LAB 1– WS-Security Examples

Service Oriented Architectures Security

Module 5 - Lab

Unit 1 – Security

Fulvio Frati

Università di Milano

Outline

- **What is WS-Security**

- **A simple example**
  - 1st step: no security
  - 2nd step: timestamp
  - 3rd step: signature
  - 4th step: full security (timestamp, signature, encryption)

What is WS-Security?

WS-Security:

- soap message protection through message integrity, confidentiality, and single message authentication

- extensible and flexible (multiple security tokens, trust domains, signature formats, and encryption technologies)

- a flexible set of mechanisms that can be used to construct a range of security protocols

Why WS-Security?

- **Implement secure soap message exchange**

How to Secure?

Integrity - information is not modified in transit

- XML signature in conjunction with security tokens
- Multiple signature, multiple actors, additional signature formats

How to Secure?

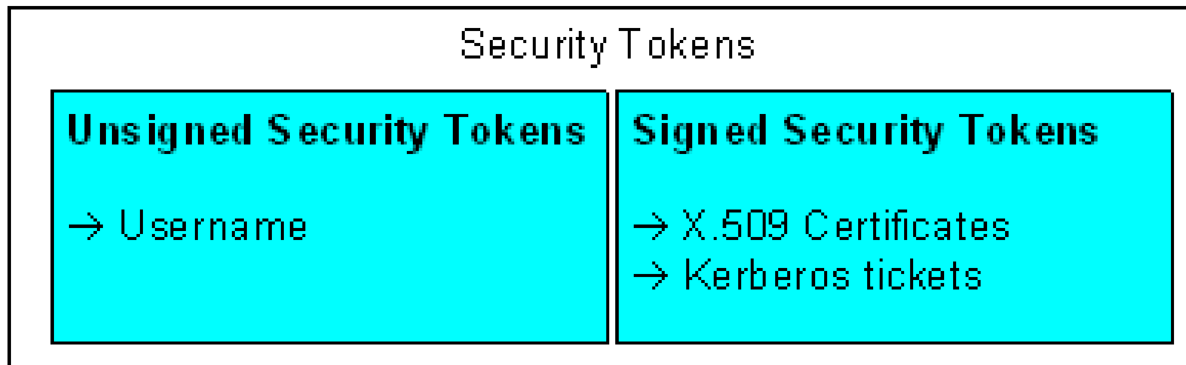
Confidentiality - only authorized actors or security token owners can view the data

- XML encryption in conjunction with security tokens
- Multiple encryption processes, multiple actors

How to Secure?

Authentication – you are whom you said you are

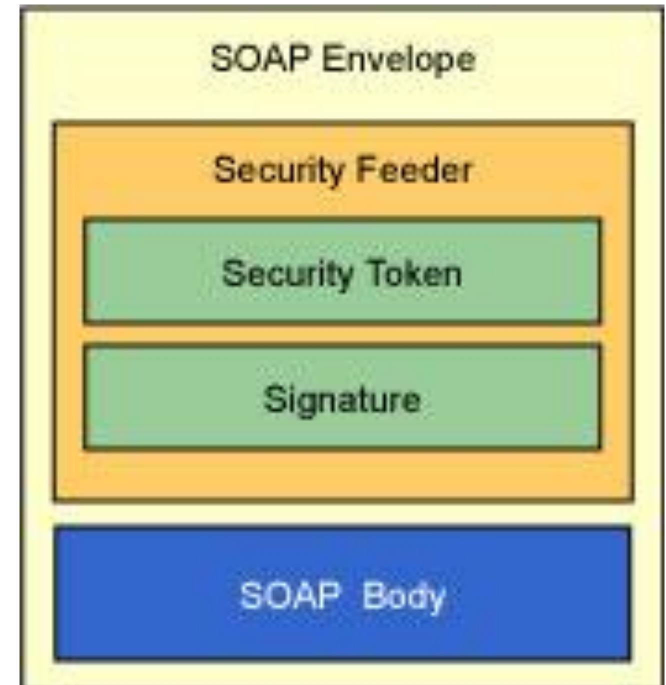
- Security Tokens



Syntax

```
<S:Envelope>
  <S:Header>
 ...
    <Security
    S:actor="..." S:mustUnderstand="...">
 ...
    </Security>
 ...
 </S:Header>

 <S:Body>…
 </S:Body>
</S:Envelope>
```



