Web Services Security

Strategies for Securing Your SOA

Aaron Mulder
CTO – Chariot Solutions

CHARIOT SOLUTIONS

Agenda

- Define Web Services Security
- DIY Security
- HTTPS
- WS-Security
- WS-I Basic Security Profile
- Conclusion
- Q&A

Defining Web Services Security

Security Features

- Covered Here
  - Authentication
  - Integrity
  - Confidentiality
- Not Included
  - Authorization
  - Auditing
  - Identity Management / Federation

Authentication

- Who's calling the service?
- Identify the caller by:
  - Username/password
  - Client certificate
  - SAML
  - Kerberos
  - Rights Expression Language
  - Others...?

Integrity

- No one can change the content of the message en route
  - Doesn't mean no one can read it, just that no one can alter it without the alteration being obvious
- SSL provides point-to-point protection, but an intermediary can alter the data if the message doesn't go directly from the sender to the recipient

Confidentiality

- No one other than the intended recipient can observe the contents of the message
  - Doesn't mean no one can alter it, just that no one can see the contents
- SSL provides point-to-point protection, but if there are intermediaries, each intermediary can read the message and may potentially leave it in plain text when it's passed on

Intermediaries

- Management systems (e.g. direct to appropriate route according to terms of service)
- 3rd party authentication systems (e.g. process authentication and redirect body accordingly)
- Initial destination is in a DMZ
- Message uses a non-HTTP protocol like JMS or SMTP

Case Study

Unified Government Database Queries

- Want 1 query system to access multiple back-end record-keeping systems (3 guesses why)
- Prefer to send "index cards" to central database for queries, rarely hit back-end systems directly
- Existing JMS infrastructure, operating strictly between government agencies
- Must include authentication and integrity (plus extensive auditing requirements)

Implementation: DIY Security

DIY Overview

- Build security information into the message content or message headers

- *Authentication:* Easy
- *Integrity:* Hard
- *Confidentiality:* Hard

DIY Authentication

- Can include a username, password, or ID token (SAML, Kerberos, etc.) in the SOAP request body
  - Could also put it in the header and add a handler to the request processing chain
- Often a staple of public, pre-Web Services XML interfaces
- Can be combined with HTTPS later

DIY Integrity/Confidentiality

- Many languages have Crypto APIs, so you can do this in a theoretically "portable" way
- But it would be quite manual, and you'd have to standardize things like:
  - Algorithms & encoding of binary data
  - Which parts of the request to cover
  - What modifications or formatting other systems are allowed to do
- Probably easier to just use WS-Security

Case Study & DIY

Unified Government Database Queries

- Can use DIY authentication
  - Incorporate identification, agency, etc. into request message or headers
- Can use DIY integrity
  - XML Digital Signature on message body, though everyone involved will need to support it
- Don't need confidentiality
  - Essentially private JMS transport

Implementation: HTTPS

HTTPS Overview

- The default option for for any HTTP Web Service that needs to be secured
- Provides confidentiality and integrity (but usually not authentication) for a point to point connection
- Requires either a certificate signed by a trusted authority or a limited set of clients

HTTPS Advantages

- Performs well (which is to say, server implementations are optimized and hardware acceleration is commonly available)
- Only requires a server certificate to establish a secure channel
- Any web server can provide HTTPS, and pass the content on to subsequent logic

HTTPS Disadvantages

- Client certificates may or may not be easy to support, and may or may not be passed through to authentication logic
  - Often combined with DIY authentication
- Not secure when used with intermediaries
- Does not protect non-HTTP protocols
  - JMS, JMS, JMS

Implementing HTTPS Security

- Web Services code is no different than for HTTP-based Web Services
- Just a matter of configuring the client and the server for HTTPS
  - Correct certificates, trust store, etc.
- Authentication via client certificates may require extra configuration (e.g. to pass certificate information from the web server to the application server)

Case Study & HTTPS

Unified Government Database Queries

- Not an option due to the JMS infrastructure
- Even if it were possible, it's hard to prove that the message wasn't altered by an intermediary since you can't tell if the HTTPS client is the original one or somewhere down the chain

Implementation: WS-Security

WS-Security 1.0 Overview

- Originally "just a couple companies", now more widely accepted and becoming the Web Services security standard
- Built on previous XML (not necessarily Web Services) security standards
  - Includes authentication, integrity, and confidentiality
- Being incorporated into future standards efforts

