0fDVJDI2Nouo=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#MessageID-5-14f45cb1e26805d106b5122a8d181911">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
                  <ds:DigestValue>C4HTRIwGd+dSe3CJEUABxKCkU+g=</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>2bHgqyl4aIdqX4lbJ4Y3Vqmxm9E=</ds:SignatureValue>
            <ds:KeyInfo>
               <wsse:SecurityTokenReference>
                  <wsse:Reference URI="#DerivedKey-Sig-1-
2fee7bc234c4140a6305d0c0fe89e115"
ValueType="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk"></wsse:Reference>
               </wsse:SecurityTokenReference>
            </ds:KeyInfo>
         </ds:Signature>
      </wsse:Security>
      <L7a:MessageID xmlns:L7a="http://www.layer7tech.com/ws/addr"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" wsu:Id="MessageID-5-
14f45cb1e26805d106b5122a8d181911">http://www.layer7tech.com/uuid/758de6e6fef32f106e7c3
0c5b1ec415b</L7a:MessageID>
   </env:Header>
   <env:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" wsu:Id="Body-2-1a75576839089d6f506d869e3e2795ad">
      <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#" Id="Body-7-
407af680717c88917a1e521995f9ff63" Type="http://www.w3.org/2001/04/xmlenc#Content">
```

```
         <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
cbc"></EncryptionMethod>
         <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
            <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
               <wsse:Reference URI="#DerivedKey-Enc-6-
4c458bc9cc36bdb7a2583f0cc5134edc"
ValueType="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk"></wsse:Reference>
            </wsse:SecurityTokenReference>
         </dsig:KeyInfo>
         <CipherData>

<CipherValue>Nccyfx/m2LRzWTDM04KvbCayfmsdzlmireCE9NDUN3gRF0rabaoFIX8uI5sk/5u52880HSYLa
AWjQPbU9BEWxH0lA2PnTTAzwTDn0nHzt9iw+19hNDV2LzQVkLuL9SwQ3Lag8l1bR78PSwn6yP6iuj3IElcgSZ2
EXx4OjBzCL5oGyEZzHgvaZ5s7W5UrRi7dFb+C3TdUa3kciNDIrvcVH0xFY7mSBSl1ul5QC+oIw2US1K9rzyHY4
s77uHU+QJx2sLqtnVScCMqxuaBb0ZBVKjf8qwTwUYY5kDBoSP2a20lGmq29UxqtKr8zfLyNEInTnxs2xaYW3B2
Z/3Z8++xu81jAdSHUNl6yp/U8cBADhbvlHum+ESz/Z3zCSCKRnigkxbdbKNRLld3QOpDD81O//aYLxr9myDSKE
o2ocdGgcXGaVCz2KmbPVlxXNF0OOEMXWOi5TsQnA14gKG0Q3pcl/Q1i1l1IpmYzi786Yp33RNdCkMLGtEvgIDD
KoqNj6HEdLZeJYijnMCjJyFFX4kcoOa96nLOkTvvFOIsPF5CMmkL8f2hUaTbYQn2IyQgoxNeI0H9Hz3WxM6fjC
jJ0YAJTcqKp7AZwonXGvPe6UckGVxElMQ8/jhk7LT1YLiqEJHralQ62dJ5Zae0tc6V+GtzTozVUtzy8gGhM/De
7QYbz6OaFQztBaVJ9Vzf81dB08iAM6hUIT9OgmoxKLFNdDyINf21mEZLr1Xl1ZqB9f23nhjTawzRQYn5In/OoI
MbMqSw+AY29nKuZFUbza5XSbg502OgkAhpKuWkzfoSWDuGtA4oFUfvI0Xo0JBjYIL4GrfW9VHUv8o5uQ7Y0AcT
yetH7ak2d/zlKorznDRl7nQkzZBr2k5szGYMmQHqfue94FH8B5sVWT+b4KAfXySnWCsfQTidetE+7x79Mqiq4q
o0qHJPCpBJh1WiYHxLd4vvVNPt8keZVMcVxCV1cWeNguNFckAn2oz1G9tEfvAZ7LYJln4BjH0eWa6l/p8Sn2ZT
Dtc1S3b+1fG5DN6I170XEqq4Hfx9m7APJft7PqVMC/yfJEwHC+KIMcapeSw5ys1bq02NIs0FJibhr9Grvp58v2
Ep9d0wPsOdzV0XK4G3IgxQznrlaPsG8ZlntYGvmXm2WcDcMUXmqrWFLeg0MAzg8Pukf3i1UfZiJANM4vWD8zEW
lrBc=</CipherValue>
         </CipherData>
      </EncryptedData>
   </env:Body></env:Envelope
```

Note that the content of the soap:Body element was replaced with the EncrypetdData element. Event with the base64-encoded encrypted data in the soap:Body, which makes the soap:Body 30% larger, the soap:Body is now less then 1/6th of the request size. WS-Security adds a tremendous amount of overhead both in terms of the on-the-wire data and in terms of processing resources required to decorate the message and to process WS-Security-mandated markup. ECO-friendly this is not.

Let’s take a look at the Response message, Figure 13-27.

Figure 13-27 Response message

```
<env:Envelope xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" actor="secure_span"
env:mustUnderstand="1">
         <wsu:Timestamp wsu:Id="Timestamp-3-2d4855e7faabb02d141fcab66182496c">
            <wsu:Created>2009-02-22T03:12:57.977813322Z</wsu:Created>
            <wsu:Expires>2009-02-22T03:17:57.977Z</wsu:Expires>
         </wsu:Timestamp>
         <wsse:BinarySecurityToken EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3" wsu:Id="BinarySecurityToken-0-
b7075f001bd857f17101e337d95bea08">MIICFjCCAX+gAwIBAgIIAIhey9DpmxQwDQYJKoZIhvcNAQEFBQAw
HzEdMBsGA1UEAwwUcm9vdC5zc2cuYXVzLnN1bi5jb20wHhcNMDkwMjE3MTQzNDM0WhcNMTEwMjE3MTQ0NDM0Wj
AaMRgwFgYDVQQDDA9zc2cuYXVzLnN1bi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJMsd30S24w+
IKgDIpA3liwG+gjwKlV5I5PyvzbtoELVu8jFdhGUoeqv6/i5NjYkFZdhkXrV8J4pCWJJ6yF+jaicoBH3AhXjEf
Ul7xcFKMrXtTq5AFt74ksFVn2Aet8Tbt49ctvzFLnHgDNqPUhtrSSmINXKRVARQ/V9An+4WgovAgMBAAGjYDBe
MAwGA1UdEwEB/wQCMAAwDgYDVR0PAQH/BAQDAgXgMB0GA1UdDgQWBBQjMiqQNEcOo7Jxbpjl8vekmL/pTjAfBg
NVHSMEGDAWgBTLSXPwBjWtZ+uHPxe1vZoL9rospjANBgkqhkiG9w0BAQUFAAOBgQB7rkS45X6rwC8IRAQK9try
+by4C6XID2ZsUX+KXYCcUpWwEHCSVo/Z9+JBHdIkPaeWBMB1zilIXguXYwZSNpSmbMFXivBzNDnOqoyhfBK3Tn
5/LdKGNdBJrg4lfZH1Ww4k814fsZ+LZPsNCv5DjgxfPSs/kNWPFc7WkaiqWylxbA==</wsse:BinarySecurit
yToken>
```

<wssc:DerivedKeyToken xmlns:wssc="http://schemas.xmlsoap.org/ws/2004/04/sc"
wsu:Id="DerivedKey-Sig-1-3d1f7642ed2e35b8d16ee28994c5803e"
wssc:Algorithm="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk/p_sha1">
<wsse:SecurityTokenReference>
<wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-
wss-soap-message-security-
1.1#EncryptedKeySHA1">+DB5vPMbQE6c16F2s2Ugugnjs4E=</wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
<wssc:Generation>0</wssc:Generation>
<wssc:Length>16</wssc:Length>
<wssc:Label>DerivedKey</wssc:Label>
<wsse:Nonce>F6Oc+OSZ+Lv2SK27AUOPlg==</wsse:Nonce>
</wssc:DerivedKeyToken>
<wssc:DerivedKeyToken xmlns:wssc="http://schemas.xmlsoap.org/ws/2004/04/sc"
wsu:Id="DerivedKey-Enc-5-9b8a940de8109092a4f7a3852ca3ede1"
wssc:Algorithm="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk/p_sha1">
<wsse:SecurityTokenReference>
<wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-
wss-soap-message-security-
1.1#EncryptedKeySHA1">+DB5vPMbQE6c16F2s2Ugugnjs4E=</wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
<wssc:Generation>0</wssc:Generation>
<wssc:Length>16</wssc:Length>
<wssc:Label>DerivedKey</wssc:Label>
<wsse:Nonce>HyspdcQdokoQz1vXYvqHIw==</wsse:Nonce>
</wssc:DerivedKeyToken>
<xenc:ReferenceList xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:DataReference URI="#Body-6-
3aa8cac38960dfb513448c5714acb5a2"></xenc:DataReference>
</xenc:ReferenceList>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"></ds:CanonicalizationMethod>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-
sha1"></ds:SignatureMethod>
<ds:Reference URI="#Body-2-4a11add2aef920c8281fc81ead115e67">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
</ds:Transforms>
<ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
<ds:DigestValue>Fkm/cCRt9LaSCbSCmZWZgl3WB9I=</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#Timestamp-3-2d4855e7faabb02d141fcab66182496c">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
</ds:Transforms>
<ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
<ds:DigestValue>xhXHXpPytbGttSp0uUYZAUhWpcs=</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#RelatesTo-4-8c1ee4c1e306c3882c9383a4f7322635">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:Transform>
</ds:Transforms>
<ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
<ds:DigestValue>WMUCqYBXzdMy3Q9BJfHyLsgQZZE=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>fjaVFhOyldFJS/Pdi8Xk1B6Lcng=</ds:SignatureValue>
<ds:KeyInfo>
<wsse:SecurityTokenReference>
<wsse:Reference URI="#DerivedKey-Sig-1-
3d1f7642ed2e35b8d16ee28994c5803e"
ValueType="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk"></wsse:Reference>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
<L7a:RelatesTo xmlns:L7a="http://www.layer7tech.com/ws/addr"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" wsu:Id="RelatesTo-4-

```
8c1ee4c1e306c3882c9383a4f7322635">http://www.layer7tech.com/uuid/758de6e6fef32f106e7c3
0c5b1ec415b</L7a:RelatesTo>
   </env:Header>
   <env:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" wsu:Id="Body-2-4a11add2aef920c8281fc81ead115e67">
      <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#" Id="Body-6-
3aa8cac38960dfb513448c5714acb5a2" Type="http://www.w3.org/2001/04/xmlenc#Content">
         <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
cbc"></EncryptionMethod>
         <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
            <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
               <wsse:Reference URI="#DerivedKey-Enc-5-
9b8a940de8109092a4f7a3852ca3ede1"
ValueType="http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk"></wsse:Reference>
            </wsse:SecurityTokenReference>
         </dsig:KeyInfo>
         <CipherData>

