RUHR-UNIVERSITÄT BOCHUM

RUB

Security Analysis of the Web Services Frameworks

Stefan Wigchers

Seminar thesis
February 25, 2011
Chair for Network and Data Security
Prof. Dr. Jörg Schwenk

advised through Juraj Somorovsky

Abstract

Web services have become more popular during the last years. Besides the functionality it is very important to secure the web service. The usage of a secure channel is not always possible or useful. The given standards of WS-Security are good to secure the message itself. However, not every framework includes it in the same way.
This paper deals with the security of web services. Two of the most popular web service frameworks will be compared in relation to their security features. It is shown step by step how to configure a web service with the most required security settings, given by WS-Security, for both frameworks.
Furthermore, the problems with these settings are demonstrated through attacks against the web services and propositions for countermeasures are given.

Contents

1. Introduction

1.1. Web Services

The most common interactions are program-to-user communication [GGKS02]. To allow a program-to-program interaction, a special interface is needed. A web service is such a software interface which works with standardized XML[1] messaging [BHM+04]. It provides all operations needed for its task, so that the other program can use them. The web service needs to be implemented by an agent, which handles all the communication. All information needed to use a web service are described in a WSDL[2] file [GGKS02].
Three roles can be defined within an interaction with web services (depicted in Figure 1.1):
The ”service registry” contains all web services created by a ”service provider”. In this register, a ”service requester” can find the web service he wants to use and the related WSDL file.
On the messaging layer, SOAP[3] messages are used for communication between server and client [GGKS02]. However, the standard does not handle any security features. To enable that, an extension is needed.

1.2. WS-Security

If the communication is between two trusted parties, only secure transport is needed. This can be realized with ”HTTPS[4]”. But a problem comes up when a third party is in the middle of the communication. For example, when a user uses an online shop and wants only his bank to be able to read his credit card information. When ”HTTPS” is used to secure the transport, the data are send over a secure connection to the online shop. The shop receives all information in plaintext and has to send them over another secure connection to the bank (Figure 1.2). To ensure, that the shop cannot read the information without using a direct connection to the bank, it is necessary to secure the

[1] Extensible Markup Language
[2] Web Services Description Language
[3] Simple Object Access Protocol
[4] Hypertext Transfer Protocol Secure

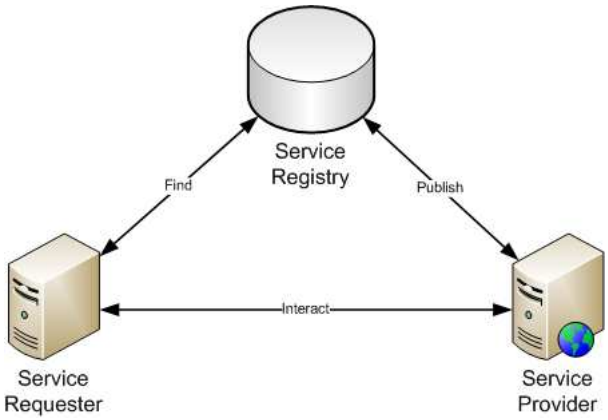

Figure 1.1.: Usage of web services

message itself.
WS-Security is an extension that allows to do that. It is specified by OASIS[5] and de-

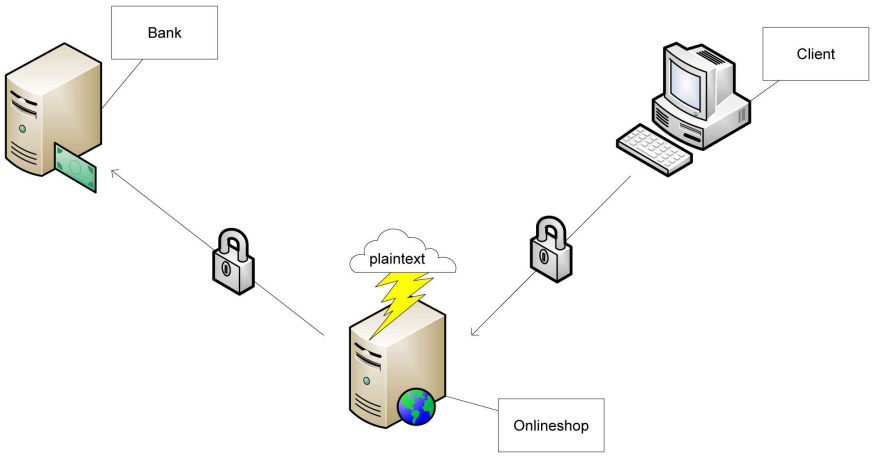

Figure 1.2.: Problem of HTTPs

scribes different mechanisms to ensure message integrity, authenticity and confidentiality [LK06]. The most significant ones are security tokens, signature and encryption. The specification includes different types of security tokens, like X.509, Kerberos and SAML [CKPM05]. It is also possible to define custom tokens [LK06]. To sign a message, XML Signature [BBF+08] is used. There are different signature algorithms which can be used to do so (RSA-SHA1, DAS-SHA1, etc.) [LK06]. The encryption is based on XML Encryption [IDS02] which can be used with Triple-DES and AES [LK06].
The WS-Security specification defines where in the message the respective mechanism has to be included. For example, all elements of the signature are included

[5]Organization for the Advancement of Structured Information Standards

in a `<ds:Signature>` element, which is included in a `<wsse:Security>` element [LK06].