WS-Security Advantages

- Covers all 3 AIC goals
- Works well with intermediaries and non-HTTP protocols
- Can selectively protect content, e.g.
  - Password but not message
  - Different parts of message for different intermediaries or recipients (password vs. credit card)
- Tools & libraries are becoming available

WS-Security Disadvantages

- Still a lot of loose room in the spec (from prior drafts, etc.)
- Performance is not so good compared to HTTPS
  - Hardware acceleration rare
  - Processed by SOAP stack not web server
- Not well integrated into SOAP platforms yet
- Interoperability can still be a challenge

WS-Security Implementations

- Apache WSS4J (open source)
  - Core logic
  - JAX-RPC integration
  - Axis integration
- Sun JWSDP (free)
  - JAX-RPC integration
- Vendor tools
  - .NET, WebMethods, WebLogic, etc.

WS-Security Interoperability

- Version 1 of Product A to Version 2 of Product B
  - Yes
- Minor changes of version, off by one service pack, or trying to use a new product:
  - Not necessarily
- Best to pick a favorable server and limit the set of acceptable clients

Case Study & WS-Security

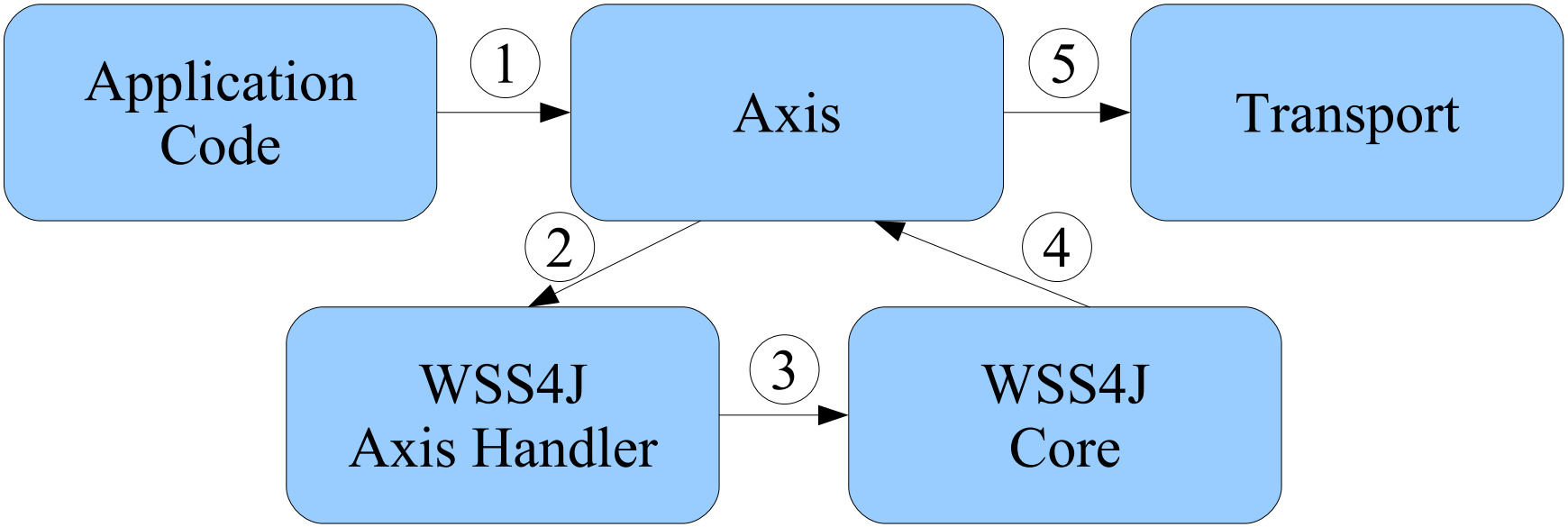
Unified Government Database Queries

- Can be used for authentication and integrity
- An alternative to custom embedding of authentication information
- An alternative to "plain" XML Digital Signature
  - Should come with better tool support
  - Will (eventually) be quite interoperable
  - Fine for now, since the set of clients is fairly limited

WS-Security with WSS4J & Axis

- WSS4J chosen because it is free / open source and has good interoperability and JAX-RPC is a PITA
- Axis chosen because WSS4J has built-in Axis and JAX-RPC support but see above
- Google knows how to configure other tool sets for WS-Security

WSS4J & Axis Architecture



CHARIOT SOLUTIONS

Steps to Run

- Deploy Axis in Tomcat, copy in various JARs
- Write a web service back-end (POJO)
- Write a `.WSDD` file for the service
- Deploy it
- Use WSDL2Java to generate a client
- Make sure the client talks to the server
- Run TCPMon and configure the client for it
  - Observe the insecure call

Steps to Run, cont.