<CipherValue>z+Pzh7GNe2CqhJGz1u/k01mShYdYYYyrp69q32Lk2IJCBnk9qxlsn9ZLp1BYe//WSVJ2K4GsQ
GqadH13XlfJOK5YRJ/jnSJ7ouz/yBldAB0LWS3Xo3FOonR70fRcz4RDZltjRunBu2ryDi06LfKZlQxzayqf6ty
HGH5oirvU8krKcNOeXTnPjPKMKlD/SU5e80lON4JjKF+OJpre6LgDS1kHgeUFPWpatcjLzNbX3APdbcATOpoxg
fl/hQc9/njuxJavnh8iTAL2fclUraPhh58PD/uKJaE2SZnjzAMiNRHW4RlELiZTqKkFrTVg0Ho50k42enoPmni
Jc0Pn6IUoss5SZSQ5T32x8JcXLYT58hAzL/shY3Vvwx0dqa0PTwrebJSzTurRAfYCsHoUUQHBxww/1Dbn2XawS
DD+8sOpBFFDYRV0plY16BXN6S/THTSOPYYQgve/RAtD+1qCDcZGfRDBp3AQBnAzNputze+uPaHjiuR75Wl5vqg
iEXuEFYss</CipherValue>
         </CipherData>
      </EncryptedData>
   </env:Body></env:Envelope>
```

The increase in size and complexity is even more glaring in the response, which when undecorated is fairly small, Figure 13-28.

Figure 13-28 Undecorated Response

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns0="urn:Sun:Michael:Czapski:XSD:MoneyOrder"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <env:Header>
        <L7a:RelatesTo
            wsu:Id="RelatesTo-4-8c1ee4c1e306c3882c9383a4f7322635"
            xmlns:L7a="http://www.layer7tech.com/ws/addr"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">http://www.layer7tech.com/uuid/758de6e6fef32f106e7c30c5b1ec415b</L7a:RelatesT
o>
    </env:Header>
    <env:Body wsu:Id="Body-2-4a11add2aef920c8281fc81ead115e67"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
        <ns0:MoneyOrderRes xmlns:tns="urn:Sun:Michael:Czapski:XSD:MoneyOrder">
            <tns:OrderDetails>
                <tns:dateTime>2009-02-22T03:13:58.312Z</tns:dateTime>
                <tns:seq>12</tns:seq>
                <tns:total>1000</tns:total>
                <tns:orderStatus>true</tns:orderStatus>
            </tns:OrderDetails>
            <tns:SenderDetails>
                <tns:customerName>Jan Kowalski</tns:customerName>
            </tns:SenderDetails>
        </ns0:MoneyOrderRes>
    </env:Body>
</env:Envelope>
```

Have you noticed the `wsse:BinarySecurityToken` tag and its base64-encoded content?

Let’s copy the entire base64-encoded text between the quotation marks from the response message, create a new text document named cert.crt, paste the content of the

clipboard into this document, Figure 13-29, save the document and double-click the document (on Windows) to open it.

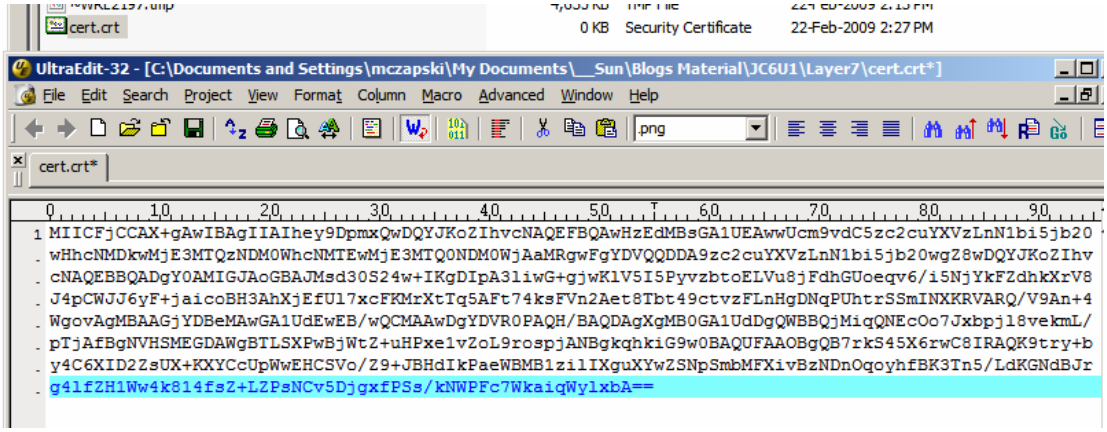

Figure 13-29 Base64-encoded content of the BinarySecurityToken

The context of the token is the X.509 certificate of the signer of the message, the ssg.aus.sun.com, in PEM format, Figure 13-30.

Figure 13-30 ssg certificate

The signer's certificate is embedded in the message so that the recipient can confirm authenticity of the message by verifying the digital signature. It is very nice of the sender to include the certificate but secure installation would not trust this certificate anyway but rather they would use their own copy of the sender's certificate, obtained out-of-band, to thwart potential man-in-the-middle attacks.

We cannot do the same to the request message because the binary security token, which carries the base64-encoded certificate of the signer, was encrypted as part of the Username Token and related tags.

13.10 *Authentication*

We already dealt with authentication in an oblique way by adding Encrypted Username Token and HTTP BASIC Authentication assertions in the examples earlier.

There are a number of assertions which can be used to convey and require sender authentication. Some notable ones, in the list in Figure 13-31, are HTTP Basic Authentication (which is not rally a WS-Security-related method), WSS Username Token Basic and the Encrypted Username Token (which are WS-Security tokens), and WSS Signature which conveys authentication information using the properties of the public cryptosystem key pairs (private keys are held by the owners, public keys are embedded in X.509 certificates and can be used to verify digital signature prepared using private keys).

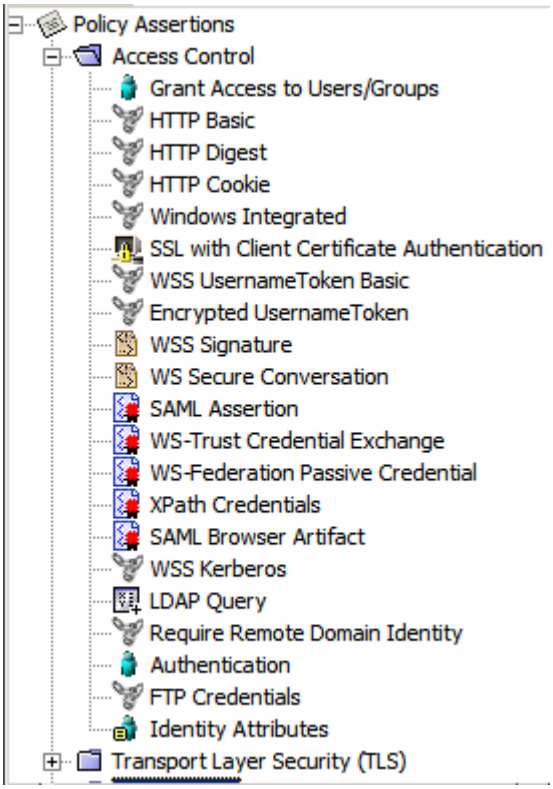

Figure 13-31 Access Control assertions

Other assertions, like XPath Credentials, provide support for conveyance of credentials using non-standard means, for example through embedding credential information in XML message bodies.

Since some of the authentication assertions are used in policies elsewhere there will be no additional example here.

14. Summary

In the Note I discussed web services security topics to set the context, discussed a Java CAPS Repository-based web services solution, walked through creation of the service provider and service consumer, configured the SecureSpan XML Gateway and the SecureSpan VPN Client infrastructure and walked through a series of iterations of policy creation and testing.

There is a great deal more to the SecureSpan XML Gateway. The discussion in this Note merely introduces some of the kinds of solutions possible with the aid of a gateway and mentions only some of the benefits of using a gateway-mediated web services security solutions, as distinct form developer-mediated ones.

15. Appendices

15.1 *Obtain the Layer 7SecureSpan XML Gateway*

Layer 7 Technologies, http://www.layer7tech.com, the maker of the Layer 7 SecureSpan XML Gateway and associated software, offer a VMware image of the SecureSpan XML Gateway for a time-limited trial. To request a trial license and software download, go to http://www.layer7tech.com/xmltrial.html, register and wait to be contacted.

Once you submitted your request, an acknowledgement similar to the one shown in Figure 15-1 will appear.

Figure 15-1 Acknowledgement of trial request

Within a business day, or so, you will receive an email, similar to that shown in Figure 15-2.

Dear Michael,

I would like to thank-you for your interest in Layer 7 Technologies and your request for an XML Trial.

I would also like to arrange a time to introduce you to Layer 7 Technologies a leading provider of security, management and governance products for SOA and Web services.

Our products are designed to address many of the performance, policy, virtualization and identity challenges associated with deploying production SOAs. We are used by leading public sector organizations including numerous agencies responsible for Defense, Intelligence, Healthcare, Treasury, Agriculture and Justice to name some and have been recognized by numerous magazines and analyst firms for our leadership and vision in the space.

If you are undertaking an SOA or Web services initiative we'd welcome an opportunity to better introduce our capabilities and how this experience may be of benefit to your needs.

Let me know when would be a good time to schedule an introductory call to further discuss your request. I would be available to speak to you after 7:00am until 11:00am your local time today.

I look forward to hearing from you.

Best regards,

Figure 15-2 Example email follow up

Once you establish contact with a Layer 7 representative, you will be asked a few questions intended to determine how best to satisfy your request. Depending on which part of the World you live in you may be put in touch with a Layer 7 partner, who will be expected to set you up with the download link and provide you a trial license for a 15 or 30 day trial, and perhaps assist you with the trial. If you eventually decide to invest in the product, this Layer 7 partner will most likely be helping you out with the planning and implementation.

When the trial request is processed you will receive an email much like the one shown in Figure 15-3. The URL and credentials will be specific to you.

Michael,

Thank you for your interest in evaluating the SecureSpan Gateway. Below you will find a URL for downloading the SecureSpan Gateway
"Soft-Appliance", documentation, evaluation license key, and a brief presentation to help get you started:

URL: http://www.layer7tech.com
Username:
Password:

The SSG VMware User Manual has step by step instructions on how to get everything up and running. A typical configuration process takes only a few minutes, but it may take a bit longer the first time through.
In most cases the default values presented should be fine.