Setting the Stage

What we need:

- Web Container: **Apache Tomcat**
  http://tomcat.apache.org/

- Web Services / SOAP / WSDL engine: **Apache AXIS2**
  http://axis.apache.org/axis2/java/core/

- Web Services security module: **Apache Rampart**
  http://axis.apache.org/axis2/java/rampart/

- TCP SOAP Messagges monitor: **Apache TCPMon**
  http://ws.apache.org/commons/tcpmon

Tutorial Architecture

- Simple Service for adding two numbers

- Four incremental security steps:
  1. No security
  2. Timestamp only
  3. Signature only
  4. Full security: Timestamp + Signature + Encryption

- Modifications overall on configuration files

Service Code

```
/**
 * Secure Service implementation class
 */
public class SecureService {
        public int add(int a, int b) {
                return a+b;
        }
}
```

- Manages the business part of the code
- The methods of the class are the methods supplied by the service
- The connection between the class and the Service Engine are managed by the *service.xml* file

Service code - 2

- AXIS2 deploys services in AAR (Axis ARchive) files
- AAR files are simple WAR archive that contains service code, service.xml, MANIFEST.MF, and possible configuration files
- AAR are create by the java command

  ```
  jar -cvf <service name>.aar *
  ```
- To deploy a service simply copy AAR in services directory of AXIS2 *webapps* subfolder

Basic Service.xml – no security

```
<service name="SecureService">
   <description>Secure Service</description>
   <parameter name="ServiceClass“
        locked="false">SecureService</parameter>
   <operation name="add">
        <messageReceiver
        class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
   </operation>
</service>
```

- At this level, defines only name of the service, name of the methods, and type of parameters

- `messageReceiver` defines the message exchange methodology

Client Code

```
public class SecureServiceClient {

    public static void main(String[] args) throws Exception {

        ConfigurationContext ctx =
        ConfigurationContextFactory.createConfigurationContextFromFileSystem
        ("axis-repo", "null");

        SecureServiceStub stub = new SecureServiceStub
        (ctx,"http://localhost:8888/axis2/services/SecureService");

        ServiceClient sc = stub._getServiceClient();

        sc.engageModule("rampart");

        int a = 3;

        int b = 4;

        int result = stub.add(a, b);

        System.out.println(a + " + " + b + " = " + result);

    }

}
```

Client Code - 2

- Client code manage the request of the service

- Client configuration are managed by the *ConfigurationContext* object

- Stub classes are generated by the *wsdl2java* AXIS2 command

```
wsdl2java -uri <service address>?wsdl -uw -p <package>
-o <source directory>
```

- Stub classes replicate the signatures of services' methods and manage the connection between client and service

- We use the Geronimo port 8888 to allow TCPMon monitoring

Client Code - 3

- The subfolder *axis-repo* contains
  - **AXIS2.xml** configuration file: manages the local configuration of the client – level of security, actions provided, users, …
  - **modules**:contains additional local modules to be used by the client – rampart, rahas, addressing
  - **keys**: contains the user keyrings that contains public-private key-pairs

1st Step – No Security - Messages

- Messages are exchanged as common SOAP envelopes

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
      <soapenv:Body>
         <ns1:add xmlns:ns1="http://ws.apache.org/axis2">
            <ns1:args0>3</ns1:args0>
            <ns1:args1>4</ns1:args1>
         </ns1:add>
      </soapenv:Body>
   </soapenv:Envelope>
```

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
      <soapenv:Body>
         <ns:addResponse xmlns:ns="http://ws.apache.org/axis2">
            <ns:return>7</ns:return>
         </ns:addResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

2nd Step - Timestamp

- The web services engine requires incoming messages to include a Timestamp, and to include a Timestamp in the outgoing messages returned to the client

- Modifications:
  - Insert PWCallback classes to access keyring
  - Service-side: change action in services.xml
  - Client-side:
    modify axis2.xml
    modify client code to load configurations:

```
ConfigurationContext ctx =
      ConfigurationContextFactory.createConfigurationContextFromFileSystem
      ("axis-repo", "axis-repo\\conf\\axis2.xml");
```

Create the Keyring

- Keyring contains public-private key-pairs
- Accessed by special classes in client and services that contain password to open the keyring and extract the keys
- Created through the java *keytool* command

