Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht

Thies Rubarth

WS-Security mit Axis

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des Projektes „Ferienklub“ im WS 2005/2006 im Masterstudiengang Informatik sollte eine IT-Landschaft erstellt werden, wie sie in einem fiktiven Ferienklub, in dem sämtliche Aktivitäten der Gäste und der Mitarbeiter durch IT unterstützt werden, denkbar wäre. Es wurde beschlossen eine SOA[1] auf Basis von Webservices aufzubauen. Desweiteren sollte ein ESB[2] eingesetzt werden.

Um in diese SOA externe Anbieter, die über das Internet mit dem Ferienklub verbunden sind (wie z.B. Autovermietungen), integrieren zu können, muss die Kommunikation zwischen den Webservices sicher sein. D.h. Authentifikation, Vertraulichkeit und Integrität müssen sichergestellt werden. Dies kann mittels Verschlüsselung und Signierung der zwischen den Webservice ausgetauschten SOAP[3]-Nachrichten erreicht werden.

Für die Verschlüsselung und Signierung von SOAP-Nachrichten gibt es die Spezifikation WS-Security (siehe. [Antony u. a. (2005)]). In diesem Projektbericht, wird beschrieben, wie diese Spezifikation umgesetzt werden kann.

1.1 Aufbau

Das Projekt wurde in drei Meilensteine unterteilt. Für das Thema WS-Security wurden die Meilensteine wie folgt festgelegt:

1. Auswahl der Laufzeitumgebung
2. Erstellen eines technischen Prototyps
3. Integration von WS-Security in den Servicebus

Der Aufbau dieses Berichts richtet sich nach den Meilensteinen. Für jeden der drei Meilensteine gibt es ein Kapitel, das die Zielsetzung des Meilensteins und dessen Ergebnis beschreibt. Am Ende dieses Berichts werden in einem weiteren Kapitel die Ergebnisse und die Erfahrungen aus dem Projekt bezüglich WS-Security zusammengefasst.

2 Erster Meilenstein: Die Umgebung

Für den ersten Meilenstein sollten geeignete Werkzeuge und Laufzeitumgebungen (Anwendungsserver und Webservice-Engine) betrachtet und ausgewählt werden. Als Programmiersprache sollte aufgrund der bereits vorhandenen Erfahrungen Java verwendet werden. Des-

[1] Service Oriented Architecture
[2] Enterprise Service Bus
[3] Simple Object Access Protocol

weiteren sollte im Rahmen des ersten Meilensteins auch ein Ansatz gewählt werden, wie WS-Security implementiert werden sollte.

Um Webservices betreiben zu können werden ein Webserver mit Servlet-Container und eine Webservice-Engine benötigt. Viele Anwendungsserver bieten selbst schon eine Webservice-Engine an. Um aber vom verwendeten Server unabhängig zu bleiben[4] wurde beschlossen eine separate Webservice-Engine zu verwenden.

2.1 Anwendungsserver

Als Web- bzw. Anwendungsserver standen folgende Produkte zur Auswahl:

- Tomcat
- JBoss
- IBM WebSphere

Die Auswahl fiel auf JBoss (siehe [JBoss]). Im Gegensatz zu Tomcat ist JBoss ein vollständer J2EE-Anwendungsserver und bietet somit zusätzliche Möglichkeiten Anwendungen und Webservices zu nutzen. Zudem handelt es sich im Gegensatz zu IBM WebSphere bei JBoss um ein OpenSource-Produkt, was zum einen die Beschaffung vereinfachte, zum anderen aber auch bessere Möglichkeiten der Fehlerklärung und -behebung verspricht, da der komplette Quellcode der Anwendung eingesehen werden kann.

2.2 Webservice-Engine

Da aus den oben genannten gründen eine separate Web-Service-Engine verwendet werden sollte, wurden die Webservice-Engines Axis 1 und Axis 2 untersucht. Axis 2 ist nicht die Nachfolge-Version von Axis 1, sondern eine komplette Neuentwicklung, die sich als Konkurenz zu Axis 1 sieht. Bei der näheren Betrachtung stellte sich heraus, dass Axis 2 nicht vollständig implementiert ist, so dass die Wahl auf Axis 1 (siehe [Axis]) fiel.

