schönberg-solutions

Sicherheit für SOAP mit Tomcat und AXIS

schönberg-solutions
Dr. Arndt Schönberg
Eschener Allee 11
26603 Aurich

www.schoenberg-solutions.de

Tel.: +49 4941 69 97 269
Fax: +49 4941 69 97 549
Mobil: 0176 / 205 167 01
eMail: schoenberg@schoenberg-solutions.de

Kurzinformation über den Autor:

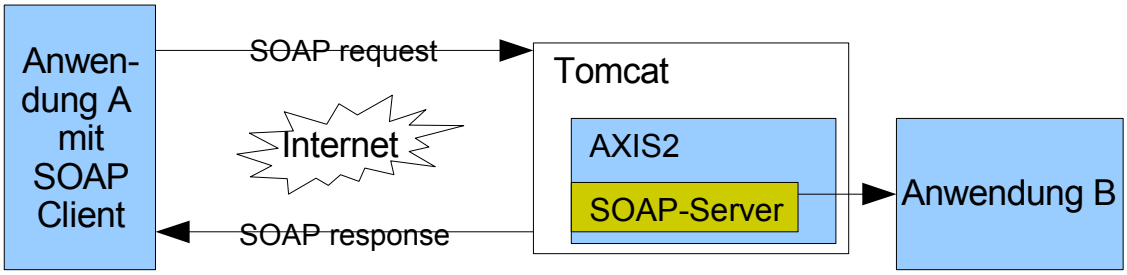
Dr. Arndt Schönberg studierte Informatik und promovierte zum Ingenieur. Er ist als IT-Berater mit den Schwerpunkten neue Technologien und OO/Enterprise-JAVA für seine Firma schönberg-solutions tätig.

Abstrakt

Zur Kommunikation zwischen Anwendungen (z.B. einem Point of Sale und einem Back Office System) oder bei der Erstellung von SOA sind klare und sichere Schnittstellen zwischen den unterschiedlichen Anwendungen notwendig. Eine flexible Art solche Schnittstellen zu realisieren ist die Verwendung des Protokolls SOAP. Eine der verbreitetsten SOAP Implementierungen ist AXIS von der Apache Gruppe. AXIS wird häufig in dem ebenfalls von der Apache Gruppe im Rahmen des Jakarta-Projekts erstellten Servlet-Container Tomcat betrieben. Wird eine solche Architektur angestrebt, gibt es verschiedene Ansatzpunkte zur Realisierung einer abgesicherten Kommunikation.

Einleitung

Mit den folgenden Ausführungen wird ein Überblick über die Absicherungsmöglichkeiten eines häufig vorkommenden Szenarios bei der Anbindung von Diensten in eine Gesamtarchitektur vorgestellt. Eine Anwendung A möchte über ein (unsicheres) Netzwerk Daten mit einer Anwendung B austauschen. Zur Realisierung einer Schnittstelle, die von unterschiedlichsten Plattformen bedient werden kann, wurde SOAP als Technologie gewählt. Es ergibt sich der Aufbau in Abbildung 1.



Die eigentliche SOAP-Schnittstelle gliedert sich in den SOAP-Client, der mit der Anwendung A verbunden ist und die zu versendenden Daten in eine SOAP-Nachricht verpackt. Auf der anderen Seite befindet sich der SOAP-Server, der die SOAP Nachricht entgegen nimmt und den Nachrichtenanteil an Anwendung B weitergibt. Bei einer synchronen Kommunikation erstellt Anwendung B die Antwortdaten und gibt diese über den SOAP-Server in einer SOAP-Antwort an den SOAP Client zurück.

In dem im Weiteren betrachteten Szenario wird der SOAP-Server als Dienst in AXIS 2 [1], der SOAP Implementierung der Apache Gruppe, betrieben. AXIS 2 wiederum ist als Servlet in Servlet-Container Tomcat [2] (Version 5.5.x) installiert. Wie sich im Folgenden zeigen wird sind einige Funktionen von AXIS 2 in der Version 1 noch nicht vollständig vorhanden. AXIS 2 wurde dennoch gewählt, da u.a. wegen des sehr performanten XML-Objektmodells AXIOM und den integrierten WS-Security Funktionen AXIS 2 gute Aussichten hat, an den Erfolg von AXIS 1.x (vergl. [3]) anzuschließen und diese Implementierung auf Dauer abzulösen.