Please respond to this email if you have any difficulty downloading the file.

Sincerely,

Layer 7 Technologies

Figure 15-3Download and license link email

Following the link, and logging in with the credentials provided, will lead to a page containing download links for the SecureSpan XML Gateway Appliance VMware image, the SecureSpan XML Gateway Manager and product documentation.

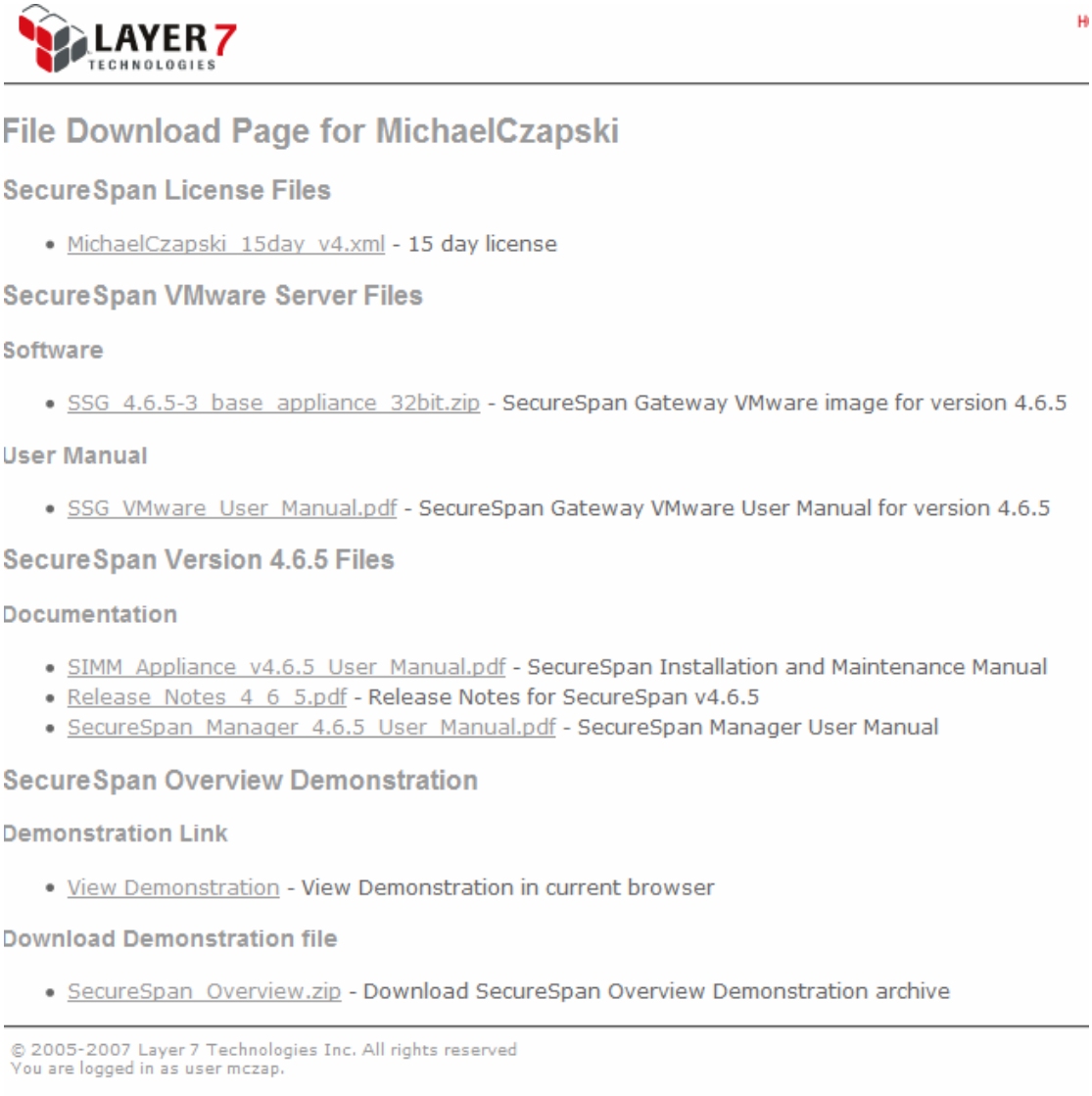

Figure 15-4 SecureSpan XML Gateway trial download page

Note that the download page does not list the SecureSpan VPN Client, which will be required to complete the end-to-end scenario discussed in this Note. I asked Layer 7 Support for the VPN Client distribution and the download page was modified to include the links for the SecureSpan VPN Client and the Manual that goes with it.

If you don't see the VPN Client then you will need to ask for it.

I encourage you to view the SecureSpan Overview presentation to get a good feel for the purpose of the tool and the kinds of issues they are intended to address.

Unzip the archives to the convenient directories and get familiar with the instructions in the "SSG_VMware_User_Manual.pdf". See section 15.2, "Configure the SecureSpan XML Gateway", for a discussion and additional instructions.

15.2 *Configure the SecureSpan XML Gateway*

To use the VMware Image of the SecureSpan XML Gateway appliance you will need a WMware Player, a VMware Workstation or a VMware Server. VMware Player,

downloadable from http://www.vmware.com/download/player/, is free to download and use.

SecureSpan XML Gateway version 4.6.5-3, VMware Image distribution, comes as a ZIP archive with the name of SSG_4.6.5-3_base_appliance_32bit.zip. You will need on the order of 3Gb of disk space to unzip and start the appliance. The appliance requires 768Mb of memory to be allocated to it so you will need at least 1.5Gb of real memory in the Host.

Unzip the archive to a convenient directory and start the appliance by, on Windows, double-clicking the SSG_32bit_VirtualAppliance_v46.5.vmx VMware configuration file.

For these who care, the important bits of VMware configuration are shown in Figure 15-1.

Figure 15-5 Important VMware configuration settings

```
memsize = "768"
guestOS = "rhel4"
ethernet0.connectionType = "bridged"
scsi0:0.present = "TRUE"
scsi0:0.fileName = "SSG_32bit_VirtualAppliance_v46.5.vmdk"
```

Consult instructions in the “SSG_VMware_User_Manual.pdf”, which was provided as part of the trial download, to start and configure the Layer 7 SecureSpan XML Gateway Appliance – see also notes below.

When you login as a ssgconfig or a root user you will be forced to change the password. The password rules are fairly restrictive. I found that a password like **L7.App4y0u&me** works for me.

On first boot I was asked whether I wish to remove the network adapter and, when I said Yes, to configure a new network adapter, to which I said Yes as well. In effect, by default the RedHat Linux distribution appears to be trying figure out what kind of network interface the guest has and configure it.

Login as ssgconfig user and configure networking – option 1. When done reboot the machine - option 5.

When running the option 1 network configuration I provided a FQDN of ssg.aus.sun.com. I found that after reboot the hostname and FQDN were not correctly configured and Option 2 did not work for me. This may or may not happen to you. I fixed my problem by adding directive “domain” to the /etc/resolv.conf, specifying the domain name I was using and getting rid of the “search” directive. Your mileage may vary. At any rate, this is not something I can help with. If you cannot get the ssgconfig to work you may need to contact Layer 7 support for assistance.

Login as ssgconfig user and run option 2.
When running option 2, SecureSpan Configuration Wizard, one is asked, at a point in the process, to provide a root password – this does not seem to be the Unix root user’s

password. Pressing Enter without specifying root password worked for me. When asked for a SSL Keystore password I provided “sslkey”

Reboot the appliance (option 5) when finished.

If you needed to modify your resolv.conf so that your domain name was properly recognised, see earlier, you will need to repeat the process after running option 2 because the resolv.conf is re-generated so your changes got lost.

Once the appliance is up and running again you can perform a ping test using a web browser. Start a web browser and enter the URL https://ssg:8443/ssg/ping
A security warning will show up the first time around. I used the Mozilla Firefox so my warning looked like that shown in Figure 15-6.

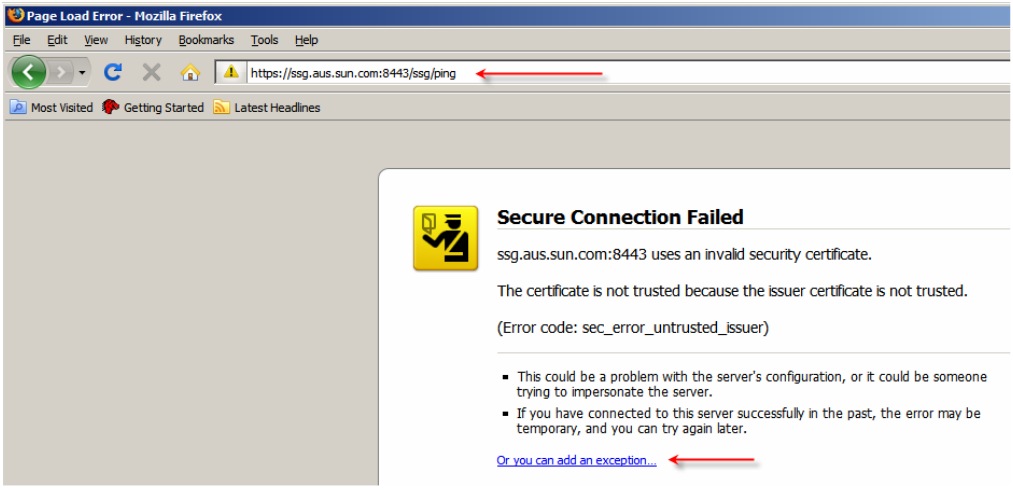

Figure 15-6 Security warning in Mozilla Firefox

I created and exception. Once done, I was asked for a username and password (HTTP BASIC Authentication). Having not created any users so far I provided “root” as the username and Unix root’s password as the password. I received a blank page with the words “503 Unlicensed”. This is not unexpected as I have not yet provided the license key to the Gateway. At any rate, the gateway is up and talking to me, however rudely.

Login to the SecureSpan WebAdmin Console using a Web Browser with a URL like: https://ssg:8443/ssg/webadmin. Provide **admin** as the username and **password** as password. Once you are through the several security-related dialogue boxes you will be told that there is no license installed, Figure 15-7, as is to be expected since it has not been installed yet.

Figure 15-7 No license

Click Yes and follow the prompts to locate and install the license key XML file which you should have received with your trial download links.