2.3 Entwicklungsumgebung

Um Zeit bei der Entwicklung zu sparen wurde nach Werkzeugen gesucht, die eine gute Unterstützung bei der Erstellung von Anwendungen für den JBoss-Anwendungsserver bieten. JBoss bietet ein Plugin für Eclipse (siehe [Eclipse]) an, dass es ermöglicht J2EE-Anwendungen direkt aus der IDE[5] zu starten.

[4] Einige Teilprojekte hatten sich schon im vornherein für den Webserver Tomcat entschieden, der keine eigene Webservice-Engine anbietet

[5] Integrated Developement Environment

Um eine J2EE-Anwendung zu starten muss diese zunächst in ein EAR[6] gepackt werden. Dieser Packvorgang wird zwar von dem JBoss-Plugin unterstüzt, erfordert jedoch einigen Konfigurationsaufwand, so dass zum Aufbauen und effizienten Nutzen der Entwicklungsumgebung einiger Lern- und damit Zeitaufwand notwendig ist.

2.4 Implementierungsansatz

Um WS-Security zu implementieren, muss die Verschlüsselung und Signierung von Daten möglich sein. Für die Webservices selbst müssen diese transparent geschehen, so dass die Webservices sich nicht darum kümmern müssen, ob und welche Daten ihrer Nachrichten verschlüsselt werden. Die Komponenten, die sich um die Verschlüsselung und Signierung der Daten kümmern, müssen daher in die Axis-Engine eingebunden werden.

2.4.1 XML-Verschlüsselung

Um die SOAP-Nachrichten, die von den Webservices ausgetauscht werden, zu verschlüsseln, müssen die Daten der Nachricht verschlüsselt werden und wieder in XML kodiert werden. Hierzu gibt es die Spezifikationen XML-Encryption (siehe [XML Encryption]) und XML-Signature (siehe [XML Signature]). Apache bietet eine Java-API (siehe [XML Security]) an, die XML-Encryption und XML-Signature unterstützt. Um diese API nutzen zu können, wird eine Bibliothek benötigt, die Implementierungen für die Verschlüsselungs- und Signierungsalgorithmen anbietet. Dafür wird die Bibliothek von Bouncy Castle verwendet (siehe [Bouncy Castle]).

2.4.2 Einbinden von WS-Security in Axis

Um die WS-Security-Implementierung in Axis einbinden zu können, muss die Nachrichtenverarbeitung von Axis betrachtet werden (siehe Abbildung 1). Die Nachrichten durchlaufen in Axis sowohl auf der Client-, als auch auf der Server-Seite eine Folge von Handler-Ketten, wobei die Ketten wiederum aus einer Folge von Handlern bestehen.

Axis definiert drei Handler-Ketten: die Service-, die Global- und die Transport-Kette. In diesen drei Ketten können eigene Handler registriert werden. Die Handler haben vollständigen Zugriff auf die SOAP-Nachrichten und können diese auswerten und manipulieren. Um WS-Security in Axis zu integrieren, werden Handler benötigt, die die SOAP-Nachrichten ver- und entschlüsseln.