2. Axis2

Axis2 is a web service framework developed by the Apache Software Foundation and has a better performance than its predecessor Axis. The main reason for this is, that Axis2 uses a combination of ”DOM” and ”SAX” [Li09] parser and not the lower performant ”DOM” parser, which is used by Axis. Another reason for the better performance is the use of ”AXIOM”, which allows efficient access to all data and gives all features which has ”DOM” to [FTW10].
All handler used by Axis2 are called one after another, which builds a chain (or flow) of handler. Every handler can modify the message or start an event (e.g. logging) [FTW10]. Axis2 has a separation between incoming and outgoing messages. This allows a fine handling for every web service. Especially for the security options, the separation becomes important.
Axis2 provides different standards like SOAP (1.1 and 1.2), WSDL (1.1 and 2.0), ”WS-Addressing” and ”WS-Policy” [FTW10]. For other standards add-ons are needed, like WS-Security, WS-SecureConversation, WS-Trust and WS-Reliable Massage [FTW10]. In this paper, the add-on Rampart is used to get WS-Security standard.

2.1. Rampart

Rampart is a security module for the Axis2 web service Framework. Without this module there is no WS-Security feature in Axis2. It provides standards like WS-Security (1.0 ans 1.1), WS-SecureConversation, WS-Security Policy and WS-Trust [Fou].
To implement Rampart in Axis2, the matching version of Rampart, has to be copied in the ”modules” directory of Axis2. When that is done, the Rampart module can be activated for all running web services on the administration page of Axis2 [FTW10].
For all examples in this paper, Axis2 (version 1.5.2) with Rampart (version 1.5) runs on an Apache Tomcat Server (version 6.0.29).

2.2. Setting up a web service with Axis2

A web service for Axis2 is build of two important parts. The first one is the Java class with all functionality in it. In this paper, an easy echo function is used to show

Listing 2.1: EchoService.Java

```
1 public class EchoService {
2
3   public String echo(String value) {
4     return value;
5   }
6 }
```

the important security features (see Listing 2.1). The other important part is the *services.xml* file. This file includes all settings for the web service. Listing 2.2 shows the *services.xml* for the above web service in Listing 2.1. All files have to be included

Listing 2.2: services.xml

```
1 <service>
2  <parameter name="ServiceClass" locked="false">EchoService</parameter>
3  <operation name="echo">
4   <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
5  </operation>
6 </service>
```

in a *.aar* file with a defined structure. The *services.xml* has to be in the META-INF folder and the *.class* files outside of this folder. The structure of the .aar file for our web service is shown in Figure 2.1.

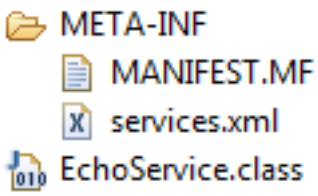

Figure 2.1.: Structure of .aar file

2.3. Security Settings

To secure a web service under Axis2, all settings have to be included in the *services.xml* file. Rampart allows the separation of *InflowSecurity* (for incoming messages) and *OutflowSecurity* (for outgoing messages). To define security options the

`<parameter name=''InflowSecurity''/''OutflowSecurity''>` element is needed. The child of this is the `<action>` element which includes all actions. To define the operations, which are wanted for the service, the `<items>`element (is child of `<action>`) is needed. All options are divided by a space. An example of this is shown in Listing 2.6.

2.3.1. Timestamp

A timestamp ensures message freshness, so that the receiver can drop the message when it is too old. This technique can prevent replay attacks, whereby an old message is send again by the attacker. The settings for creating a timestamp for outgoing messages or requiring a timestamp for every incoming message are easy using rampart. Listing 2.3 shows the server settings, whereby the incoming messages need a

Listing 2.3: services.xml with timestamp settings

```
<service>
  <parameter name="InflowSecurity">
    <action>
      <items>Timestamp</items>
    </action>
  </parameter>
  <parameter name="OutflowSecurity">
    <action>
      <items>Timestamp</items>
      <timeToLive>300</timeToLive>
      <precisionInMilliseconds>true</precisionInMilliseconds>
    </action>
  </parameter>
  <parameter name="ServiceClass" locked="false">
    EchoService
  </parameter>
  <operation name="echo">
    ...
  </operation>
</service>
```

timestamp and the outgoing messages get a timestamp with 300 milliseconds to live. Messages with expired timestamps become dropped. The problem is that there is no possibility to ensure that the timestamp was not manipulated. To do that an additional mechanism is needed, like signature.

2.3.2. Signature

In order to ensure message integrity and that the received message is not changed during the transport, signatures must be applied. The XML Signature standard defines a number of signing algorithms, key identifiers and the structure of the signature, applied on XML data tree. When using rampart for the signature, a keystore with a private and public key for server and client are required. Moreover, two new files are needed in the .aar file. The first one is ”crypro.properties” which includes the location, type and password of the keystore and the crypto implementation which is used (Listing 2.4). The

Listing 2.4: crypto.properties

```
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=server_password
org.apache.ws.security.crypto.merlin.file=server.jks
```

second file is a password callback class (here called ”MyCallback.class”). This class is called by rampart and gives back the password for the private key in the keystore. Listing 2.5 shows an example class. Now the services.xml can be configured. The

Listing 2.5: MyCallback.Java [FTW10]

```
import org.apache.ws.security.WSPasswordCallback;
...
import java.io.IOException;

public class MyCallback 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.getUsage()==WSPasswordCallback.SIGNATURE){
                          if("server".equals(pc.getIdentifier()))
                            pc.setPassword("server_password");
                        }
                }else{
                        throw new UnsupportedCallbackException(callbacks[i],"
                            Unrecognized Callback");
      }
    }
  }
}
```

inflow part is short, because there is not much data needed. Only the location of the

”crypto.properties” file has to be declared. For the outflow more data are important. The user of the private key are important and the location of the password callback class. Optional are the signature algorithm, the digest algorithm and the definition of the signed parts. The server required a message with a timestamp and a signed body

Listing 2.6: services.xml with timestamp and signature settings

```
<service>
  <parameter name="InflowSecurity">
    <action>
      <items>Signature Timestamp</items>
      <signaturePropFile>crypto.proberties</signaturePropFile>
    </action>
  </parameter>
  <parameter name="OutflowSecurity">
    <action>
      <items>Signature Timestamp</items>
      <user>server</user>
      <signaturePropFile>crypto.proberties</signaturePropFile>
      <passwordCallbackClass>MyCallback</passwordCallbackClass>
      <timeToLive>30</timeToLive>
    </action>
  </parameter>
  ...
</service>
```

element (this is the standard setting). The outgoing messages have a timestamp and are signed, too.
The problem of this signature is the vulnerability against *signature wrapping*. This attack is shown in chapter ”Attacks against Axis2”.