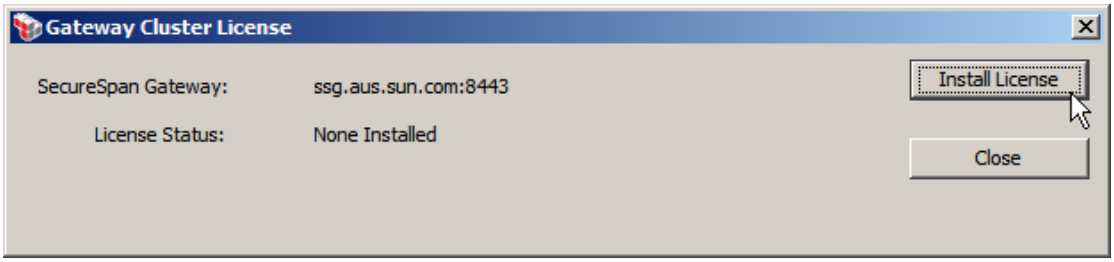

Figure 15-8 Click Install License

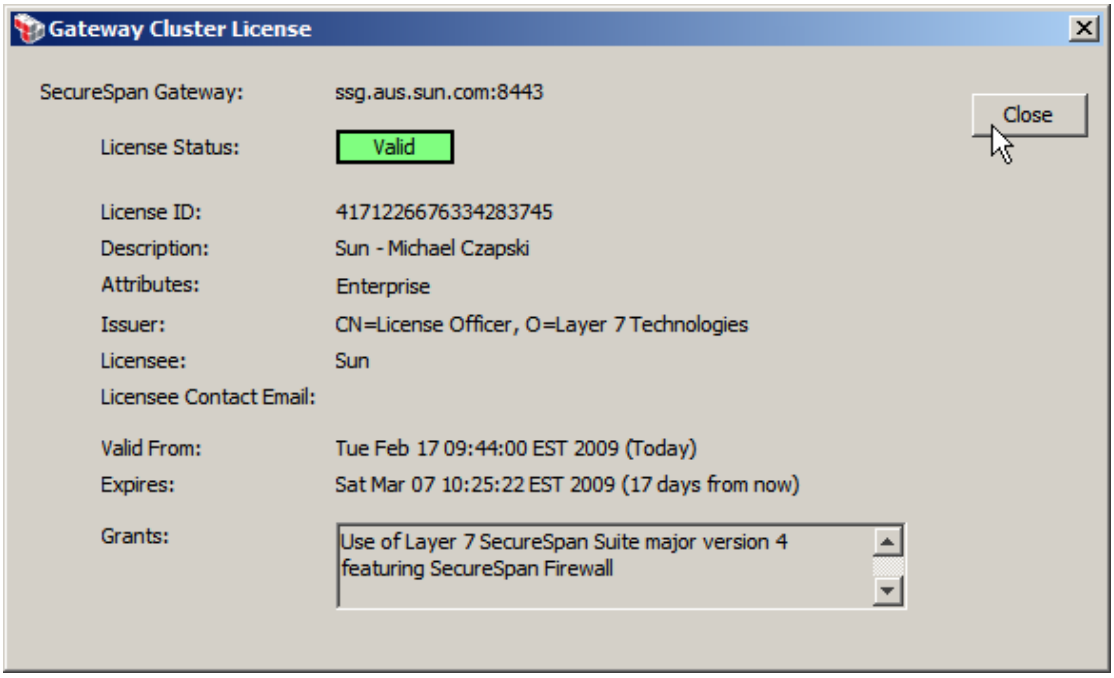

Figure 15-9 Acknowledge license installation

When done, the browser window will display the SecureSpan Manager Home page, much as is shown in Figure 15-10.

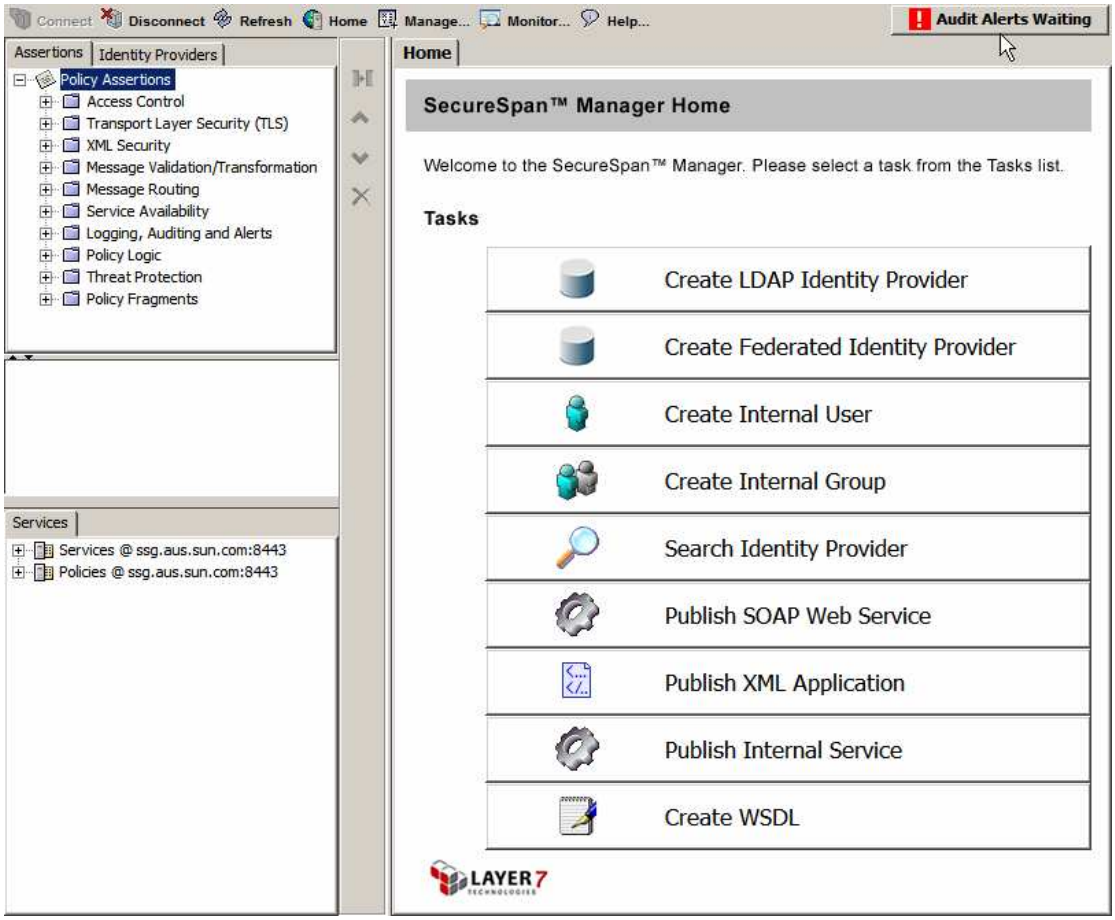

Figure 15-10 SecureSpan Manager

Exit the browser and start it again. Connect again to the Gateway to start the Gateway Manager. You will be asked to change your password from “password” to something else. Beware, the password rules for this password are different form the password rules used to set the root password on Unix. The password I suggested for the Unixs root and ssgconfig accounts, L7.App4y0u&me, will not work because one of the rules says “can’t have two or more the same consecutive characters”. I dropped one “p” from the password and it worked so my Gateway Manager uses the password of “L7.Ap4y0u&me”.

15.3 *Install and Configure the SecureSpan VPN Client*

We will have two cooperating “sites”. One site will use the SecureSpan XML Gateway and the other site will use the SecureSpan VPN Client. Both sites could use the Gateway but that would require more resources then my notebook can provide so only one Gateway will be installed.

If you are on a Unix platform, which includes Linux platforms, have read of the appropriate section in Chapter Six of the Installation and Maintenance Manual (Appliance Edition).

You should have SecureSpan VPN Client distribution downloaded. I use the one for Windows, which comes as a Windows Installer named “SecureSpan XML VPN Client 4.6.5 Installer.exe”.

My environment for this Note is Microsoft Windows so anything I write about, that has an OS dependency, pertains to Windows.

Start the installer in the usual Windows manner. Accept the License Agreement, nominate the target directory, and click Install. You will be asked whether you would like the VPN Client to run as a Windows Service – I chose to not install it as a service.

When the installation is finished you will get an entry, in your programs list, pointing to the SecureSpan XML VPN Client user interface. Figure 15-13 illustrates this.

Figure 15-11 SecureSpan XML VPN Client shortcut

Start the VPN Client. Nothing apparent will happen. The SecureSpan XML VPN Client will show up in the System Tray, as shown in Figure 15-12.

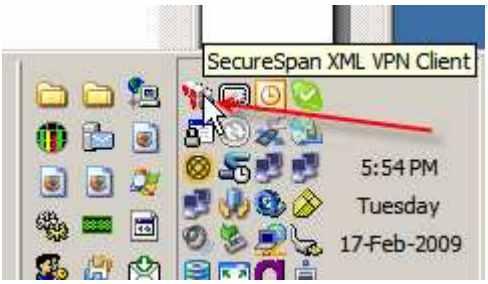

Figure 15-12 VPN Client in System Tray

Click on the tray icon to make the UI appear, see Figure 15-13.

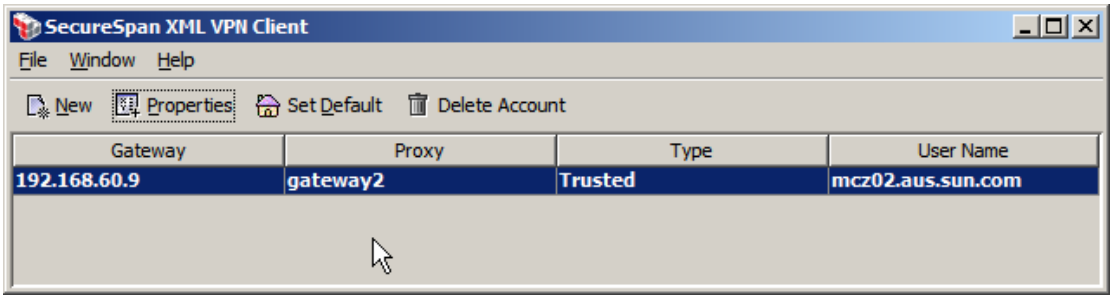

Figure 15-13 SecureSpan VPN Client UI

Click the New button.