[6] Enterprise Application Archive

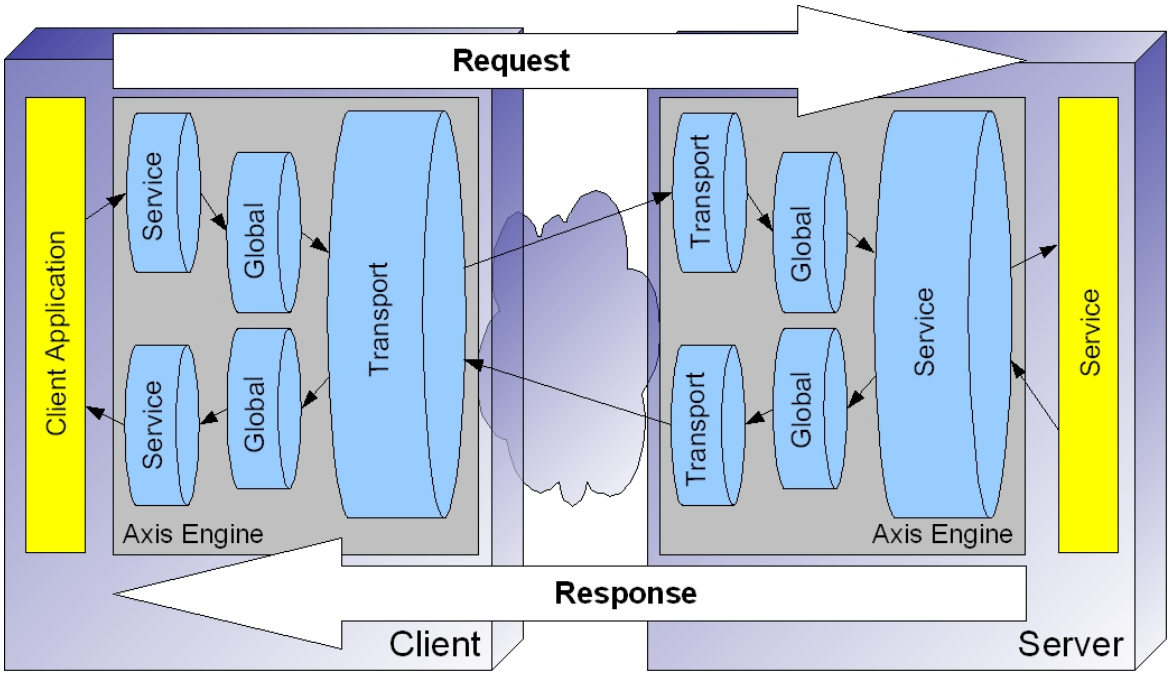

Abbildung 1: Die Nachrichtenverarbeitung von Axis

3 Zweiter Meilenstein: Der technische Prototyp

Für den zweiten Meilenstein sollte ein technischer Prototyp erstellt werden, der zeigt, dass der im ersten Meilenstein gefundene Ansatz für die Implementierung von WS-Security realisierbar ist. Dazu sollten folgende Punkte realisiert werden:

- Zugreifen mit Axis-Handlern auf die SOAP-Nachrichten
- Ver- und entschlüsseln von XML-Daten mit XML-Security
- Ver- und entschlüsseln der Daten aus dem SOAP-Envelope

3.1 Funktionalität des Prototyps

Der Prototyp besteht aus vier Axis-Handlern, jeweils zwei für die Client- und für die Server-Seite, die den Inhalt des SOAP-Envelopes ver- bzw. entschlüsseln (siehe Abbildung 2). Zum Verschlüsseln wird ein zufälliger Schlüssel erzeugt. Mit diesem Schlüssel werden die Daten verschlüsselt. Damit die Daten auf der anderen Seite wieder entschlüsselt werden können, muss der zufällig erzeugte Schlüssel übertragen werden. Dazu wird er mit einem festen Schüssel verschlüsselt. Dieser feste Schlüssel muss in der Praxis durch den öffentlichen