2.3.3. Encryption

In order to ensure message confidentiality XML Encryption is applied. The settings are similar to the signature settings and the same files are needed. The keystore of the server includes the public key of the client, which is required for encryption. Public keys are not password protected, so that no changing of the password callback is needed for outgoing encryption. Indeed, when incoming messages are encrypted, changes are necessary. Listing 2.7 shows the new callback class. The crypto.properties file is not changed to the signature settings. To encrypt the message, a symmetric-key algorithm is used and this key is encrypted with a public-key algorithm. Ramparts allow *Triple-DES* or different *AES* versions as symmetric-key algorithms, whereby *AES-128* is default. The standard public key algorithm is RSA. Similar to the signature is the capability to choose the part which shall be encrypted, the algorithms to

Listing 2.7: MyCallback.Java for decryption and signature [FTW10]

```
import org.apache.ws.security.WSPasswordCallback;
...
import java.io.IOException;

public class MyCallback 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.getUsage()==WSPasswordCallback.SIGNATURE){
                          if("server".equals(pc.getIdentifier()))
                            pc.setPassword("server_password");
                        } else if(pc.getUsage()==WSPasswordCallback.DECRYPT){
                          if ("server".equals(pc.getIdentifier()))
                            pc.setPassword("server_password");
                        }
                }else{throw new UnsupportedCallbackException(callbacks[i],"
                    Unrecognized Callback");
}}}}
```

do that and the key identifier. Listing 2.8 shows a sample for decryption of incoming messages and encryption of the reply.

2.3.4. Username Token

Another solution for authentication is an addition of Username Tokens. A mechanism to check username and password is required. For outgoing username token, the password callback has to be changed. In Listing 2.9 the new callback class is shown. This is only a minimal example and in real it is useful to use a database for all password instead of many entries in the callback class. The password can send in plaintext or as a password digest. The problem is, that both options are not secure because the digest method is only a base64 encoded version of the password. It is important to encrypt the message or use a secure transport (like https). The password digest is the default option in rampart [FTW10]. With the element ”passwordType” and the option ”PasswordText”, the password is sent in plaintext. To prevent replay attacks, rampart adds a nonce and a timestamp.
The example in Listing 2.10 shows a setting which requires a username for incoming messages and inserts the username ”server” with the password ”server_pwd” (base64 encoded) in outgoing messages.

Listing 2.8: services.xml with timestamp and signature settings

```
<parameter name="InflowSecurity">
  <action>
    <items>Encrypt</items>
    <decryptionPropFile>crypto.proberties</decryptionPropFile>
    <passwordCallbackClass>MyCallback</passwordCallbackClass>
  </action>
</parameter>
<parameter name="OutflowSecurity">
  <action>
    <items>Encrypt Timestamp</items>
    <encryptionPropFile>crypto.proberties</encryptionPropFile>
    <passwordCallbackClass>MyCallback</passwordCallbackClass>
    <encryptionUser>client</encryptionUser>
    <timeToLive>30</timeToLive>
  </action>
</parameter>
```

2.3.5. WS-SecurityPolicy

An additional possibility to set up the security is to use WS-Policy. The settings for this can be included in the services.xml file and allow to choose if only one of the settings or all of them must be fulfilled. Listing 2.11 shows an example for WS-Policy.

2.4. Problems of Axis2

There are some problems when using the standard settings of Axis2, which are shown below.

2.4.1. Timestamps

Axis2 and Rampart allow only one timestamp per message and defines the location within the security header. The problem is the detection of timestamps inserted deeper in the document. When another timestamp is wrapped in a new element, it is not detected. This is normally no problem, because the wrapped timestamp is not verified for its freshness. When using a signed timestamp, that can be a problem. An attacker can use signature wrapping [GJLS09] to make a replay attack possible.
Figure 2.2 shows a SOAP message with a signed body and timestamp. The attacker uses the wrapping method, because he wants to perform a replay attack. He puts the signed timestamp as a child of the new element `<test>` in a new position (red marked) and adds a new timestamp (which has a new id and is not expired) to the old

Figure 2.2.: Signature wrapping with timestamp

Listing 2.9: Complete MyCallback.Java [FTW10]

```
import org.apache.ws.security.WSPasswordCallback;
...
import java.io.IOException;

public class MyCallback 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.getUsage() == WSPasswordCallback.USERNAME_TOKEN){
                                if("server".equals(pc.getIdentifier()))
                                        pc.setPassword("server_pwd");
                        } else if (pc.getUsage()==WSPasswordCallback.SIGNATURE){
                          if("server".equals(pc.getIdentifier()))
                            pc.setPassword("server_password");
                        } else if(pc.getUsage()==WSPasswordCallback.DECRYPT){
                          if ("server".equals(pc.getIdentifier()))
                            pc.setPassword("server_password");
                        }

                }else{
                        throw new UnsupportedCallbackException(callbacks[i],"
                            Unrecognized Callback");
                }
        }
}}
```

position (green marked). When the receiver checks the signature, it is valid, because the old signature is not changed. To check if the message is expired, the receiver uses the new timestamp, because the old one is not noticed.
Moreover, there could found other security issues. Rampart accept messages with a timestamp created in the future or without information when it is created. A more precise verification of that would be desirable.

2.4.2. Signature

The attack shown above, can also be executed with every other signed element. There is no default verification of the XML schema[1] verification in it. That is the reason why the normal signature setting should never be used in its minimal form. Therefore, it is highly recommend to use XML schema validation and WS-Security Policy validation.