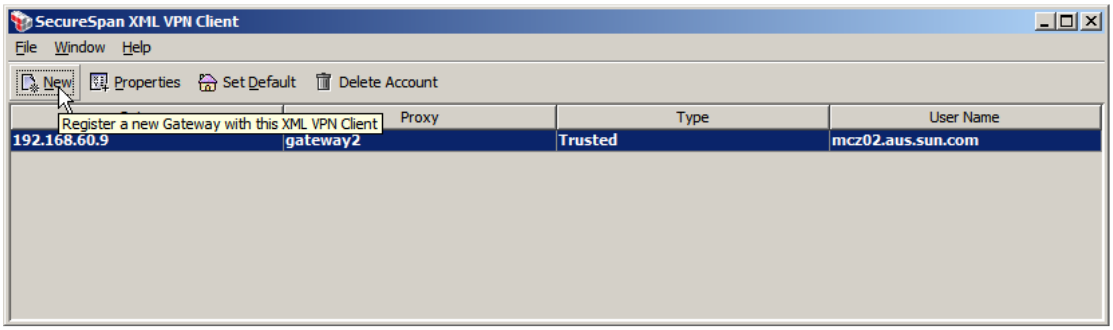

Figure 15-14 Register new Gateway

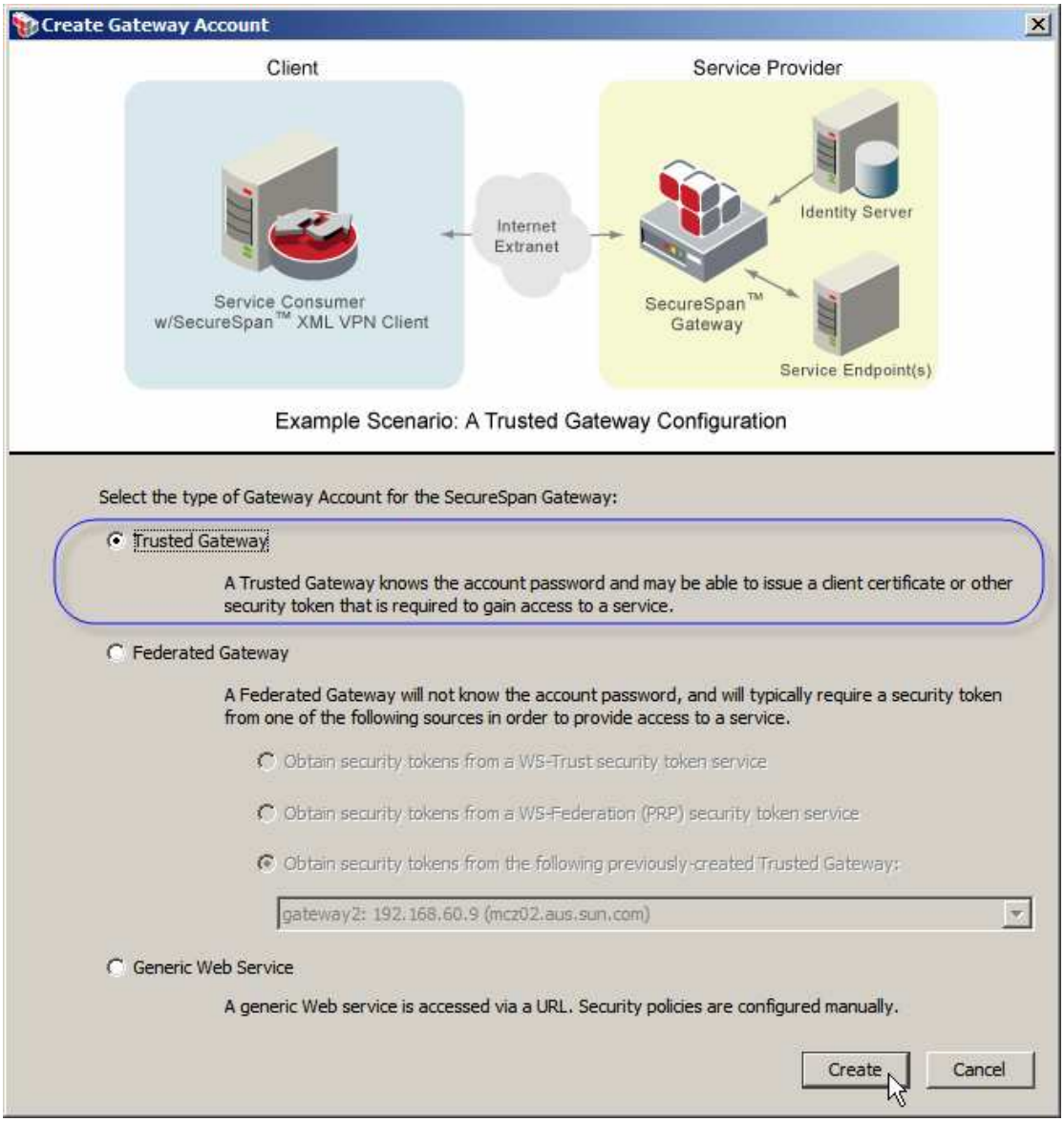

Figure 15-15 Choose Trusted Gateway

Provide the gateway name/address as specified when you configured the SecureSpan Gateway – best specify FQDN.

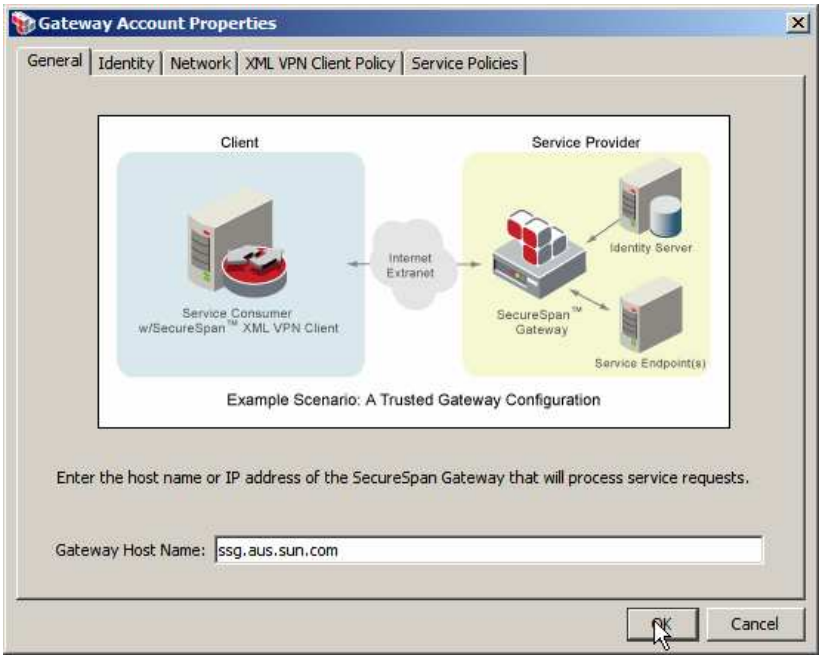

Figure 15-16 Provide Gateway Name/Address

You should see the gateway registered. If you have any other gateway, like I do, select and delete it.

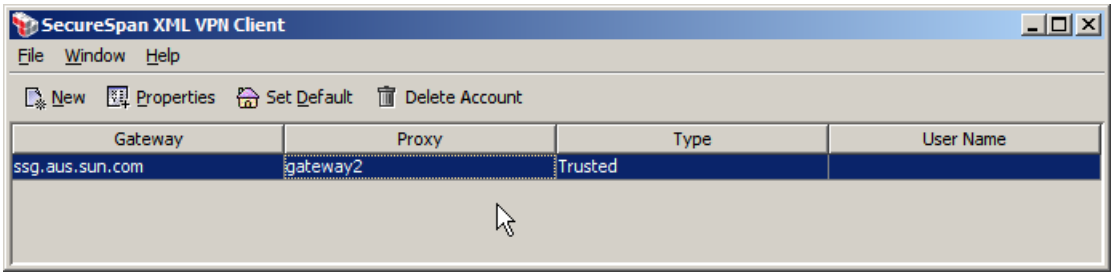

Figure 15-17 Gateway registered

We will configure the VPN Client in the most straight forward manner possible, with just one Trusted Gateway Account, set as a default account.

Pull down the File Menu and click Properties. Gateway Account Properties Wizard, open on General Tab, should appear. This is shown in Figure 15-16. Click on the Identity Tab, provide username “admin” and password “password”, then check the Save Password checkbox. Figure 15-18 illustrates this. If you have accessed the SSG Manager a couple of times using the admin user, you will have been forced to change the password. If this is what happened then provide the correct password here. Mine is “L7.Ap4y0u&me”.

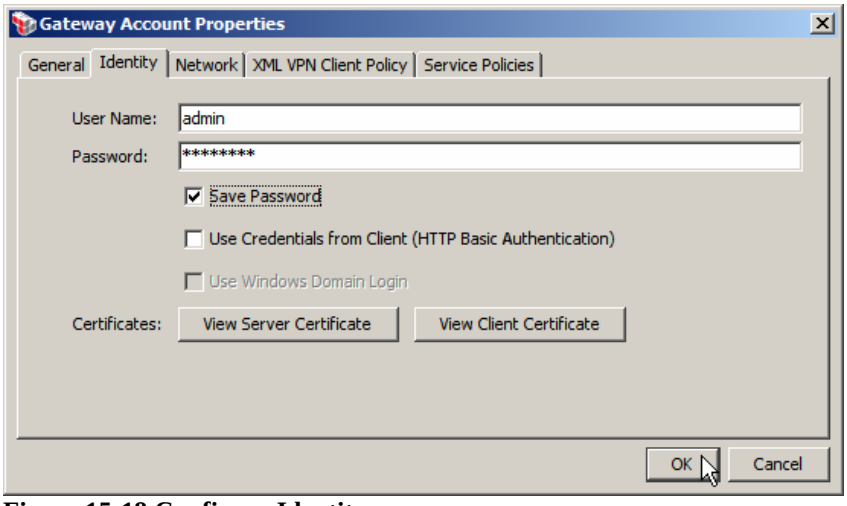

Figure 15-18 Configure Identity

Click on the Network Tab.

Take note of the "Proxy URL" – by default http://localhost:7070/*gatewayn*, where *gatewayn* is the name in the proxy column in the list of gateway accounts in the General Tab.

Take note of the standard gateway ports, by default 8080 and 8443. Figure 15-19 highlights the items of specific interest.