Gefahren auf der Reise

Um eine sichere Kommunikation zu ermöglichen, muss sichergestellt werden, dass beide Kommunikationspartner wissen mit wem sie reden und dass die übermittelten Anfragen und Antworten nicht von einer dritten Partei abgehört werden können (Wer möchte schon, dass die letzten Verkaufsdaten der Niederlassung in China von jedem eingesehen werden

können). Es ist also zu klären, wie sich die Gesprächspartner authentifizieren und wie die Daten sicher übertragen werden können. Nicht eingegangen wird auf die Absicherung gegen Hacker – also auf die Absicherung der Installation von Tomcat und AXIS durch beispielsweise das Einspielen von Patches, Aktualisierung des Betriebssystems und Zugangskontrollen. Für den Betrieb von Tomcat bietet z.B. [4] einen guten Überblick.

Transportschutz

Um die übertragenen Daten zu schützen, hat sich bei vielen Anwendungen (z.B. Online-Shops, Banking) der Einsatz von TLS (Transport Layer Security) bzw. SSL (Secure Sockets Layer) durchgesetzt. Bei TLS handelt es sich um eine Weiterentwicklung des Verschlüsselungsprotokolls SSL. Im allgemeinen Sprachgebrauch und in diesem Artikel wird für beide Varianten SSL als Bezeichnung verwendet.

SSL arbeitet oberhalb der Transportschicht (nach OSI-Modell) und unterhalb der Anwendungsschicht für die Anwendungen transparent. Durch die Verwendung von SSL tauschen die Gesprächspartner Schlüssel aus, mit deren Hilfe die verschickten Nachrichten verschlüsselt werden. Damit eine verschlüsselte Verbindung zu einem Server aufgebaut werden kann muss, dieser ein Zertifikat bereitstellen, dass vom Client akzeptiert werden muss. Überprüft der Client die Echtheit (z.B. durch Vergleich des Fingerprints), kann er nicht nur davon ausgehen, dass die Kommunikation nicht abgehört werden kann, sondern auch, dass der Gesprächspartner (Server) der ist, für den er sich ausgibt.

Eine einfache Möglichkeit die Clients einzuschränken, die mit dem Server kommunizieren dürfen, besteht darin, die IP-Adressen einzuschränken, von denen Tomcat eine Verbindung annimmt. Hierfür kann die Konfigurationsdatei `server.xml` um das Attribut `RemoteAddrValve` erweitert werden. Beispielsweise wird durch die Zeile

```
<Valve className="org.apache.catalina.valves.RemoteAdrValve" allow="127.0.0.1,10.1.*.*"/>
```

localhost und der Adressbereich 1.10.*.* für eine Kommunikation zugelassen. Alle Anfragen von anderen Adressen werden abgelehnt. Da dieses Verfahren keine Authentifizierung im eigentlichen Sinne enthält und nur funktioniert, wenn alle Clients feste IP-Adressen besitzen, wird im Folgenden auf andere Möglichkeiten der Authentifizierung eingegangen.

Wer bist du?

Die möglichen Authentifizierungsmethoden für eine SOAP-Schnittstelle können danach unterschieden werden, an welchem Punkt der Verbindung die Auswertung der Anmeldeinformationen stattfindet. Ohne Anspruch auf Vollständigkeit kann z.B. bei dem Aufbau der Verbindung mit Hilfe von Zertifikaten nicht nur die Identität des Servers, sondern auch die der Clients geprüft werden. Auf ebene der Servlets – AXIS 2 läuft als Servlet in Tomcat – kann Tomcat Berechtigungen prüfen. Innerhalb von AXIS 2 kann das Modul RAMPart eingesetzt werden, dass eine Implementierung der OASIS Web Services Security (siehe [5] und [6]) ist. Weiterhin kann die Authentifizierung auch in die nachgelagerten Anwendungen verschoben werden, indem die Anmeldinformationen in diesen Anwendungen ausgelesen (z.B. Übertragung in den Nutzdaten) und ausgewertet werden.