[1] Description of the structure of the document

Listing 2.10: services.xml with username settings for incoming messages

```
<parameter name="InflowSecurity">
  <action>
    <items>UsernameToken</items>
    <passwordCallbackClass>MyCallback</passwordCallbackClass>
  </action>
</parameter>
</parameter>
```

2.4.3. SOAPAction Spoofing

Another problem of Axis2 is the vulnerability against SOAPActoin spoofing attacks. If the message is sent over http, the attacker can change the http-header field ”SOAPAction”. Normally this field includes the same function name as the body. If the

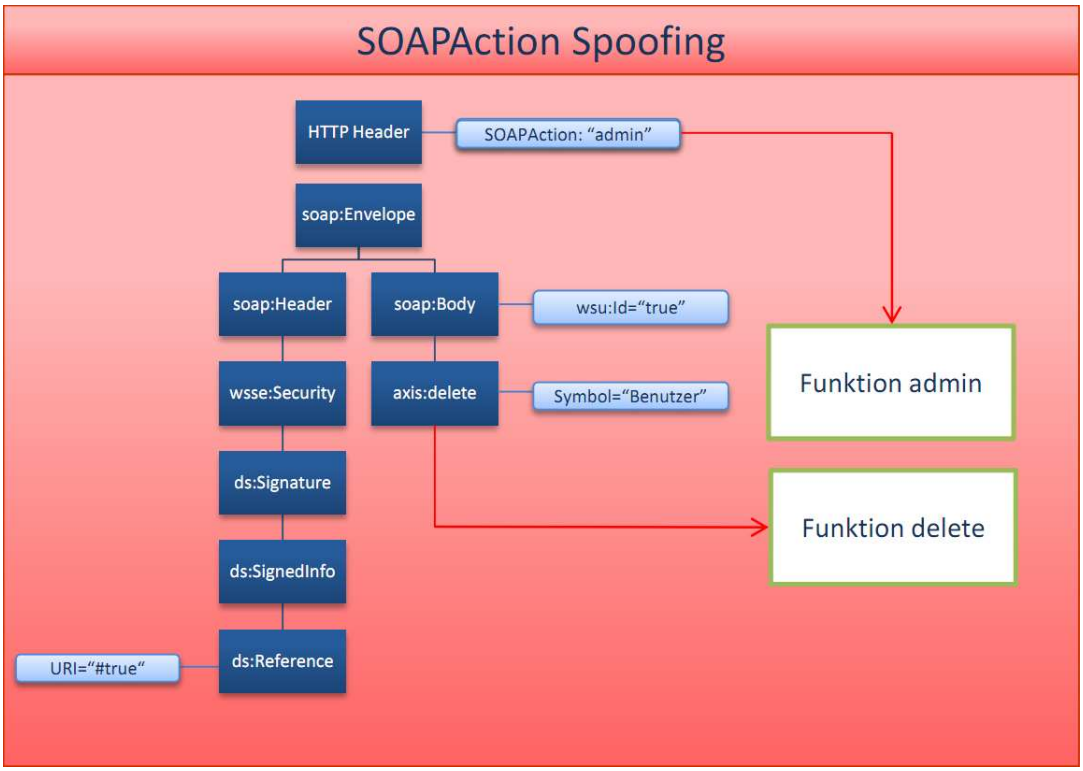

Figure 2.3.: SOAPAction Spoofing with encrypted body

attacker knows the name of other functions in the web service, he can change the SOAPAction field and the server does not run the function which is called in the body, but the function in the http-header (see Figure 2.3).
This attack works also with a signed body.

Listing 2.11: services.xml ws-policy [axp]

```
<wsp:Policy
    wsu:Id="SigEncr"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
        oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
   <wsp:ExactlyOne>
      <wsp:All>
        <sp:AsymmetricBinding xmlns:sp="http://schemas....07/securitypolicy">
          <wsp:Policy>
           ...
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict />
             </wsp:Policy>
            </sp:Layout>
            <sp:IncludeTimestamp />
            <sp:OnlySignEntireHeadersAndBody />
          </wsp:Policy>
        </sp:AsymmetricBinding>
          <sp:Wss10 xmlns:sp="http://schemas....securitypolicy">
          <wsp:Policy>
            <sp:MustSupportRefKeyIdentifier />
            <sp:MustSupportRefIssuerSerial />
          </wsp:Policy>
        </sp:Wss10>

         <sp:SignedParts xmlns:sp="http://schemas....5/07/securitypolicy">
           <sp:Body />
         </sp:SignedParts>

         <sp:EncryptedParts xmlns:sp="http://schemas....5/07/securitypolicy">
          <sp:Body />
        </sp:EncryptedParts>

         <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
           <ramp:user>service</ramp:user>
          <ramp:encryptionUser>client</ramp:encryptionUser>
           <ramp:passwordCallbackClass>MyCallback</ramp:passwordCallbackClass>
           <ramp:signatureCrypto>
             <ramp:crypto provider="org.apache.ws.security.components.crypto.Merlin">
               <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.
                   type">JKS</ramp:property>
               <ramp:property name="org.apache.ws.security.crypto.merlin.file">server
                   .jks</ramp:property>
               <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.
                   password">pwd</ramp:property>
             </ramp:crypto>
           </ramp:signatureCrypto>

           <ramp:encryptionCypto>
            ...
        </ramp:RampartConfig>
      </wsp:All>
   </wsp:ExactlyOne>
 </wsp:Policy>
```

3. JBossWS

The JBossWS web service framework was released in 2008 [Bra]. It is based on JAX-WS[1] which is a Java-API for creating web services in an easier way [Ora].
JBossWS is a part of the JBoss aplication server and offers three different web service stacks: JBoss Native, Apache CXF and Sun Metro [Bra]. This paper shows the security settings of the native stack which comes by default with the application server [Solb]. JBossWS includes beside WS-Addressing and WS-Policy other standards like WS-