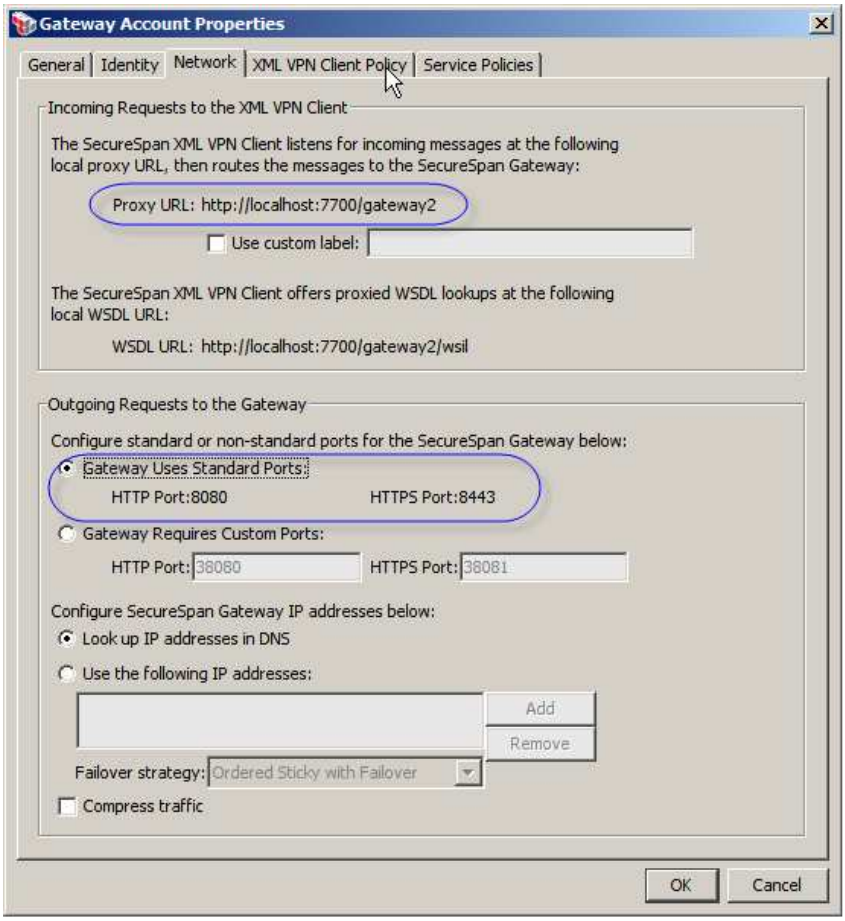

Figure 15-19 Network Tab – default setting

Switch to XML VPN Client Policy Tab and uncheck the “Use SSL by Default” checkbox, see Figure 15-20. We would like to snoop on the wire.

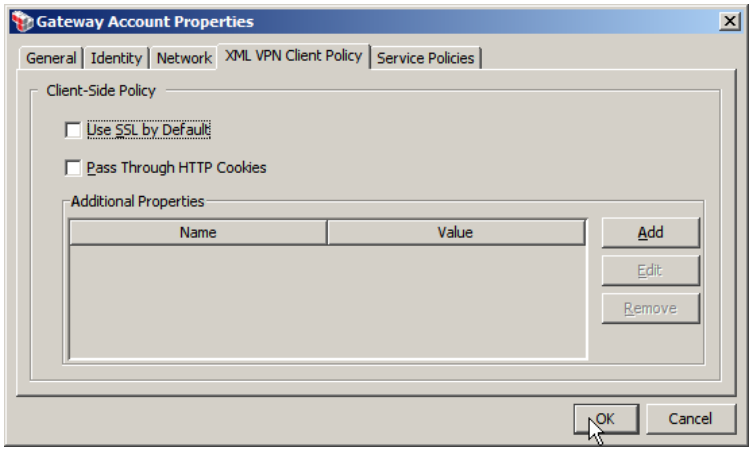

Figure 15-20 Uncheck the “Use SSL by Default” checkbox

When done, click OK to close the properties.
You can exit from the VPN Client UI. When you exit the VPN Client it will disappear from the System Tray.

15.4 *Install certificates for the VPN Client and the Gateway*

All configurations that use encryption or digital signing require the use of X.509 Certificates and Private Key Keystores. In this section we will add a Certification Authority (CA) Certificate and two End-Use Certificates/Key Stores for use in examples. Note that all of these objects are objects I created and should not be used for anything except experimentation.

The zip archive, mcz_ssg_certs_and_keys.zip, contains a number of objects, some of which are certificates and some of which are keystores, see Figure 15-21. The archive is available for download from http://mediacast.sun.com/users/Michael.Czapski-Sun/media/mcz_ssg_certs_and_keys.zip/details.

Figure 15-21 Contents of the archive

```
7-Zip 4.42  Copyright (c) 1999-2006 Igor Pavlov  2006-05-14

Listing archive: mcz_ssg_certs_and_keys.zip

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------
2009-02-21 09:46:18 ....A          441          269  ssgcli\ssgcli.conf
2009-02-21 09:46:46 ....A          870          707  ssgcli\ssgcli.der.crt
2009-02-21 09:46:46 ....A         1658         1652
ssgcli\ssgcli.eXchange.pkcs12.keystore.p12
2009-02-21 09:47:14 ....A         2351         1800  ssgcli\ssgcli.jks.keystore
2009-02-21 09:46:28 ....A         1233          874  ssgcli\ssgcli.pem.cer
2009-02-21 09:46:48 ....A         1800         1293  ssgcli\ssgcli.pem.cer.stunnel
2009-02-21 09:46:28 ....A         1233          874  ssgcli\ssgcli.pem.crt
2009-02-21 09:46:18 ....A          655          485  ssgcli\ssgcli.pem.csr
2009-02-21 09:46:18 ....A          561          448  ssgcli\ssgcli.pem.private.key
2009-02-21 09:46:44 ....A         3615         2045  ssgcli\ssgcli.pem2.crt
2009-02-21 09:46:46 ....A         2684         2678  ssgcli\ssgcli.pkcs12.keystore.p12
2009-02-21 09:47:14 D....            0            0  ssgcli
2009-02-21 09:45:08 ....A          441          271  ssgtwy\ssgtwy.conf
2009-02-21 09:45:32 ....A          870          705  ssgtwy\ssgtwy.der.crt
2009-02-21 09:45:38 ....A         1658         1654
ssgtwy\ssgtwy.eXchange.pkcs12.keystore.p12
2009-02-21 09:46:00 ....A         2351         1799  ssgtwy\ssgtwy.jks.keystore
2009-02-21 09:45:24 ....A         1233          876  ssgtwy\ssgtwy.pem.cer
```

```
2009-02-21 09:45:38 ....A         1800         1294  ssgtwy\ssgtwy.pem.cer.stunnel
2009-02-21 09:45:24 ....A         1233          876  ssgtwy\ssgtwy.pem.crt
2009-02-21 09:45:10 ....A          655          484  ssgtwy\ssgtwy.pem.csr
2009-02-21 09:45:10 ....A          561          447  ssgtwy\ssgtwy.pem.private.key
2009-02-21 09:45:32 ....A         3615         2044  ssgtwy\ssgtwy.pem2.crt
2009-02-21 09:45:34 ....A         2684         2682  ssgtwy\ssgtwy.pkcs12.keystore.p12
2009-02-21 09:46:00 D....            0            0  ssgtwy
2006-12-19 21:43:44 ....A         1399         1028  DemoCA.pem.crt
2006-12-19 21:43:44 ....A          991          781  DemoCA.der.crt
2006-12-19 21:43:44 ....A         4236         2363  DemoCA.pem2.crt
------------------- ----- ------------ ------------  ------------
                                 40828        30429  27 files
```

Objects with the file extensions of “cer”, “crt” and “crt.stunnel” are X.509 Certificates. Objects with the file extensions of “pkcs12.keystore.p12” and “jks.keystore” are keystores – in PKCS#12 and Java KeyStore formats respectively. Keystores are password-protected. Passwords are “ssgtwyssgtwy” for “ssgtwy.pkcs12.keystore.p12” and “ssgtwy.jks.keystore”, and “ssgclissgcli” for “ssgcli.pkcs12.keystore.p12” and “ssgcli.jks.keystore”. Both keystore types contain the same private key. The reason there are two is that some cryptographic tools use the PKCS#12 keystores and others use the JKS keystores. I have a set of scripts I developed years ago, which produce all the different forms at in the same session so it is more trouble for me to remove the ones I don’t use then to keep all of them.

The Certification Authority (CA) certificate, used to sign the end-use certificates for ssgtwy and ssgcli, is DemoCA. DemoCA’s certificate is in the DemoCA.*.crt. All pem, pem2 and der are different encodings of the same certificate. Use whichever comes first. PEM format is a Base64-encoded version of the DER format. If asked to paste the certificate, paste the content of the PEM format certificate.

Unzip the content of the archive to a convenient directory.

15.4.1 Add Certificates and Keys to the Gateway store

Pull down the Manage menu of the SSG Manager and choose Manage Certificates. Figure 15-22 illustrates this.

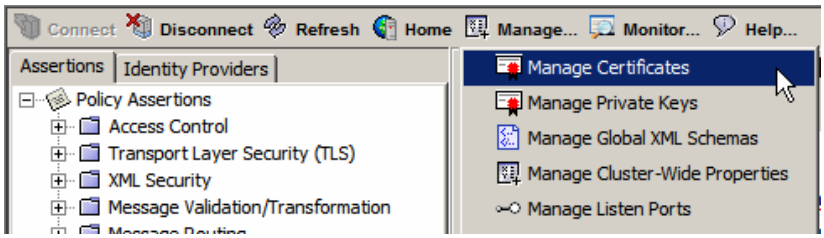

Figure 15-22 Activate the Manage->Manage Certificates functionality

Click the Add button, select the Import from a File option and Browse to where the certificates are, see Figure 15-23.

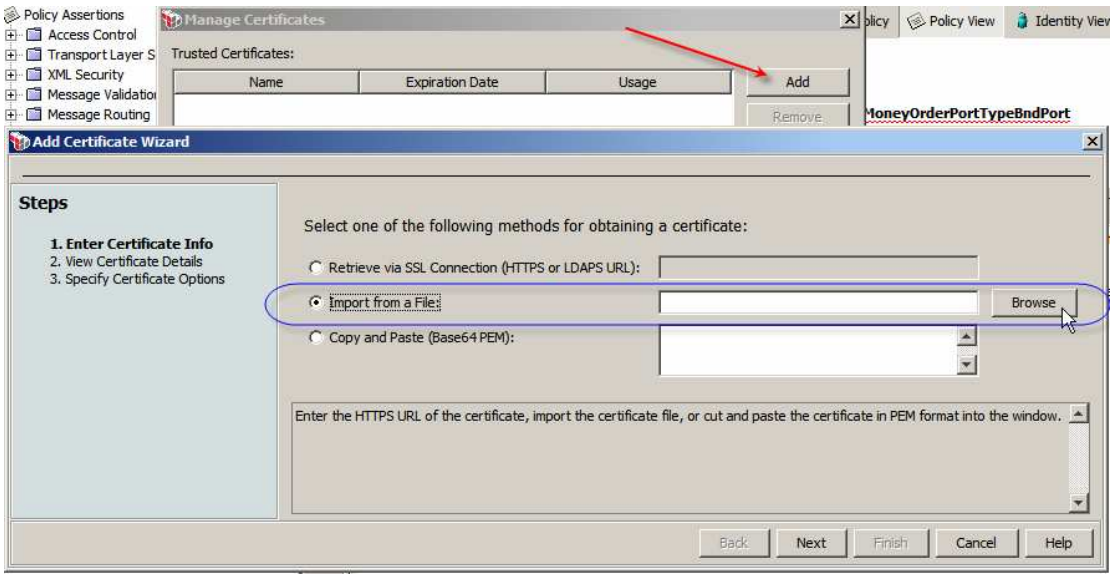

Figure 15-23 Start the certificate import process

Pick DemoCA.crt, click Open, Figure 15-24, and click Next.