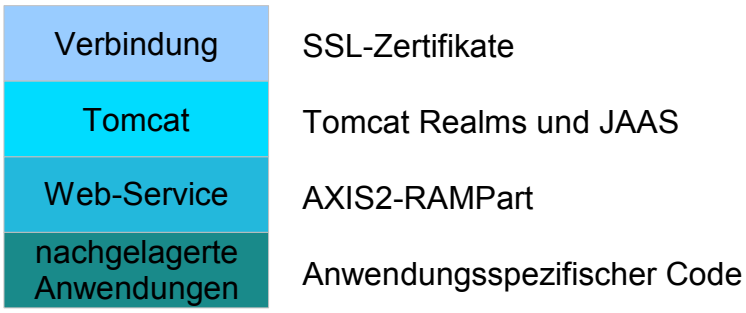


Abbildung: Alternativen der Nutzer-Authentifizierung

Die Verwendung von SSL-Client-Zertifikaten für eine beidseitige Authentifizierung wird für die Absicherung des betrachteten Szenarios selten verwendet. Dies mag daran liegen, dass die in den Schlüsselspeichern der Anwendungen verwarten Zertifikate schwieriger zu Verwalten sind und sich für die Authentifizierung von Nutzern klassische Login / Passwort Kombinationen eher eignen. Aus diesem Grund werden Client-Zertifikate hier nicht weiter behandelt. Eine der wenigen (älteren) Dokumentationen zu dem notwendigen Vorgehen findet sich in [7].

Container-Based / HTTP Basic-Authentifizierung

Tomcat stellt verschiedene Authentifizierungsmechanismen, so genannte Realms, zur Verfügung, die eine Authentifizierung realisieren. Bei diesen Mechanismen können verschiedene Datenspeicher (Datei, Datenbank, LDAP Server...) zur Verwaltung der Anmeldeinformationen verwendet werden. Zu einem Nutzer werden in diesen Datenspeichern Login, Passwort und Gruppenzugehörigkeiten verwaltet. Auf Basis dieser Informationen wird in den Konfigurationsdateien von Tomcat festgelegt welche Nutzer auf die einzelnen Servlets Zugriff haben. Da AXIS 2 als Router-Servlet in Tomcat betrieben wird, können über diesen Mechanismus die Zugriffe auf AXIS 2 beschränkt werden. Diese Beschränkung gilt dann für alle SOAP-Dienste, die in dem Router-Servlet laufen. Sollen SOAP-Dienste mit unterschiedlichen Rechten definiert werden, muss das Router-Servlet mehrmals installiert werden. Wenn die Authentifizierung von Tomcat durchgeführt wurde, kann das Servlet ebenfalls auf die Anmeldeinformationen zugreifen und so eine weitere Prüfung durch die Anwendung durchführen.

Die Konfiguration von Tomcat in der server.xml und von AXIS in der Datei web.xml ist u.a. in [2] und [7] beschrieben.

Um diese Authentifizierung zu verwenden muss in dem SOAP-Client Basic-Authentifizierung realisiert werden. Dies geschieht im Allgemeinen durch erfassen der notwendigen Informationen im HTTP Header. Hier zeigt die Version 1.0 Version von AXIS 2 eine gravierende Schwäche. In der derzeit zum Download stehenden Version ist das setzen der Informationen für die Basic-Authentifizierung nicht (mehr) möglich, da die entsprechende Klasse fehlt. In dem aktuellen Snapshot ist diese Funktionalität aber wieder zu finden, so dass in dem kommenden Release Basic-Authentifizierung wieder verwendet werden kann.

Wenn die entsprechende Klasse

```
org.apache.axis2.transport.http.HttpTransportProperties.BasicAuthentication
```

zur Verfügung steht, können die Authentifizierungsinformationen an Instanzen der Klassen `ServiceClient` oder `OperationClient` gebunden werden. Dies geschieht in den jeweiligen Optionen.

```
HttpTransportProperties.BasicAuthentication basicAuthentication =
            new HttpTransportProperties().new BasicAuthentication();

basicAuthentication.setUsername("Username");
basicAuthentication.setPassword("myPass");

OperationClient operationClient = new OperationClient();
operationClient.getOptions().setProperty(
org.apache.axis2.transport.http.HTTPConstants.BASIC_AUTHENTICATION,
                    basicAuthentication);
```

Zu Beachten ist bei dieser Art der Authentifizierung, dass sie auf HTTP basiert. Da SOAP auch mit anderen Transportprotokollen funktioniert, ist dies eine Einschränkung.

WS-Security

Speziell zur Absicherung von Web-Services (WS) ist die WS-Security entwickelt worden (vergl. [9]). Für JAVA ist die WS-Security in dem Apache Projekt WSS4J realisiert (vergl. [10]). Die wichtigsten Funktionen sind Authentifizierung, Signierung und Verschlüsselung. In Bezug auf die bisherigen Ausführungen ist bei diesem integrierten Ansatz der unterschiedlichen Sicherheitsmechanismen insbesondere der letzte Punkt interessant. Durch die Verschlüsselung der Nachrichten ist die Verschlüsselung des Transportwegs mit SSL nicht mehr zwingend erforderlich – wenn auch nicht schädlich.