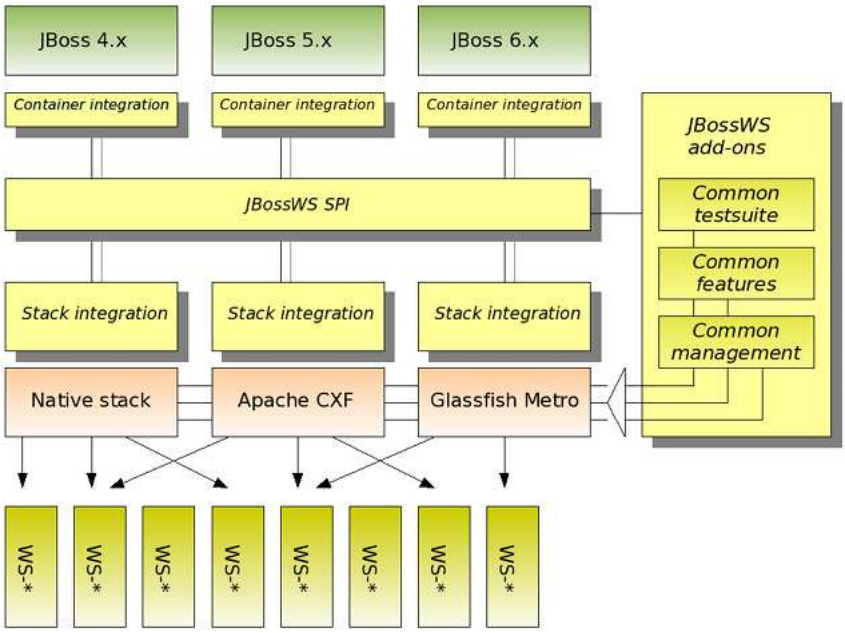

Figure 3.1.: JBoss project architecture [jbo]

ReliableMessaging, WS-Eventing and WS-Security (1.0) [jbo]. This means that no modul is needed to get messages security with JBossWS. In this paper a JBoss Aplication Server (version 5.1.0) with JBossWS-Native (version 3.3.1) under Windows 7 is used.

[1] Java API for XML - Web Services

3.1. Setting up a web service with JBossWS

The web service for JBossWS has the same function as the one for Axis2. However, there are differences in the Java code.
In line 6 is "*@WebService*" important to mark the class EchoService as a web ser-

Listing 3.1: EchoService.Java

```
1  package service;
2
3  import javax.jws.WebService;
4  import javax.jws.WebMethod;
5
6  @WebService
7  @EndpointConfig(configName = "Standard WSSecurity Endpoint")
8  public class EchoService
9  {
10    @WebMethod
11    public String echo(String value)
12    {
13       return value;
14    }
15 }
```

vice and in line 9 stands "*@WebMethod*" to mark that "*echo()*" can be called. Line 10 is only important for WS-Security, because it defines the security handler [Sola]. A web service without WS-Security settings, doesn't need this line.
To deploy the service, a deployment descriptor has to be implemented, too. The "web.xml" file includes all information for it. These files has to be saved in a *.war*

Listing 3.2: web.xml

```
1  <web-app>
2    <servlet>
3      <servlet-name>EchoService</servlet-name>
4      <servlet-class>service.EchoService</servlet-class>
5    </servlet>
6
7    <servlet-mapping>
8      <servlet-name>EchoService</servlet-name>
9      <url-pattern>/*</url-pattern>
10   </servlet-mapping>
11 </web-app>
```

file. Figure 3.2 shows the structure of this file.

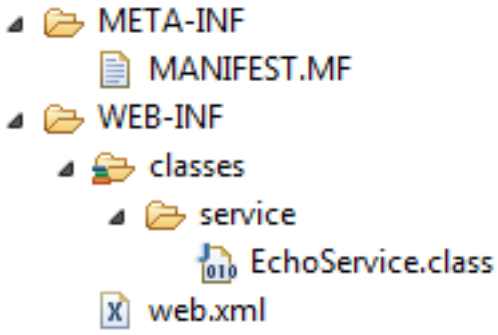

Figure 3.2.: Structure of .war file

3.2. Security Settings

The security settings under JBossWS have to be included in a special XML file. On the server side, this file is "*jboss-wsse-server.xml*" and it is in the same folder as the *web.xml*. The configuration for outgoing messages is child of the element <config>. The options for incoming messages are child of <requires> which is child of <config>. Each option has its own element in this file.

3.2.1. Timestamp

JBossWS allows the same settings for timestamps as Axis2. Listing 3.3 shows the

Listing 3.3: jboss-wsse-server.xml

```
<jboss-ws-security xmlns="http://www.jboss.com/ws-security/config"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://www.jboss.com/ws-security/config
                   http://www.jboss.com/ws-security/schema/jboss-ws-security_1_0.xsd">

  <timestamp-verification createdTolerance="10"
                          warnCreated="false"
                          expiresTolerance="10"
                          warnExpires="false" />

  <config>
    <timestamp ttl="30"/>
    <requires>
      <timestamp/>
    </requires>
  </config>
</jboss-ws-security>
```

configuration for the server. All incoming messages require a timestamp and outgoing

messages include one with 30 seconds to live. JBossWS affords additional options for timestamps [Solc] (line 6 in Listing 3.3). These options may be important when client and server have not exact the same time. *"createdTolerance"* allows messages with a timestamp created 10 seconds in the future and *"expiresTolerance"* allows to accept messages with a 10 second expired timestamp. The two other options enable a log of messages which use the previous options. These settings are optional and line 6 can be dropped when they are not needed.