- Create a client-side `.WSDD` with security
- Write Callback Handlers (for authentication); prepare keystores & config files (for integrity and/or confidentiality)
- Run client again
  - Should fail, but see security stuff in TCPMon
- Update and redeploy server `.WSDD`
- Run client again, should work

Getting Axis Running

- Install Tomcat (I used 5.5.12)
- Download Axis (I used 1.3)
- Download WSS4J (I used 1.1.0)
- Copy Axis `webapps/axis` to Tomcat `webapps` directory
- Copy JavaMail, JAF, Bouncycastle, Apache XML Security, WSS4J, and OpenSAML JARs into *`tomcat`*`/webapps/axis/WEB-INF/lib` & *`axis-install`*`/lib`

Confirm Axis Installation

- Visit `http://localhost:8080/axis` and `http://localhost:8080/axis/happyaxis.jsp`
- Check that the Axis happiness page does not complain about any missing libraries

Writing the Web Service

- Can just be a POJO – we'll set any public methods to be exposed as services

```
public class Sample {
    public int[] getValues() {
        return new int[]{12, 14};
    }
    public String getName(int value) {
        return "Name "+value;
    }
}
```

Deploying the Web Service

- Compile and copy the class to the `WEB-INF/classes` of the Axis web app
- Create a `.WSDD` file for Axis:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java=
     "http://xml.apache.org/axis/wsdd/providers/java">
    <service name="Sample" provider="java:RPC">
        <parameter name="className"
              value="org.test.Sample"/>
        <parameter name="allowedMethods" value="*"/>
    </service>
</deployment>
```

Deploying the Web Service, cont.

- Restart Tomcat (to make sure web service class is visible)
- Run Axis deploy tool with `.WSDD` file:

```
java -classpath ... org.apache.axis.client.AdminClient
   sample.wsdd
```

- Need all libs from `axis/lib` on the Class Path for this command to work (and all subsequent commands too)

Confirm Service

- Visit

  `http://localhost:8080/axis/servlet/AxisServlet`

  to confirm that the service is listed
- Click the WSDL link next to the service to make sure it works

Generate Client Code

- Use the Axis WSDL2Java tool to generate client code

```
java -classpath ... org.apache.axis.wsdl.WSDL2Java
   -o outputDirectory
   http://localhost:8080/axis/services/Sample?wsdl
```

- Check for the generated code under `localhost/axis/services/ServiceName/` in the output directory (this package can be configured)

Write Client main()

- Create a class to call the client:

```
public static void main(String args[]) {
    try {
        SampleService service =
                        new SampleServiceLocator();
        Sample port = service.getSample();
        int[] all = port.getValues();
        String name = port.getName(all[0]);
        System.out.println("Output: " + name);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Test the Client

- Compile the class with the main method along with all the generated client code
- Run the main class, with all the `axis/lib` JARs on the Class Path
- Should get the expected output (in this case, "`Output: Name 12`")