```
keytool -genkey -keystore mykeys.jks -alias fulvio –keyalg RSA
```

Timestamp – Services.xml

```
<parameter name="InflowSecurity">
   <action>
        <items>Timestamp </items>
   </action>
</parameter>

<parameter name="OutflowSecurity">
   <action>
        <items>Timestamp </items>
   </action>
</parameter>
```

Password Callback class

```
public class PWCallback implements CallbackHandler {
  public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
        if (callbacks[i] instanceof WSPasswordCallback) {
            WSPasswordCallback pc=(WSPasswordCallback)callbacks[i];
            if (pc.getIdentifer().equals("fulvio")) {
                pc.setPassword("password");
            } else {
                throw new UnsupportedCallbackException(callbacks[i],
                        "Unknown user");
            }
        } else {
        throw new UnsupportedCallbackException(callbacks[i],"Unrecognized
    Callback");
```

Timestamp – AXIS2.xml

```
<parameter name="OutflowSecurity">
   <action>
        <items>Timestamp</items>
   </action>
</parameter>
<parameter name="InflowSecurity">
   <action>
        <items>Timestamp</items>
   </action>
</parameter>
```

2nd Step – Timestamp - Messages

- We used rampart to add timestamp to SOAP envelopes. They provide a way to limit the lifespan of messages, Timestamp expires five seconds after the creation of the message, so if the message is older than that, the message must be rejected.

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="true">
  <wsu:Timestamp
         xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
                wss-wssecurity-utility-1.0.xsd" wsu:Id="Timestamp-1">
   <wsu:Created>2010-12-10T22:58:45.656Z</wsu:Created>
   <wsu:Expires>2010-12-10T23:03:45.656Z</wsu:Expires>
  </wsu:Timestamp></wsse:Security>
```

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="true">
   <wsu:Timestamp
         xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
                wss-wssecurity-utility-1.0.xsd" wsu:Id="Timestamp-2">
         <wsu:Created>2010-12-10T22:58:46.296Z</wsu:Created>
         <wsu:Expires>2010-12-10T23:03:46.296Z</wsu:Expires>
   </wsu:Timestamp>
   <wsse11:SignatureConfirmation xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" wsu:Id="SigConf-1" /> </wsse:Security>
```

3rd Step - Signature

- Signing a message involves creating a version of the data that's been encrypted in a known way, so that decrypting it provides a value comparable to the original

- Create the security.properties file to indicate which type of encription to exploit

- Modify services.xml and AXIS2.xml to manage signature

security.properties File

```
org.apache.ws.security.crypto.provider=
   org.apache.ws.security.components.crypto.Merlin

org.apache.ws.security.crypto.merlin.keystore.type=jks

org.apache.ws.security.crypto.merlin.keystore.password=password

org.apache.ws.security.crypto.merlin.file=mykeys.jks
```

Signature - services.xml

```
<parameter name="InflowSecurity"><action>
   <items>Signature</items>
   <user>fulvio</user>
   <passwordCallbackClass>PWCallback</passwordCallbackClass>
   <signaturePropFile>security.properties</signaturePropFile>
</action></parameter>

<parameter name="OutflowSecurity"><action>
   <items>Signature</items>
   <user>fulvio</user>
   <passwordCallbackClass>PWCallback</passwordCallbackClass>
   <signaturePropFile>security.properties</signaturePropFile>
</action></parameter>
```

Signature – AXIS2.xml

```
<parameter name="OutflowSecurity"><action>
    <items>Signature</items>
    <user>fulvio</user>
    <passwordCallbackClass>client.PWCallback</passwordCallbackClass>
    <signaturePropFile>axis-repo\\conf\\security.properties</signaturePropFile>
    <encryptionUser>fulvio</encryptionUser>
    <signatureParts>Body</signatureParts>
</action></parameter>

<parameter name="InflowSecurity"><action>
    <items>Signature</items>
    <user>fulvio</user>
    <passwordCallbackClass>client.PWCallback</passwordCallbackClass>
    <signaturePropFile>axis-repo\\conf\\security.properties</signaturePropFile>
    <signatureParts>Body</signatureParts>
</action></parameter>
```

3rd Step – Signature - Messages

- Axis2 signs, the relevant part of the message and adds the signature. It then sends the message. When the service receives the message, it accesses the keystore to get the public key for that user, and then verifies the signature.