3.2.2. Signature

To setup the signature options it is not required to build a callback class for passwords. However, for JBossWS it is necessary to create a truststore beside the keystore. The

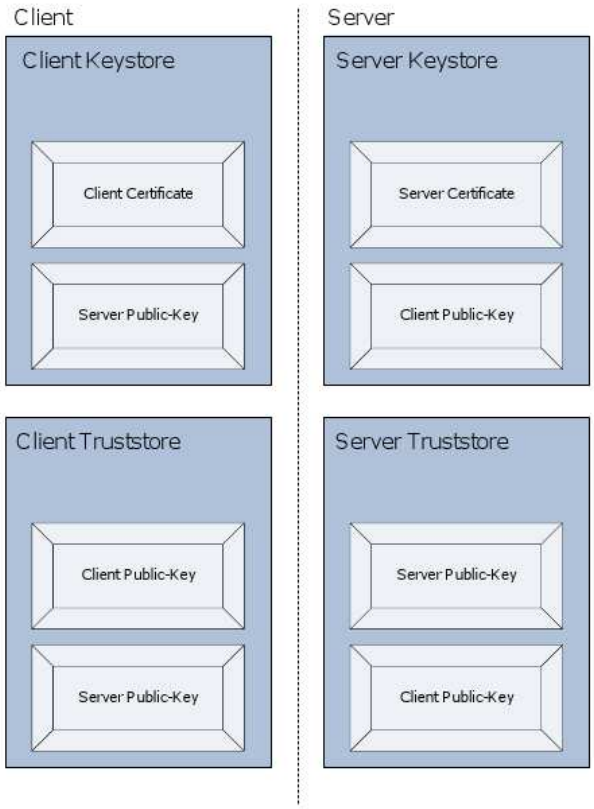

Figure 3.3.: Keystores and Truststores [JJ09]

keystore includes the private key of the server (or client) and all public keys. The truststore includes the public keys of all parties.
The server requires with the settings in Listing 3.4 a messages with signed body and a timestamp. The request includes a signed body and timestamp. To sign body and

Listing 3.4: jboss-wsse-server.xml

```
1  <key-store-file>WEB-INF/server.keystore</key-store-file>
2  <key-store-password>server_pwd</key-store-password>
3  <key-store-type>jks</key-store-type>
4  <trust-store-file>WEB-INF/server.truststore</trust-store-file>
5  <trust-store-password>server_pwd</trust-store-password>
6
7  <config>
8      <sign type="x509v3"
9            alias="server"
10           includeTimestamp="true"/>
11     <requires>
12       <timestamp/>
13       <signature/>
14     </requires>
15 </config>
```

timestamp, the key with alias ”server” and the ”x509v3” is used. All important information about the key- and the truststore are in lines 1 - 5. These settings are much easier than that ones in Axis2.

3.2.3. Encryption

The same applies to encryption with JBossWS. It is no password callback required for the encryption. The settings are very similar to the settings of signature. Listing 3.5 shows an example for a setting with encryption. The settings for ”type” and ”alias” are

Listing 3.5: jboss-wsse-server.xml

```
1  <key-store-file>WEB-INF/server.keystore</key-store-file>
2  ...
3  <trust-store-password>server_pwd</trust-store-password>
4
5  <config>
6      <encrypt type="x509v3"
7               alias="client"
8               algorithm="aes-128"
9               keyWrapAlgorithm="rsa_15"
10              tokenReference="directReference" />
11     <requires>
12       <timestamp/>
13       <encryption/>
14     </requires>
15 </config>
```

the same as for signature. ”algorithm” defines the symmetric algorithm and allows the following options: AES 128(default), AES 192,256 and Triple DES. The asymmetric algorithm is defined under ”keyWrapAlgorithm” and allows RSA v.1.5 (default)and RSA OAEP. For the ”tokenReference” are these values allowed: directReference (default), keyIdentifier and x509IssuerSerial.

3.2.4. Username Token

The config of username and password is as easy as for Axis2. The main element is `<username>` which has optional settings for *password digest*, *nonce* and the time when it is *created*.
It is important to include the line *”@SecurityDomain(''JBossWS'')”* in the

Listing 3.6: jboss-wsse-server.xml

```
<key-store-file>WEB-INF/server.keystore</key-store-file>
...
<trust-store-password>server_pwd</trust-store-password>

<config>
    <username digestPassword="true"
              useNonce="true"
              useCreated="true"/>
    <requires>
      <timestamp/>
      <username/>
    </requires>
</config>
```

web service code (under *”@WebService”*). This security domain includes the settings for the users, passwords and roles. These information are to be found under JBOSS_Folder\server\default\conf\props in the files ”jbossws-users.properties” and ”jbossws-rols.properties”.

3.2.5. WS-SecurityPolicy

To use WS-Policy with JBossWS, the user has to create his own wsdl file and mark that in the java code with *”@WebService(wsdlLocation="WEB-INF/wsdl/own.wsdl")”*. Listing 3.7 shows an example of a wsdl file with WS-Policy.

3.3. Problems of JBossWS

The problems, found for JBossWS are shown below.

Listing 3.7: wsdl wiht ws-policy [Sola]