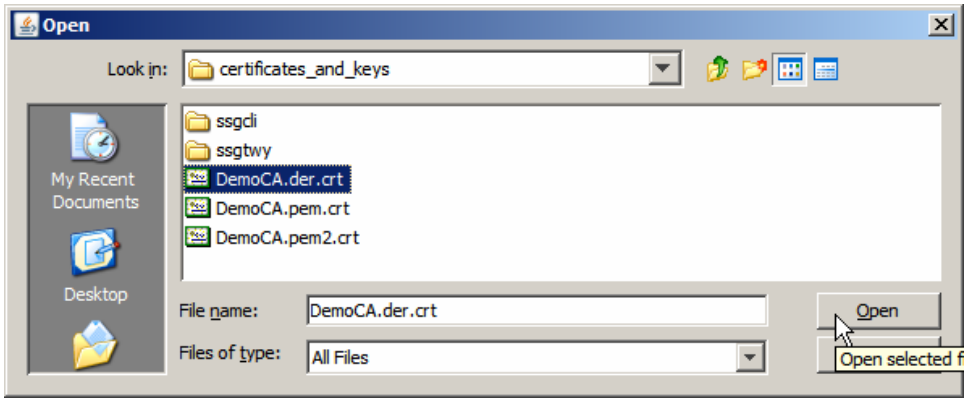

Figure 15-24 Pick the CA certificate

The Certificate Details panel will appear, Figure 15-25. Click Next.

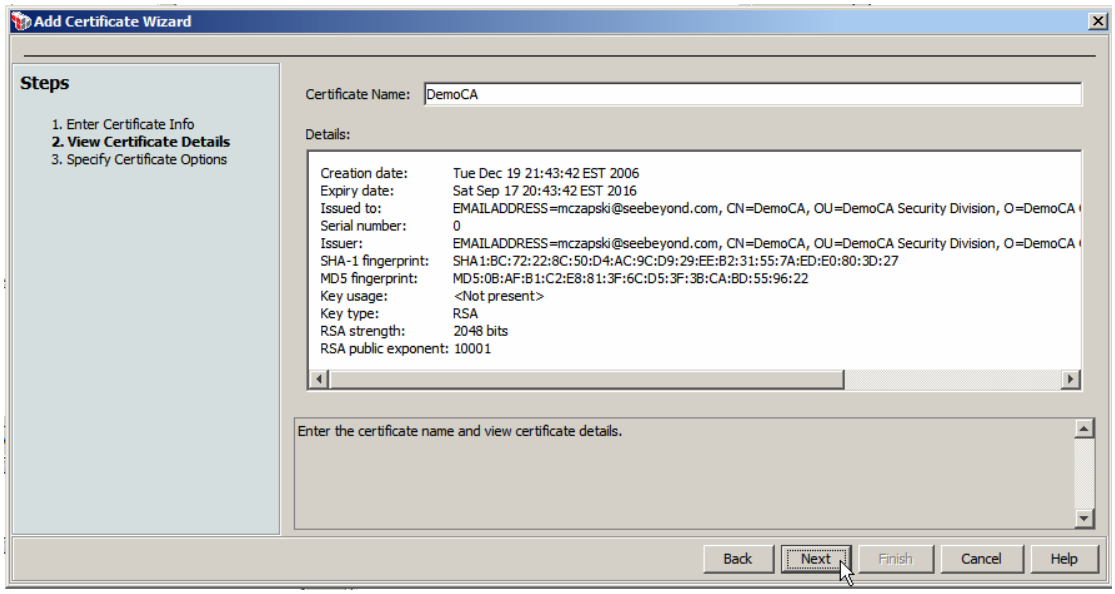

Figure 15-25 Certificate details

Check the Signing Certificates … checkboxes and click Finish, see Figure 15-26.

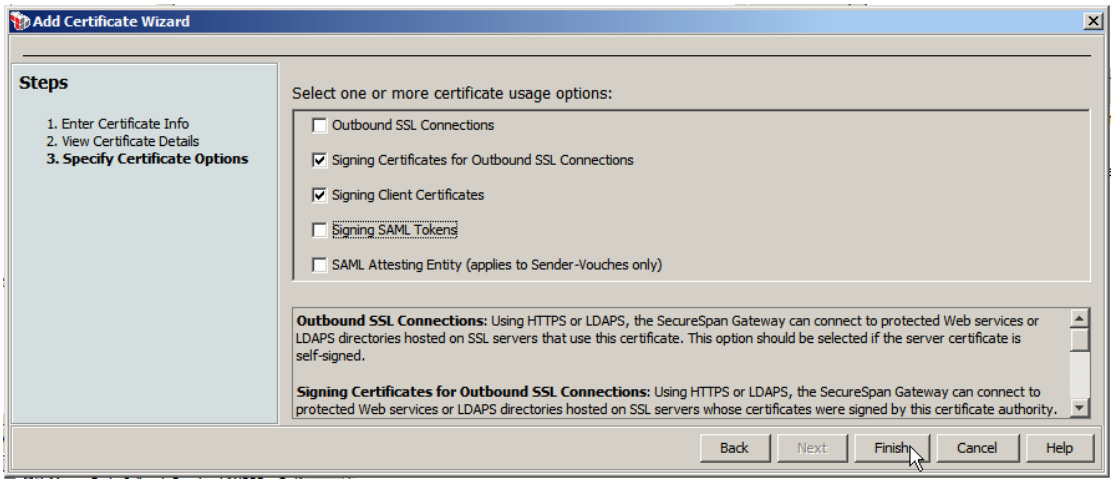

Figure 15-26 Select certificate usage options

Repeat the steps to import the “ssgcli” certificate, the certificate that will represent the SecureSpan VPN Client. Select the other three usages as shown in Figure 15-27.

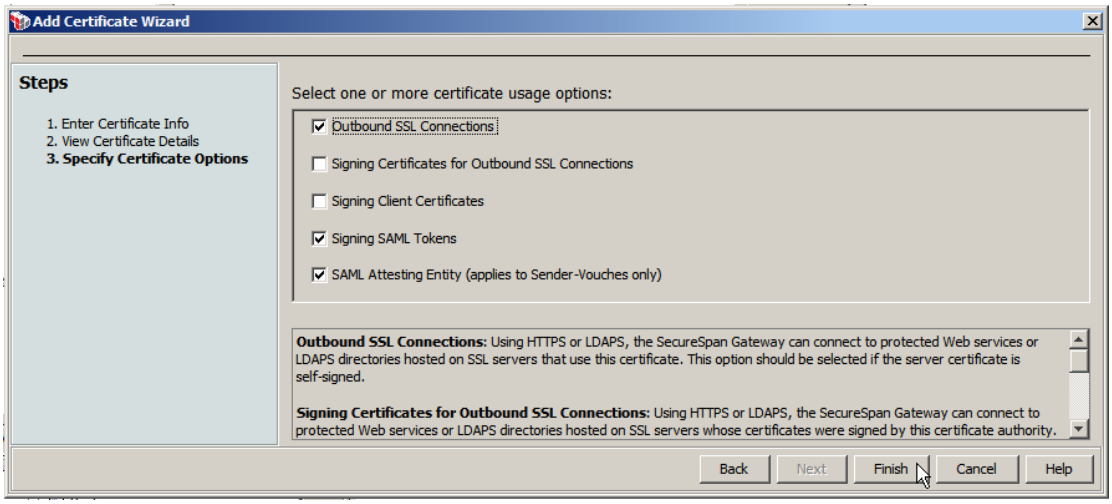

Figure 15-27 Select usages for the VPN Client certificate

The “ssgcli” certificate will be used by the Gateway to validate ssgcli’s digital signatures over messages received from it, and to encrypt messages bound for the ssgcli, the VPN Client. The DemoCA Certificate is used to validate certificates which were issued and signed by the DemoCA. Both ssgcli and ssgtwy certificates were issued by the DemoCA. Finish import by clicking the close button, Figure 15-28.

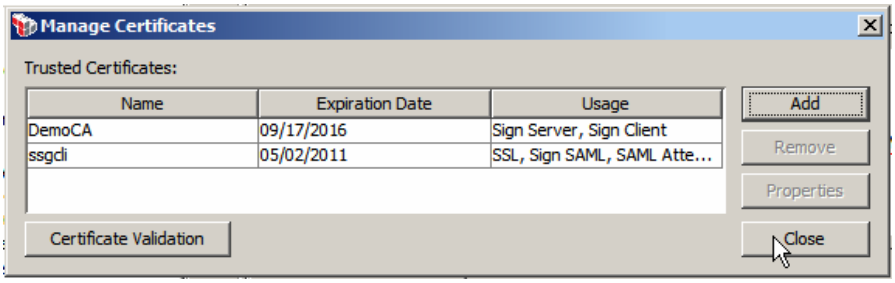

Figure 15-28 List of certificates in the Gateway’s store

Now let's add the "ssgtwy" private key, residing in the PKCS#12 keystore. The ssgtwy private key will be used by the Gateway to digitally sign outbound messages and to decrypt inbound messages.

In the SSG Manager pull down the Manage drop down and select Manage Private Keys. Click the Import button, locate the ssgtwy.pkcs12.keystore.p12 file, select it, enter the password "ssgwtyssgtwy", enter "ssgtwy" as an alias, click OK and click Close. Figure 15-29 shows the new private key imported into the Gateway store.