```xml
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="Signature-1">
   <ds:SignedInfo>
       <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
       <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
       <ds:Reference URI="#id-2">
         <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>sDtm6Lc7/amLp576X5cv1NDy2jY=</ds:DigestValue>
       </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>Z6rgmCJVs5SFyOaAstcDIuHovMoEBoQnW7FgoNVwALtNjp56WlDYWw+F9puU4bHV0FF
    TloC+m+YQ9qvnk1IJijUY7BZxQantAhUiQmXB95bn0LnEn1mNeem4TdbZSNMxJlG9JaefHiKZY21FiTUb56vO1
    gttKo3p6aJ6qa63NtA=</ds:SignatureValue>
    <ds:KeyInfo Id="KeyId-B3FDB613E8DD0F597A12920233777652">
       <wsse:SecurityTokenReference
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
                    utility-1.0.xsd" wsu:Id="STRId-B3FDB613E8DD0F597A12920233777653">
          <ds:X509Data><ds:X509IssuerSerial>
          <ds:X509IssuerName>CN=fulvio,OU=dti,O=unimi,L=cre,ST=cr,C=it</ds:X509IssuerName>
          <ds:X509SerialNumber>1291903493</ds:X509SerialNumber>
          </ds:X509IssuerSerial></ds:X509Data>
       </wsse:SecurityTokenReference>
</ds:KeyInfo></ds:Signature>
```

4th Step – Full Security

- The message is signed, encrypted, and enriched with the timestamp

- Encryption obscure information so that competitors and other users can't read it

- Encryption is add to the application modifying services.xml and AXIS2.xml

Full Security – services.xml

```
<parameter name="InflowSecurity"><action>
   <items>Timestamp Signature Encrypt</items>
   <user>fulvio</user>
   <passwordCallbackClass>PWCallback</passwordCallbackClass>
   <signaturePropFile>security.properties</signaturePropFile>
</action></parameter>

<parameter name="OutflowSecurity"><action>
   <items>Timestamp Signature Encrypt</items>
   <user>fulvio</user>
   <passwordCallbackClass>PWCallback</passwordCallbackClass>
   <signaturePropFile>security.properties</signaturePropFile>
</action></parameter>
```

Full Security – AXIS2.xml

```
<parameter name="OutflowSecurity"><action>
   <items>Timestamp Signature Encrypt</items>
   <user>fulvio</user>
   <passwordCallbackClass>client.PWCallback</passwordCallbackClass>
   <signaturePropFile>
        axis-repo\\conf\\security.properties
   </signaturePropFile>
   <signatureKeyIdentifier>SKIKeyIdentifier</signatureKeyIdentifier>
   <encryptionKeyIdentifier>SKIKeyIdentifier</encryptionKeyIdentifier>
   <encryptionUser>fulvio</encryptionUser>
   <signatureParts>Body</signatureParts>
   <optimizeParts>
        //xenc:EncryptedData/xenc:CipherData/xenc:CipherValue
   </optimizeParts>
</action></parameter>
```

4th Step – Full Security

- Axis2 has replaced the actual request with an *EncryptedData* element that includes information on how the data was encrypted, as well as the actual encrypted data (in the *CypherData* and *CypherValue*) elements

- The data was encrypted with a shared key, which means that the message has to include that key so that it can by decrypted. The shared key has been encrypted with the receiver's public key and embedded in the Header, in the *EncryptedKey* element. This key also includes a *ReferenceList*, which includes a *DataReference* that points back to the data this key was used to encrypt

- So to reverse direction, the receiver (the server) receives the message, uses its own private key to decrypt the shared key, and then uses the shared key to decrypt the body of the message.

Summarizing

- Security levels are set up with only little changes in the configuration files

- In order for web services to be truly useful in the enterprise environment, it needs to have the appropriate security capabilities. By combining with technologies such as XML Signature and XML Encryption and providing a standard way of presenting that information, WS-Security makes it possible to protect both incoming and outgoing SOAP messages from several different security threats

- By requiring digital signatures, you can limit access to authorized individuals or organizations, as well as verifying that information has not been altered in transit. By including encryption, it is possible to prevent data from being seen (or at least understood) by unintended recipients. And by adding a Timestamp (and signing it) you can prevent messages from being captured and replayed