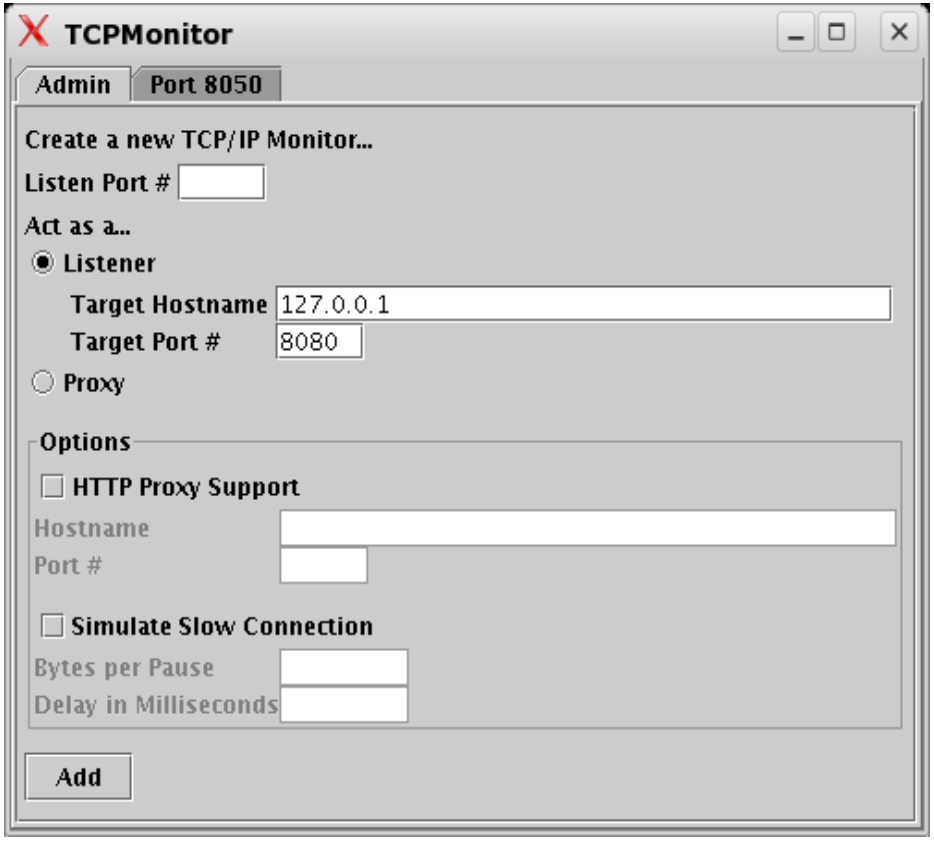
Start the TCPMon

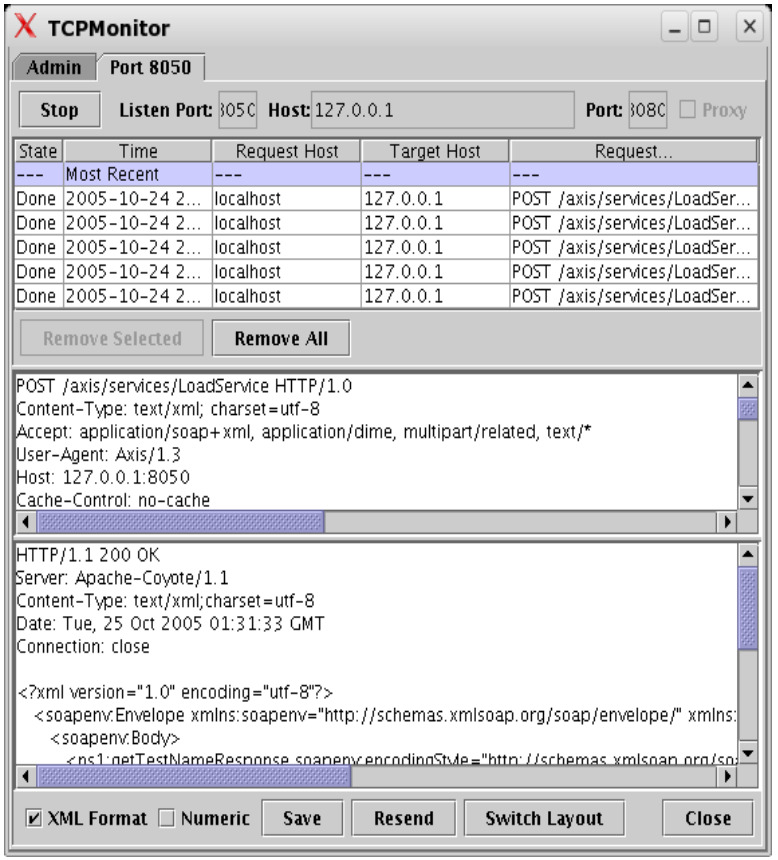
- An Axis tool to let us show the SOAP request and response
- Start it like this:

```
java -classpath ... org.apache.axis.utils.tcpmon
```

- Put in a different port (8050) as the Listen Port # and hit "Add"