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name='HelloService' targetNamespace='...'...>
 <types>
  ...
 </types>
 <wsp:Policy wsu:Id='X509EndpointPolicy' xmlns:wsu='http://...1
     -wss-wssecurity-utility-1.0.xsd'>
  <wsp:All>
   <sp:jboss-ws-security xmlns:sp='http://www.jboss.com/ws-security/schema/
       jboss-ws-security_1_0.xsd'>
    <sp:key-store-file>WEB-INF/server.keystore</sp:key-store-file>
    <sp:key-store-password>server_pwd</sp:key-store-password>
    <sp:trust-store-file>WEB-INF/server.truststore</sp:trust-store-file>
    <sp:trust-store-password>server_pwd</sp:trust-store-password>
    <sp:config>
     <sp:encrypt alias='client' type='x509v3'/>
     <sp:requires>
      <sp:encryption/>
     </sp:requires>
    </sp:config>
   </sp:jboss-ws-security>
  </wsp:All>
 </wsp:Policy>
 ...
</definitions>
```

3.3.1. Timestamps

The processing of timestamps is not perfect. The number and location of timestamps is not defined. This means that there is no detection if more than one timestamp are in the message. And additionally only the last timestamp is verified. This allows an attacker to ignore a signed timestamp when he performs a replay attack.
Figure 3.4 shows the message which is sent by the attacker. He has added a new timestamp with a different expire time. JBossWS verifies the signature with the first timestamp (red marked) and checks if the message is expired with the second timestamp (green marked). The message will be accepted!
JBoss allows no timestamps in the future, when it is not explicitly allowed within the settings.

3.3.2. Signature

Signature wrapping here is the same problem as by Axis2 (see Figure 2.2). This is a general problem of WS-Security and has the be prevented from the frameworks by

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:test="http://test/">
   <soapenv:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <ds:Signature Id="Signature-52" 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#rsa-sha1"/>
               <ds:Reference URI="#id-53">
                  <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>uI0KQOBmlSprwV7stw5cRRIHH/I=</ds:DigestValue>
               </ds:Reference>
               <ds:Reference URI="#Timestamp-51">
                  <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>Nr07olWSe6OTLTge2DT75zoj4jg=</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>...</ds:SignatureValue>
            <ds:KeyInfo Id="KeyId-53377FEC76175556F1129440379102856">
               ...
            </ds:KeyInfo>
         </ds:Signature>
         <wsu:Timestamp wsu:Id="Timestamp-51" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <wsu:Created>2011-01-07T12:36:31Z</wsu:Created>
            <wsu:Expires>2011-01-07T12:36:32Z</wsu:Expires>
         </wsu:Timestamp>
         <wsu:Timestamp wsu:Id="faked-imestamp" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <wsu:Created>2011-01-07T12:36:31Z</wsu:Created>
            <wsu:Expires>2011-01-07T12:38:32Z</wsu:Expires>
         </wsu:Timestamp>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body wsu:Id="id-53" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <test:greetClient>
         <!--Optional:-->
         <arg0>?</arg0>
      </test:greetClient>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 3.4.: Two timestamps

using additional techniques like XML schema validation and WS-Security Policy verification.

3.3.3. SoapUI

Testing the web service deployed on JBossWS with SoapUI[2], brings problems with signatures and encryption. The first problem is, that JBoss requires spaces in <ds: X509IssuerName> after every comma and SoapUI does not set them (Figure 3.5).

```
<ds:KeyInfo Id="KeyId-DD2A3FBBB488FCE61D12944912232325">
<wsse:SecurityTokenReference wsu:Id="STRId-DD2A3FBBB488FCE61D12944912232326" xmlns:wsu="http://docs.oa
<ds:X509IssuerSerial>
<ds:X509IssuerName>CN=Unknown,OU=Unknown,O=Unknown,L=Unknown,ST=Unknown,C=Unknown</ds:X509IssuerName>
<ds:X509SerialNumber>1292085103</ds:X509SerialNumber>
</ds:X509IssuerSerial>
</ds:X509Data></wsse:SecurityTokenReference>
```

Figure 3.5.: Key identifier generated from SoapUI

The second problem hits the encryption. JBoss requires a ”wsu:Id” attribute in the body element. SoapUI does not add such an attribute so that a dummy has to add manually. Figure 3.6 shows a corrected message under SoapUI with spaces (red marked) and a dummy ”wsu:Id” attribute (green marked).
These problems are not problems of JBoss itself, they appear only in combination with SoapUI.

[2] http://www.soapui.org/

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:test="http://test/" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns
   <soapenv:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <xenc:EncryptedKey Id="EncKeyId-DD2A3FBBB488FCE61D12944924271108">
            <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
               <wsse:SecurityTokenReference>
                  <ds:X509Data>
                     <ds:X509IssuerSerial>
                        <ds:X509IssuerName>CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown</ds:X509IssuerName>
                        <ds:X509SerialNumber>1292085075</ds:X509SerialNumber>
                     </ds:X509IssuerSerial>
                  </ds:X509Data>
               </wsse:SecurityTokenReference>
            </ds:KeyInfo>
            <xenc:CipherData>
               <xenc:CipherValue>...</xenc:CipherValue>
            </xenc:CipherData>
            <xenc:ReferenceList>
               <xenc:DataReference URI="#EncDataId-7"/>
            </xenc:ReferenceList>
         </xenc:EncryptedKey>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body wsu:Id="element-1-1272320911598-1522000">
      <xenc:EncryptedData Id="EncDataId-7" Type="http://www.w3.org/2001/04/xmlenc#Content">
         <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
         <ds:KeyInfo xmlns:ds="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="#EncKeyId-DD2A3FBBB488FCE61D12944924271108"/>
            </wsse:SecurityTokenReference>
         </ds:KeyInfo>
         <xenc:CipherData>
            <xenc:CipherValue>...</xenc:CipherValue>
         </xenc:CipherData>
      </xenc:EncryptedData>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 3.6.: A corrected message under SoapUI

4. Comparison

As can be seen, both frameworks offer almost the same range of security settings. Axis2, compared to JBossWS, needs a module to get WS-Security features. With Rampart, Axis2 has a module which is easy to include and gives all important features.
The handling of the security features is not perfect for both of the frameworks but it is a good basis which may can strengthened with additional techniques, like XML schema validation. It was shown that creating a signed timestamp is a bit easier with JBossWS. However, JBossWS does not exactly check the number of timestamps and allows replay attacks on an easier way. The timestamp settings are more extensive using JBossWS.
Both frameworks are vulnerable to signature wrapping. There is no validation of the XML schema or a WS-Security Policy verification by default in both frameworks.
Finally the settings for both frameworks is shown (Axis2: Listing 4.1, JBossWS: Listing 4.2), which included the three important WS-Security settings. Both settings create a request with an encrypted signature which signs the body and the timestamp. And both require the same for incoming messages.

Listing 4.1: Settings for a secure web service with Axis2