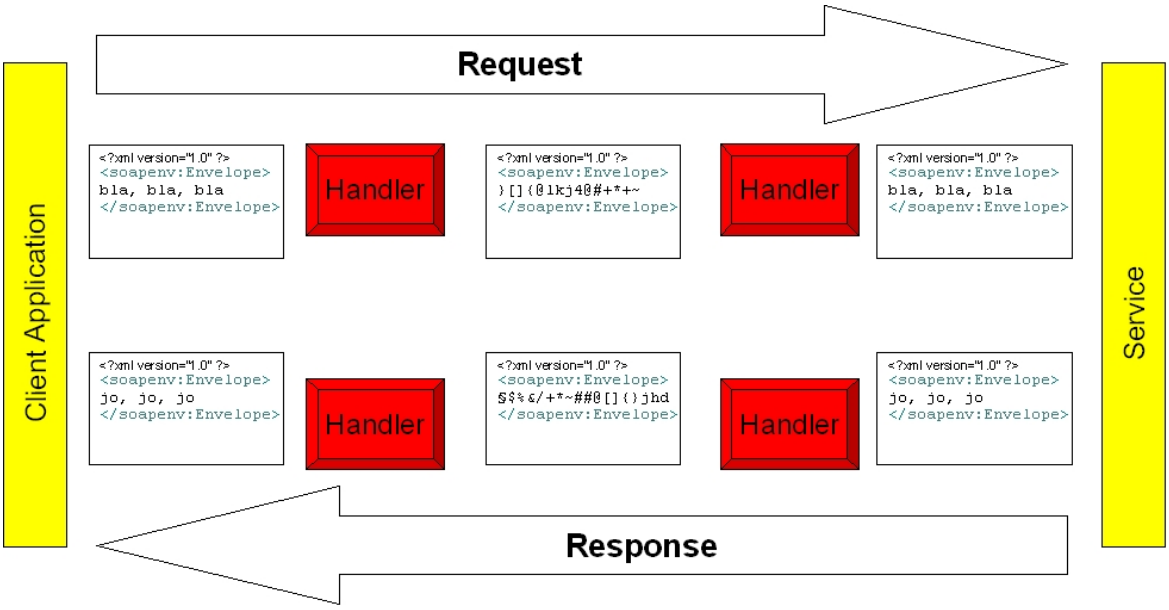

Abbildung 2: Die vier Axis-Handler des Prototyps

Schlüssel des Kommunikationspartners oder einen anderen geheimen Schlüssel ersetzt werden.

Der Prototyp ist nicht WS-Security-konform, da die Daten des SOAP-Envelops zwar gemäß XML-Encryption verschlüsselt werden, nicht aber in einen durch WS-Security spezifizierten Security-Token eingebettet werden.

3.2 Beschreibung des Prototyps

Um in einem Handler auf die SOAP-Nachrichten zugreifen zu können, bietet Axis ein Message-Objekt an, dass den SOAP-Envelope für den Request und die Response beinhaltet. Während das auslesen der SOAP-Envelopes wie erwartet funktioniert, ist das verändern der SOAP-Envelopes über die DOM-API nicht möglich. Um also die SOAP-Nachrichten verschlüsseln zu können muss der Inhalt des SOAP-Envelopes in ein DOM-Dokument umgewandelt werden, welches dann manipuliert werden kann. Es muss dann ein neuer SOAP-Envelope erzeugt werden, der dann zurück in das Message-Objekt geschrieben werden kann. Das erzeugen eines neuen SOAP-Envelopes wird von Axis ebenfalls nicht unterstützt, weshalb hierfür eine eigene SOAP-Envelope-Klasse benötigt wird, die aus einem DOM-Dokument ein SOAP-Envelope erstellt (*org.hawhh.trub.encryption.soap.SecuritySoapEnvelope*).

Das Entpacken des SOAP-Envelopes, das Ver- bzw. Entschlüsseln der XML-Daten und das Erstellen des neuen SOAP-Envelopes wird von einer Helfer-Klasse

(*org.hawhh.trub.encryption.soap.EncryptSOAPEnvelope*) übernommen. Diese Klasse stellt statische Methoden zu Verfügung, mit denen die SOAP-Envelopes aus dem Request oder der Response ver- oder entschlüsselt werden können.

3.3 Registrieren der Axis-Handler

Auf der Server-Seite können die Axis-Handler registriert werden, in dem sie in der Datei deploy.wsdd, mit der der Webservice auf dem Server gestartet wird, eingetragen werden. Die Handler sind dann in der Service-Kette (siehe Abbildung 1) der Axis-Engine aktiv.

Auf der Client-Seite ist das registrieren der Handler nur möglich, wenn der Webservice manuell aufgerufen wird, d.h. die von Axis generierten Proxyklassen nicht verwendet werden. Dort müssen im Java-Code die Handler hart gesetzt werden (siehe Listing 1, Zeile 20).

```
1  Service service = new Service();
2  Call call = (Call)service.createCall();
3  call.setTargetEndpointAddress("http://.../TestService");
4
5  OperationDesc oper = new OperationDesc();
6  oper.setName("getBean");
7  oper.setReturnType(new javax.xml.namespace.QName(
8     "http://.../test", "Bean"));
9  oper.setReturnClass(de.hamburg.uas.cs.ws.Bean.class);
10 oper.setReturnQName(new javax.xml.namespace.QName("", "getBeanReturn"));
11 oper.setStyle(org.apache.axis.constants.Style.RPC);
12 oper.setUse(org.apache.axis.constants.Use.ENCODED);
13
14 call.setOperation(oper);
15 call.setUseSOAPAction(true);
16 call.setSOAPActionURI("");
17 call.setSOAPVersion(org.apache.axis.soap.SOAPConstants.SOAP11_CONSTANTS);
18 call.setOperationName(new javax.xml.namespace.QName(
19    "http://.../test", "getBean"));
20 call.setClientHandlers(new ClientEncryptionHandler(),
21    new ClientDecryptionHandler());
22
23 Bean resp = (Bean)call.invoke(new java.lang.Object[] {});
```

Listing 1: Manueller Webservice-Aufruf mit Axis

4 Dritter Meilenstein: Integration in den Servicebus

Für den dritten Meilenstein sollten die Axis-Handler des Prototypen in den Servicebus integriert werden. Als Servicebus wird Mule (siehe [Mule]) verwendet. Um den Protoyp in den Servicebus zu integrieren, muss es möglich sein die Handler in Axis global zu registrieren, da die Aufrufe der Webservices nicht mehr manuell ausgeführt werden können.