TCPMon





CHARIOT SOLUTIONS

Update Client for TCPMon

- Override the URL used to contact the service in the main() method, using the port that TCPMon is running on instead:

```
SampleService service = new SampleServiceLocator();
Sample port = service.getSample(new URL(
        "http://localhost:8050/axis/services/Sample"));
...
```

- Recompile and run the client again

Observed Request

```
<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope xmlns:soapenv=
          "http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!-- note: no SOAP header present -->
  <soapenv:Body>
    <ns1:getValues soapenv:encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1="http://axis.sample.org"/>
  </soapenv:Body>
</soapenv:Envelope>
```

Add Authentication to Client

- Create a client-side `.WSDD` file:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java=
           "http://xml.apache.org/axis/wsdd/providers/java">
  <service name="Sample">
    <requestFlow>
      <handler type=
           "java:org.apache.ws.axis.security.WSDoAllSender">
        <parameter name="action" value="UsernameToken"/>
        <parameter name="user" value="SomeUsername"/>
        <parameter name="passwordType" value="PasswordText"/>
        <parameter name="passwordCallbackClass"
                   value="org.test.TestCallbackHandler"/>
      </handler>
    </requestFlow>
  </service>
</deployment>
```

Create Client Callback Handler

- The authentication logic uses a JAAS callback handler to gather missing information (such as the password)
- The callback type is `WSPasswordCallback`, which is a WSS4J class
- You need to write a callback handler that takes one of these and provides (in this case) the password

Test Callback Handler Code

```
public void handle(Callback[] callbacks) throws IOException,
                                    UnsupportedCallbackException {

    for (int i = 0; i < callbacks.length; i++) {
        Callback callback = callbacks[i];
        if(callback instanceof WSPasswordCallback) {
            WSPasswordCallback pwc =
                                (WSPasswordCallback) callback;
            pwc.getIdentifier(); //username from .WSDD
            pwc.setPassword("mypassword");
        } else {
            throw new UnsupportedCallbackException(callback);
        }
    }
}
```

Server-Side Callback Handler

- The server side will also use a Callback Handler, with the same Callback type, to validate the authentication
  - It should throw an exception if the username and/or password was not valid
- You can use the same handler on both sides for test purposes – so long as it doesn't throw an exception the server will assume the authentication is valid

Deploy Callback Handler

- Put the class in a JAR in:
  - `tomcat/webapps/axis/WEB-INF/lib`
    - The server-side callback handler
  - `axis/lib`
    - The client-side callback handler
- Restart Tomcat to make sure the server-side one is picked up

Update the Client Configuration

- Deploy the client `.WSDD` to create a `client-config.wsdd` file in the current directory

```
java -classpath ... org.apache.axis.utils.Admin client
   sample-client.wsdd
```

- Confirm that `client-config.wsdd` was written and has settings for the Sample service

Test the Client

- Run the client code again, with `client-config.wsdd` in the current directory or on the Class Path (and the client callback handler on the Class Path)
- Observe the request in TCPMon
- Should get a SOAP Fault as a response because the server doesn't understand the authentication request yet (and it's marked for mandatory processing)

Observed Request w/ Auth

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:xsd="..."
xmlns:xsi="...">
  <soapenv:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" soapenv:mustUnderstand="1">
      <wsse:UsernameToken>
        <wsse:Username>Aaron</wsse:Username>
        <wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-
profile-1.0#PasswordText">mypassword</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
  ...
```

Update Server .WSDD

```
<deployment xmlns="..." xmlns:java="...">
  <service name="Sample" provider="java:RPC">
   <parameter name="className" value="org.test.Sample"/>
   <parameter name="allowedMethods" value="*"/>
   <requestFlow>
     <handler type=
  "java:org.apache.ws.axis.security.WSDoAllReceiver">
       <parameter name="passwordCallbackClass"
                  value="org.test.TestCallbackHandler"/>
       <parameter name="action" value="UsernameToken"/>
     </handler>
   </requestFlow>
  </service>
</deployment>
```

Redeploy Server Side

- Run Axis deploy tool with updated server-side `.WSDD` file:

```
java -classpath ... org.apache.axis.client.AdminClient
   sample.wsdd
```

- Visit the Axis service list page and WSDL just to make sure you don't get any weird errors
  - Putting some JARs in `common/lib` instead of `WEB-INF/lib` causes problems

Retry Client

- Should work!
- Server response is unchanged for authentication
- If you put output in the callback hander(s), you should see it on both the client side (gathering the info) and the server side (validating the info)
- Can now try more meaningful service implementation, callback handers, etc.