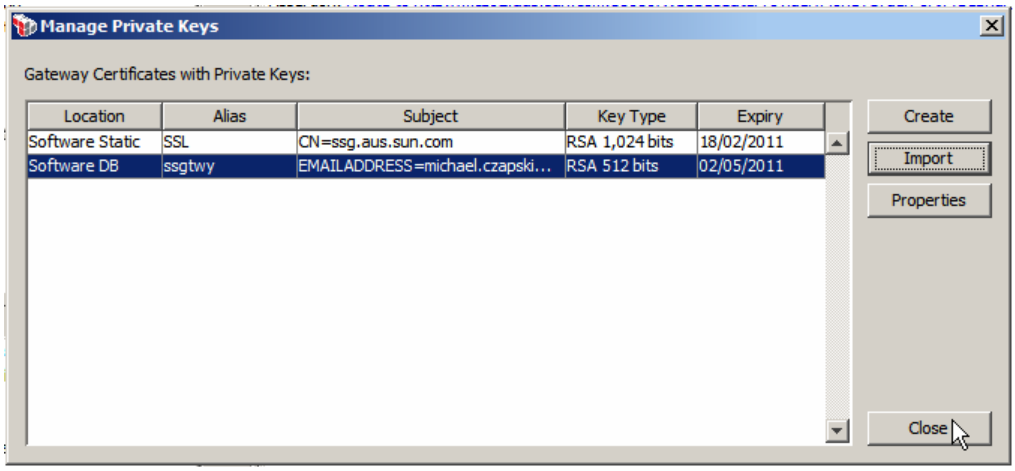

Figure 15-29 Begin Import of the ssgtwy private key

The Gateway now has the certificate that will be used by the VPN Client, the private key the Gateway will use and the CA Certificate, which will be used to validate certificates issued by it.

15.4.2 Add Private Key and Certificate to the VPN Client store

Switch to the VPN Client UI. Click Properties, switch to Identity Tab and click the View Client Certificate button, click the Import Client Certificate button, Figure 15-30.

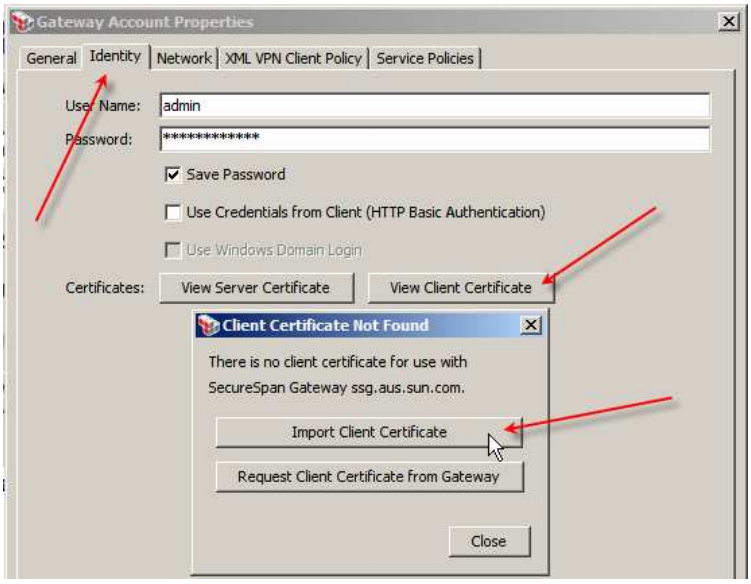

Figure 15-30 Start the Import certificate wizard

Locate the PKCS#12 Keystore, which is what the wizard is actually looking for, select it and click the Import Certificate button, Figure 15-31. There is a bit of confusion of terms here. PKCS#12 Keystore, containing the private key, is a different object from the X.509 Certificate, but never mind. The PKCS#12 keystore contains both the certificate and its corresponding private key.

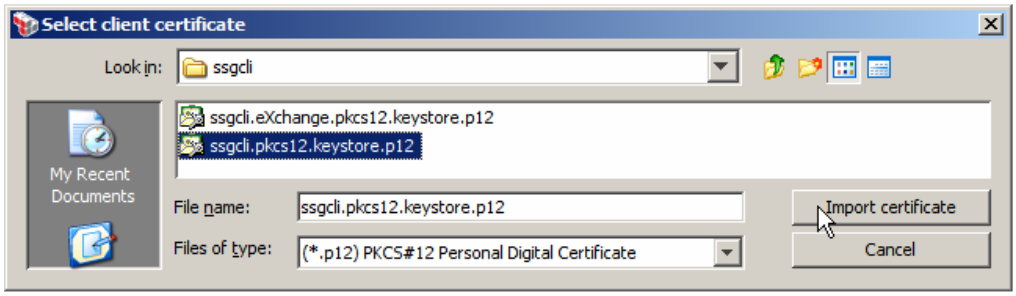

Figure 15-31 Select and import the PKCS#12 Keystore

Specify “ssgclissgcli” for the password and click OK. Successful import will be conferment, see Figure 15-32.

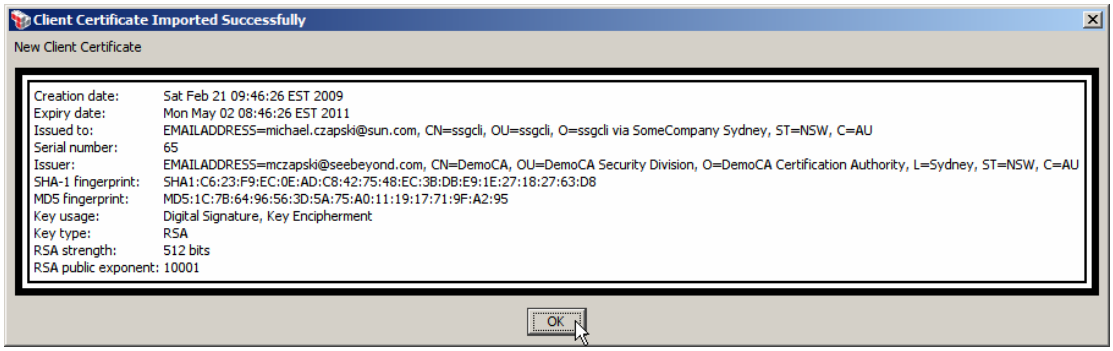

Figure 15-32 Successful import of the Keystore

Done. The VPN Client now has cryptographic objects, some of which it will need for digital signing of outbound messages and decryption of inbound messages.

15.5 *Create Java CAPS Environment*

Create a Java CAPS Environment, WSSecGateEnv, as illustrated in Figures Figure 15-33 through Figure 15-36.

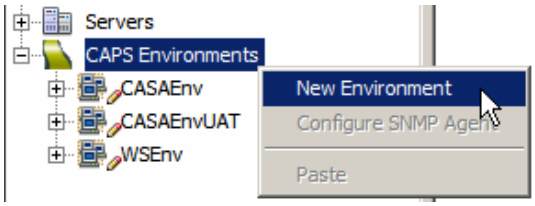

Figure 15-33 Create Java CAPS Environment

Add a Logical Host. Add a Sun Java System Application Server and a Sun Java System Message Queue. See Figure 15-34 for the menu options to use.

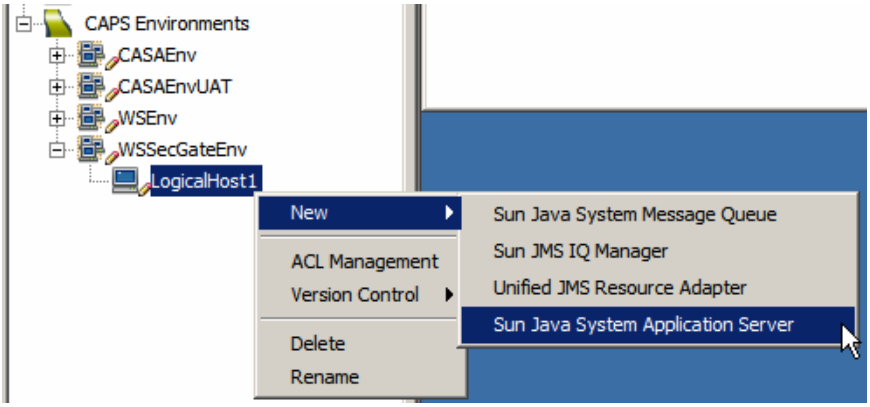

Figure 15-34 Add an Application Server to the Environment's Logical Host.

Set the properties for both to provide authentication credentials, host names and port numbers that reflect your environment. When configuring properties for the JMS Message Server, provide the Sun JMQ Server URL that includes the redelivery handling incantation. For me the URL looks like that shown in Figure 15-35.

Figure 15-35 JMQ URL with redelivery handling incantation

```
mq://localhost:37676/?JMSJCA.redeliveryhandling=1:move(same:$DLQ)
```

Add a new UDDI External System container and modify its properties to reflect your environment. Part of the process is illustrated in Figure 15-36.

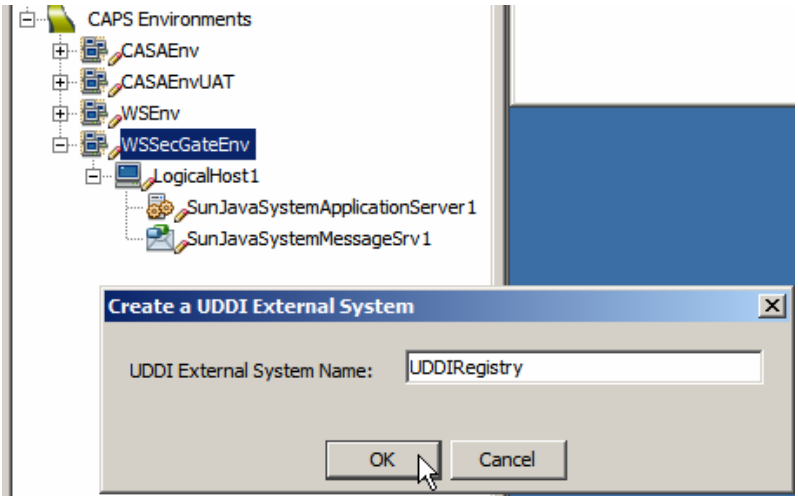

Figure 15-36 Add new UDDI External System

We will add and configure Web Services Client and Server External System containers as we go along in solution development.

15.6 *Obtain and use the Apache TCP Mon*

One of the issues with SOAP decoration, which is what MTOM and others do, is that the on-the-wire message looks different from what the sending and the receiving applications see. It is very hard to make the application server and the client log what they are sending and receiving and even then there is a good chance that what is logged differs from what is sent / received. To see what is really exchanged a wire snooper of some sort is required.

Apache TCP Mon, see http://ws.apache.org/commons/tcpmon/index.html, can be used as a convenient proxy to view the on-the-wire messages exchanged between web services invokers and providers. Download the TCP Mon from the site. Tutorial at http://ws.apache.org/commons/tcpmon/tcpmontutorial.html has a nice explanation of the usage modes.

To start the TCP Mon from the command line in a direct intermediary mode with specific host and port configuration one could say (on Windows):

```
C:> cd C:\tools\tcpmon-1.0-bin\build
C:> tcpmon.bat 38081 localhost 38080
```

This will start the TCP Mon with the listening port 38081, relaying messages to port 38080 on localhost.

```
C:> cd C:\tools\tcpmon-1.0-bin\build
C:> tcpmon.bat 8888
```

This will start the TCP Mon as a proxy listening on port 8888. One needs to configure one's client to use the proxy.