WS-Security Examples

Sicurezza delle Architetture orientate ai Servizi

Ernesto Damiani, Fulvio Frati

Università degli Studi di Milano

Outline

- **What is WS-Security**

- **A simple example**
  - 1st step: no security
  - 2nd step: timestamp
  - 3rd step: simple authentication
  - 4th step: signature
  - 5th step: full security (timestamp, signature, encryption)
  - Other Examples

What is WS-Security?

WS-Security:

- Part of **WS-* stack**
- 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?

1 - Integrity - information is not modified in transit

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

How to Secure?

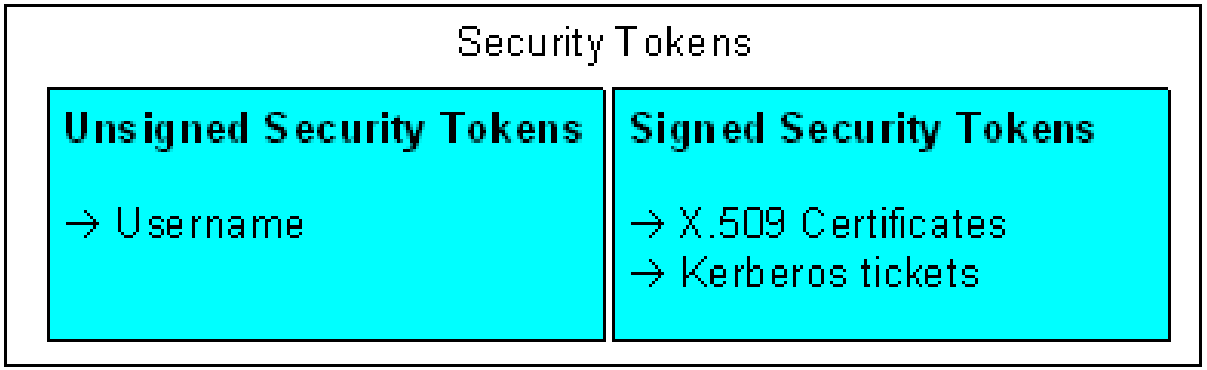
2 - Confidentiality - only authorized actors or security token owners can view the data

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

How to Secure?

3 - Authentication – you are whom you claim to be

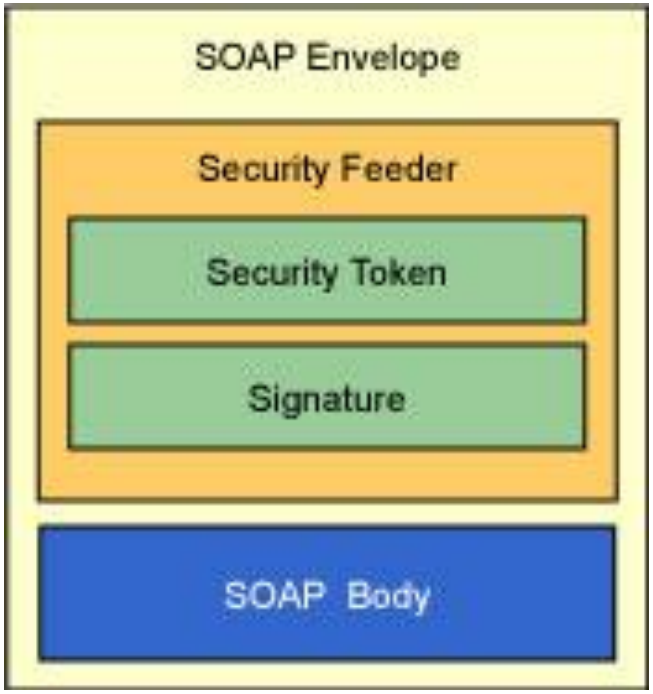
- Security Tokens



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/

Tutorial Architecture

- Simple Service for adding two numbers

- Five incremental security steps:
  1. No Security
  2. Timestamp only
  3. Simple Authentication
  4. Signature only
  5. Full Security: Timestamp + Signature + Encryption

- Modifications 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
  - Path **`/service/SecureService.java`**

- 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
  - Path: **`service/META-INF/service.xml`**

Service code - 2

- AXIS2 deploys services in AAR (Axis ARchive) files
- AAR files are simple WAR archives that contain service code, *service.xml*, *MANIFEST.MF*, and possible configuration files
- AAR are created by the *jar* Java command

```
jar -cvf <service name>.aar *
```

- To deploy a service simply copy AAR in services directory of AXIS2 *webapps* subfolder
  - Path: **`/var/lib/tomcat8/webapps/AXIS2`**

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", "axis-repo/conf/axis2.xml");

     SecureServiceStub stub = new SecureServiceStub
          (ctx,"http://localhost:8080/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 manages the requests of the service
- Client configurations 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 (WSDL) and manage the connection between client and service

Client Code - 3

- The subfolder **`client/src/axis-repo`** contains
  - **conf/AXIS2.xml** configuration file: manages the local configuration of the client – level of security, actions provided, users, keys, …
  - **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
  - command: **`tcpdump -i lo -A  port 8080`**

```
<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

- Web services engine requires that incoming messages includes a *Timestamp*, and includes a Timestamp in outgoing messages returned to the client

- Modifications:
  - **Service-side**: change action in *services.xml*
  - **Client-side**: change action in *axis2.xml*

Create the Keyring

- Keyrings contain 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>
```

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 – Simple Authentication

- Grant the access only to authorized users

- Username and password digests are sent in the header

- Modifications:
  - Insert PWCallback classes to access keyring
  - Service-side: change action in **services.xml**
  - Client-side: change action in **axis2.xml**

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");
```

Simple Authentication – services.xml

```
<!-- SIMPLE AUTHENTICATION -->
<parameter name="InflowSecurity">
   <action>
        <items>UsernameToken</items>
        <passwordCallbackClass>
               PWCallback
        </passwordCallbackClass>
   </action>
</parameter>
```

Simple Authentication – axis2.xml

```
<!-- SIMPLE AUTHENTICATION -->
<parameter name="OutflowSecurity">
   <action>
        <items>UsernameToken</items>
        <user>fulvio</user>
        <passwordCallbackClass>
               client.PWCallback
        </passwordCallbackClass>
   </action>
</parameter>
```

3rd Step – Simple Authentication - Messages

- Username and password are included in the client request
- Server response are sent without security

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
   <soapenv:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
            wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="true">
         <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
           wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-1">
                <wsse:Username>fulvio</wsse:Username>
                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
                     username-token-profile-1.0#PasswordDigest">
                     9Qu+VaRSFI+GriaDvu+A+s+dty4=
                     </wsse:Password>
                <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
                     wss-soap-message-security-1.0#Base64Binary">
                     G6b6oAnX3L0b/tPb5RqymQ==</wsse:Nonce>
                <wsu:Created>2012-01-08T10:27:36.421Z</wsu:Created>
            </wsse:UsernameToken>
         </wsse:Security>
      </soapenv:Header>
      <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>
```

4th 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>
```

4th Step – Signature - Messages

- Rampart signs the selected parts of the message and adds the signature. Then, it 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.

```
<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>
```

5th Step – Full Security

- The message is signed, encrypted, and annotated 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>
```

5th Step – Full Security

- Axis2 has replaced the 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
- 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.
- By adding a Timestamp (and signing it) you can prevent messages from being captured and replayed