Im Allgemeinen werden die Funktionen von WSS4J nicht direkt in den Sourcecode eingebunden, sondern die verwenden Sicherheitsfunktionen in Konfigurationsdateien beschrieben. Für AXIS 2 gibt es zur Realisierung der WS-Security das Modul RAMPart. Dieses wird in AXIS 2 eingebunden und für Server und Client in `service.xml` und `axis2.xml` entsprechende Parameter gesetzt. Eine Beschreibung über den Einsatz findet sich unter [11]. Weitere Informationen zur Anwendung von RAM Part finden sich unter [12] und [13].

Anwendungsspezifischer Code (Custom Security)

Bei Verfahren, die „Custom Security“ realisieren, wird die Authentifizierung durch die Anwendungen hinter der SOAP-Schnittstelle – und nicht durch vorgelagerte Software wie Tomcat oder das AXIS-Servlet – durchgeführt. Hierfür werden die Anmeldeinformationen in den Nutzdaten der SOAP-Nachricht verpackt und durch den Server an die Anwendung weitergereicht (auch die Auswertung von HTTP-Header-Informationen ist in Anwendungen möglich). Der große Nachteil der Custom Security liegt darin, dass „das Rad neu erfunden wird“ und somit ein hoher Aufwand und eine höhere Fehlerwahrscheinlichkeit, als bei den etablierten und vielfach getesteten Verfahren besteht. Als zusätzlicher, verfeinerter Authorisierungsschritt, der auf vorgelagerte Informationen zugreift kann Custom Security jedoch sinnvoll sein.

Zusammenfassung

Es gibt verschiedene Möglichkeiten SOAP Dienste abzusichern. Die Verwendung von SSL als Sicherung der Verbindung und der Echtheit des Servers ist, wenn keine WS-Security eingesetzt wird, unverzichtbar. Für die Verwendung von HTTP-Authentifizierung spricht, dass es eine bekannte Technologie ist, die insbesondere die Einstiegshürden reduziert. Leider ist die derzeitige Implementierung von AXIS 2 an dieser Stelle noch nicht vollständig. Wenn HTTP-Authentifizierung kurzfristig verwendet werden soll, muss also auf AXIS 1.x zurückgegriffen werden. In naher Zukunft wird diese Manko in AXIS 2 jedoch behoben sein. Die Verwendung von SSL-Client-Zertifikaten wird sicherlich nur in Sonderfällen verwendet werden, da der Einsatz komplizierter und die Sicherheit nicht höher ist als bei HTTP-Authentifizierung über SSL.

Zukünftig wird die Absicherung von SOAP-Diensten mit Hilfe der WS-Security deutlich an Bedeutung gewinnen. Insbesondere dadurch, dass die Sicherheitsfunktionen Verschlüsselung und Authentifizierung integriert zur Verfügung stehen, ist eine einfachere Konfiguration und Anwendungsentwicklung möglich, als bei der SSL / HTTP-Authentifizierung. Zusätzliche Funktionen wie die Signierung werden an Bedeutung gewinnen und stehen in diesem Ansatz ebenfalls integriert zur Verfügung.

Die Custom Security sollte nur für nachgelagerte Auswertungen der Sicherheitsmerkmale verwendet werden. Die Basisauthentifizierung sollte durch die zuvor beschriebenen Standardmechanismen erfolgen.

Wie die Ausführungen gezeigt haben, stehen der sicheren Anwendung von SOAP-Diensten über unsichere Netze heutzutage nur noch kleinere Unwegbarkeiten im Wege. Somit sollte die vielfach praktizierte Sicherung der Dienste durch Geheimhaltung der Adresse (security by obscurity) bald der Vergangenheit angehören.

[1] http://ws.apache.org/axis2/

[2] http://tomcat.apache.org/

[3] JAVA Web Services mit Apache AXIS; Dapeng Wang (Hrsg.); Software&Support Verlag, 2004

[4] Tomcat 5 – Einsatz in Unternehmensanwendungen mit JSP und Servlets; Lajos Moczar; Addison-Wesley, 2004

[5] http://ws.apache.org/wss4j/

[6] http://ws.apache.org/axis2/modules/wss4j/1_0/security-module.html

[7] http://lab.lpicn.org/pub/books/FAQ_Tomcat_SOAP_SSL.pdf

[8] http://www.javaranch.com/journal/200603/WSSecurity.html

[9] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[10] http://ws.apache.org/wss4j/

[11] http://ws.apache.org/axis2/modules/wss4j/1_0/security-module.html

[12] http://www.wso2.net/presentations/wss4j/java/2006/08/04/apache-rampart

[13] http://www.wso2.net/tutorials/wss4j/2006/06/15/setting-up-keystores

Einschub: Tomcat auf SSL vorbereiten

Um durch SSL gesicherte Verbindungen aufbauen zu können, muss der Server ein entsprechendes Zertifikat bereitstellen. Es ist möglich selbstsignierte Zertifikate oder Zertifikate, die durch öffentliche Stellen vergeben und beglaubigt wurden, zu verwenden. Im Folgenden werden selbstsignierte Zertifikate, die mit Hilfe eines privaten Schlüssels signiert wurden, betrachtet. Ein so beglaubigtes Zertifikat wird Tomcat in einem vertrauenswürdigen Schlüsselspeicher zur Verfügung gestellt.