Das globale registrieren der Handler scheint in Axis über Konfigurationen zu funktionieren. Dabei sind die Konfigurationen nicht als Konfigurationsdateien, sondern als Java-Objekte zu verstehen (die gegebenenfalls aus Konfigurationdateien erstellt werden können). Wie genau die Konfigurationen erstellt werden müssen, konnte im Rahmen des Projektes nicht ermittelt werden.

Der zweite Grund, der die Integration in den Servicebus verhinderte ist die eingeschränkte Axis-Engine von Mule, die keine Handler unterstützt.

5 Fazit

Der technische Prototyp hat gezeigt, dass die Implementierung von WS-Security mit Axis prinzipiell möglich ist. Vorrausgesetzt, dass das globale Registrieren der Handler wie versprochen funktioniert, ist auch die Integration in einen Servicebus oder ein anderes Framework, das mit einer Axis-Engine arbeitet möglich, so dass die Verschlüsselung und die Signierung für die Webservices transparent sind.

5.1 Probleme

Die Verfehlung des dritten Meilensteines ist im wesentlichen darauf zurückzuführen, das am Ende des Semesters nicht mehr ausreichend Zeit für das Projekt zur Verfügung stand[7]. Ein weiteres großes Problem ist die sehr dünne Dokumentation von Axis, die sich sehr auf das einrichten von Webservices konzentriert und für weiterführende Entwicklungen nur überblickartige Informationen zur Verfügung stellt. Darüber hinaus ist die Axis-API in weiten Teilen sehr umständlich und unverständlich, so dass auch der Try-and-Error-Ansatz nicht gewünscht schnell zum Erfolg führt.

Den Prototypen mit Hilfe von Work-Arounds in den Servicebus zu integrieren, war nicht möglich, da der Servicebus nur über eine beschränkte Axis-Engine verfügt, die keine Handler unterstützt. Axis selbst in den Servicebus einzubinden war im Rahmen des Projektes nicht mehr möglich.

[7] Der Zeitmangel ist in diesem Fall auch (aber nicht nur) auf die persönlichen Umstände des Autors zurückzuführen

5.2 Offene Punkte

Um WS-Security vollständig zu implementieren sind folgende Schritte notwendig:

- Implementieren von Signaturen
- Einbetten von verschlüsselten Daten in WS-Security konforme Tokens
- Verwenden von öffentlichen Schlüsseln zum Übertragen der symmetrischen Schlüssel
- Globales registrieren der Handler

Optional ist es denkbar, dass WS-Policy-Information ausgewertet werden, um gegebenenfalls nur Teile einer SOAP-Nachricht zu verschlüsseln oder Teile einer SOAP-Nachricht unterschiedlich zu verschlüsseln (z.B. für einen anderen Empfänger, bei weiterleitung von Daten an einen anderen Service).

Literatur

[Antony u. a. 2005] ANTONY, Nadalin ; KALER, Chris ; HALLAM-BAKER, Phillip ; MONZILLO, Ronald: Web Services Security: SOAP Message Security 1.1. (2005)

[Axis] : *Axis Homepage*. – URL `ws.apache.org/axis`

[Bouncy Castle] : *Bouncy Castle Homepage*. – URL `www.bouncycastle.org`

[Eclipse] : *Eclipse Homepage*. – URL `www.eclipse.org`

[JBoss] : *JBoss Homepage*. – URL `http://www.jboss.org`

[Mule] : *Mule Homepage*. – URL `http://mule.codehaus.org`

[XML Encryption] : *XML Encryption*. – URL `http://www.w3.org/Encryption/2001`

[XML Security] : *XML Security Hompage*. – URL `http://xml.apache.org/security`

[XML Signature] : *XML Signature*. – URL `http://www.w3.org/Signature`