Encrypt the Password, Client

- Update the client-side `.WSDD` file:

```
<deployment ...>
  ...
  <handler type=
            "java:org.apache.ws.axis.security.WSDoAllSender">
    <!-- include same parameters as before plus: -->
    <parameter name="addUTElement" value="Nonce Created" />
    <parameter name="encryptionPropFile"
                                value="crypto.properties" />
    <parameter name="encryptionKeyIdentifier"
                                value="X509KeyIdentifier" />
    <parameter name="encryptionUser" value="cert-ID" />
    <parameter name="encryptionParts"
      value="{Element}{http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd}UsernameToken" />
   </handler>
```

Encrypt the Password, Server

```
<deployment xmlns="..." xmlns:java="...">
  <service name="Sample" provider="java:RPC">
   <parameter name="className" value="org.test.Sample"/>
   <parameter name="allowedMethods" value="*"/>
   <requestFlow>
     <handler type=
  "java:org.apache.ws.axis.security.WSDoAllReceiver">
       <parameter name="passwordCallbackClass"
                  value="org.test.TestCallbackHandler"/>
       <!-- action is updated, prop file is new -->
       <parameter name="action"
                        value="UsernameToken Encrypt"/>
       <parameter name="decryptionPropFile"
                        value="crypto.properties" />
     </handler>
...
```

Create crypto.properties Files

- Used by both client and server to point to a keystore. The named file (and listed keystore file) should be on the Class Path. It should look like:

```
org.apache.ws.security.crypto.provider=
            org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.file=my-keystore-file
org.apache.ws.security.crypto.merlin.keystore.type=JKS
org.apache.ws.security.crypto.merlin.keystore.password=foo
org.apache.ws.security.crypto.merlin.keystore.alias=my-key
```

Encryption Issues

- If there are many clients and one server, it may be convenient for each client to include its own public key in the request
- Need to make sure Axis doesn't reformat the SOAP message before passing it on
- Can layer handlers to significantly change settings for different parts of the body
- There are configuration options for this stuff and more

More Use Cases

- Also easy to sign all or part of the SOAP body
- Also easy to sign and encrypt all of the SOAP body
- Also possible to separately sign/encrypt multiple chunks within the SOAP body
- Signing and encrypting can use the same or different keystore properties files

WSS4J & Axis Thoughts

- It works
- Documentation is poor and scattered
- Not obvious how to use WSS4J for J2EE web services
  - Perhaps can hook in via JAX-RPC handlers?
  - May not be possible to hook WS authentication to server's authentication mechanism
- If you have a friendlier web services tool set with WS-Security support, use it!

The Future: WS-I Basic Security Profile

About WS-I BSP

- Web Services Interoperability Organization
- WS-I Basic Profile ensures interoperability between J2EE, .Net, and other Web Services implementations
  - Includes SOAP/WSDL but not security
- WS-I Basic Security Profile is currently in draft, including WS-Security and various authentication token types

WS-I BSP Expectations

- WS-Security interoperability is getting better but will remain somewhat hit or miss until WS-I BSP is finalized
- Once WS-I BSP is finalized, J2EE specs and various products will include support and be certified for interoperability
- You may prefer not to deploy WS-Security until this happens
  - Or, find out your vendor's certification plans

Conclusion

Questions for Securing Services

1) Will you use JMS or other intermediaries?
2) What are the performance requirements?
3) What is the set of possible clients?
4) What types of authentication are required?
5) Are integrity and/or confidentiality required? If so, on what parts of the message?
6) What are the platform/vendor's plans for WS-I BSP support?

Conclusion

- Depending on the answers, WS-Security may be a good way to go
  - Main challenges are performance and interoperability with arbitrary SOAP stacks
- HTTPS is relatively high-performance and very interoperable, but:
  - It requires an authentication solution
  - It only works for HTTP-based Web Services

Discussion

Aaron Mulder: ammulder@chariotsolutions.com

Download slides at: http://www.chariotsolutions.com/