Die folgenden Ausführungen beziehen sich auf eine Windows-XP Installation. Die notwendigen Schritte unter LINUX-Distributionen sind analog zu diesen. Alternativ zu dem verwendeten JAVA-Tool können Zertifikate auch mit OpenSSL (http://www.openssl.org) erstellt werden.

Erzeugen eines Schlüssel und Zertifikat
Um einen Schlüssel zu erzeugen, mit dessen Hilfe Zertifikate beglaubigt werden können, wird das mit dem JDK installierte Tool `keytool` verwendet. Führen Sie in einer MS-Dos-Box folgenden Befehl aus, der einen RSA-Schlüssel „tomcat“ der Länge 1024 mit dem namen „SOAP-SERVER“ in einem Basisschlüsselspeicher (keystore) in dem conf-Verzeichnis von Tomcat ablegt. Mit dem Wert, der hinter „CN=“ angegeben wird wird der Common Name angegeben, der den Eigentümer des Zertifikats bezeichnet. <PW1> und <PW2> sind durch die Passwörter für den Schlüssel und den Schlüsselspeicher zu ersetzen.

```
%JAVA_HOME%\jre\bin\keytool -genkey -alias tomcat -keyalg RSA -keysize 1024
-dname "CN=SOAP_SERVER" -keypass <PW1> -storepass <PW2> -keystore
%CATALINA_HOME%\conf\keystore
```

Im nächsten Schritt wird ein Zertifikat mit der Bezeichnung tomcat-server.crt erzeugt und mit dem erstellten Schlüssel signiert. Dieses Zertifikat wird ebenfalls in den Schlüsselspeicher abgelegt. Nach eingabe des Befehls

```
%JAVA_HOME%\jre\bin\keytool -export -alias tomcat -file
%CATALINA_HOME%\conf\tomcat-server.crt -keystore
%CATALINA_HOME%\conf\.keystore
```

wird das Schlüsselspeicher-Passwort erfragt und das Zertifikat abgelegt. Um das Zertifikat verwenden zu können, muss dieses Tomcat in einem Schlüsselspeicher zur Verfügung gestellt werden. Mit dem folgenden Befehl wird das Zertifikat in den Schlüsselspeicher cacerts übernommen (und dieser ggf. angelegt).

```
%JAVA_HOME%\jre\bin\keytool -import -file tomcat-server.crt -keystore
%CATALINA_HOME%\CONF\cacerts
```

Da hier auf den Basisschlüsselspeicher zugegriffen wird, muss das Passwort eingegeben und das Vertrauen bestätigt werden.

Tomcat konfigurieren

Um das Zertifikat zu verwenden, muss die Konfiguration von Tomcat angepasst werden. Hierfür muss die Datei `%CATALINA_HOME%\conf\Server.xml` um die Zeilen in dem folgenden Listing nach der Connector-Definition für Port 8080 in der Standardkonfiguration eingefügt werden.

```
<Connector port="8443"
          maxThreads="150" minSpareThreads="25"
          maxSpareThreads="75"
          enableLookups="false" disableUploadTimeout="true"
          acceptCount="100" debug="0" scheme="https" secure="true"
          clientAuth="false" sslProtocol="TLS"
          keystoreFile="conf/.keystore"
          keystorePass="secret" truststoreFile="conf/cacerts"/>
```

TLS / SSL connector

Es wird ein Connector für Port 8443 erzeugt, der keine Client-Authentifizierung benötigt und als Protokoll TLS verwendet. In den letzten Parametern werden der Basis-Schlüsselspeicher und das zugehörige Passwort und der Schlüsselspeicher der vertrauenswürdigen Zertifikate angegeben.

Ab diesem Zeitpunkt wird alle Kommunikation über Port 8443 verschlüsselt. Beim ersten Aufruf eines Servlets über diesen Port (nicht vergessen https statt http zu verwenden) muss der Client dem erstellen Zertifikat vertrauen.