```
<parameter name="InflowSecurity">
    <action>
      <items>Timestamp Signature Encrypt</items>
      <signaturePropFile>crypto.properties</signaturePropFile>
      <decryptionPropFile>crypto.properties</decryptionPropFile>
      <passwordCallbackClass>MyCallback</passwordCallbackClass>
    </action>
</parameter>
<parameter name="OutflowSecurity">
    <action>
      <items>Timestamp Signature Encrypt</items>
      <timeToLive>30</timeToLive>
      <user>server</user>
      <signaturePropFile>crypto.properties</signaturePropFile>
      <passwordCallbackClass>MyCallback</passwordCallbackClass>
      <signatureParts>
      {Element}{http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-utility-1.0.xsd}Timestamp,
      {}{}Body
      </signatureParts>
      <encryptionPropFile>crypto.properties</encryptionPropFile>
      <encryptionUser>client</encryptionUser>
    </action>
</parameter>
```

Listing 4.2: Settings for a secure web service with JBossWS

```
<key-store-file>WEB-INF/server.keystore</key-store-file>
<key-store-password>server_pwd</key-store-password>
<key-store-type>jks</key-store-type>
<trust-store-file>WEB-INF/server.truststore</trust-store-file>
<trust-store-password>server_pwd</trust-store-password>
<config>
    <sign type="x509v3"
          alias="server"
          includeTimestamp="true"/>
    <encrypt type="x509v3"
             alias="client"
             algorithm="aes-128"
             keyWrapAlgorithm="rsa_15"
             tokenReference="directReference" />
    <requires>
      <timestamp/><signature/><encryption/>
    </requires>
</config>
```

5. Conclusion

In this paper we analyzed the security features of two most used web service frameworks, Axis2 and JBossWS. WS-Security is available for both frameworks with almost all features it provides.
The implementation of XML Signature and XML Encryption is complete, only username token are not entirely implemented (e.g. SAML token which are not completely implemented in Axis2 and missing in JBossWS).
However, the standard settings for WS-Security cause problems, which pose a high risk for a web service. JBossWS does not implement the timestamps in a proper way which causes that messages are vulnerable to replay attacks. Axis2 has a vulnerability to SOAPAction spoofing even if the body of the message is signed. Both frameworks are vulnerable to signature wrapping attacks. This is indeed a problem of the XML Signature standard. However, there exist countermeasures against these attacks as XML schema validation and WS-Security Policy verification.
It is desirable that these features are activated by default, because not every user of the frameworks knows about the danger or the possibilities to prevent such attacks.

A. Bibliography

[axp] Sweet xml. Website. Available online at http://blog.sweetxml.org/2008/01/rampart-policy-samples-for-runing-axis2.html; visited on January 30 2011.

[BBF+08] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. Xml signature syntax and processing (second edition). Website, 2008. Available online at http://www.w3.org/TR/xmldsig-core/; visited on January 2nd 2011.

[BHM+04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. Website, 2004. Available online at http://www.w3.org/TR/2004/NOTE-ws-arch-20040211; visited on November 2nd 2010.

[Bra] Heiko Braun. Jbossws 3.0 - web service framework. Website. Available online at http://www.jbossworld.com/2008/downloads/pdf/thursday/Track_Services_and_Integration_JBOSS_10-1050am_Introduction_to_Web_Services_Heiko_Braun.pdf; visited on January 4nd 2011.

[CKPM05] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Security assertion markup language (saml) v2.0. Website, 2005. Available online at http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf; visited on November 2nd 2010.

[Fou] Apache Software Foundation. Website. Available online at http://axis.apache.org/axis2/java/rampart/; visited on December 27nd 2010.

[FTW10] Thilo Frotscher, Marc Teufel, and Dapeng Wang. *Java Web Services mit Apache Axis2*. entwickler.press, 2010.

[GGKS02] Karl D. Gottschalk, Stephen Graham, Heather Kreger, and James Snell. Introduction to web services architecture. *IBM Systems Journal*, 41(2):170–177, 2002.

[GJLS09] Sebastian Gajek, Meiko Jensen, Lijun Liao, and Joerg Schwenk. Analysis of signature wrapping attacks and countermeasures. Technical report, 2009.

[IDS02] Takeshi Imamura, Blair Dillaway, and Ed Simon. Xml encryption syntax and processing. Website, 2002. Available online at `http://www.w3.org/TR/xmlenc-core/`; visited on January 2nd 2011.

[jbo] Jbossws homepage. Website. Available online at `http://www.jboss.org/jbossws`; visited on January 4nd 2011.

[JJ09] Javid Jamae and Peter Johnson. *JBoss in Action: Configuring the JBoss Application Server*. Manning, 2009.

[Li09] Chengkai Li. Xml parsing, sax/dom. Technical report, 2009.

[LK06] Kelvin Lawrence and Chris Kaler. Web services security: Soap message security 1.1. Website, 2006. Available online at `http:\docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`; visited on November 2nd 2010.

[Ora] Oracle. Oracle fusion middleware: Getting started with jax-ws web services for oracle weblogic server. Website. Available online at `http://download.oracle.com/docs/cd/E12839_01/web.1111/e13758.pdf`; visited on January 30 2011.

[Sola] Alessio Soldano. Jbossws native user guide. Website. Available online at `http://community.jboss.org/wiki/JBossWS-NativeUserGuide`; visited on January 4nd 2011.

[Solb] Alessio Soldano. Jbossws user guide. Website. Available online at `http://community.jboss.org/wiki/JBossWS-UserGuide`; visited on January 4nd 2011.

[Solc] Alessio Soldano. Jbossws ws-security options. Website. Available online at `http://community.jboss.org/wiki/JBossWS-WS-Securityoptions`; visited on January 4nd 2011.