Technology

Dit artikel behandelt de rol die de WS-Security specificatie speelt in de wereld van webservices. Bij het toepassen van webservice-technologieën waarbij communicatie en transacties tussen verschillende partijen plaatsvinden is het belangrijk om de integriteit en privacy te garanderen van de informatie die wordt uitgewisseld. De WS-Specificatie geeft ons een standaard die past in de visie van de interoperabiliteit van webservices.

WS-Security en webservices

Implementatie met XFire SOAP stack

Ronald van Aken
is werkzaam als consultant bij Sirius ICT solutions BV te Amsterdam

Allereerst wil ik ingaan op wat deze specificatie ons biedt met het toepassen van beveiliging en vervolgens een voorbeeld tonen hoe je dit kunt implementeren door middel van de XFire SOAP stack.

Beveiliging in de wereld van webservices

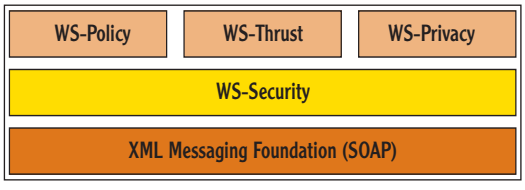
Het toepassen van beveiliging op client/server-applicaties zoals we dat kennen betekent veelal een extra laag aanbrengen over ons communicatieprotocol. Een van de bekendste toepassingen hiervan is het gebruik van SSL over HTTP. In de wereld van webservices wordt dit echter een stuk lastiger door het feit dat webservices niet protocol gebonden zijn. Een webservice kan als transport verschillende oplossingen gebruiken zoals FTP, Messaging of HTTP. Hierdoor kun je niet altijd uitgaan van beveiliging op transport niveau, de oplossing hiervoor is de WS-Security-specificatie. Deze specificatie stelt ons in staat om beveiliging toe te passen op bericht niveau-onafhankelijk van het transportprotocol wat we gebruiken. Dit geeft ons de transparantie om beveiliging te garanderen onafhankelijk van de wijze van communicatie, tevens is de wijze hoe we beveiliging toepassen uiterst flexibel zodat je niet meer doet dan nodig is.

Transport VS bericht beveiliging

Webservices worden meestal via HTTP beschikbaar gesteld, dit protocol heeft voorzieningen voor beveiliging op transport niveau in de vorm van SSL (HTTPS). Dit gaat uit van peer-to-peer en een SOAP-bericht kan over meerdere stations gaan voordat het bij zijn eind bestemming komt. In een dergelijk geval kan HTTPS een dure oplossing zijn door het vele encrypten en decrypten door de verschillende tussenstations. Daarnaast is HTTPS gericht op gebruik van HTTP als communicatie- en webservices-protocol (andere transportprotocollen kunnen bijvoorbeeld JMS of FTP zijn). Doordat bij het gebruik van webservices het transportprotocol niet altijd een gegeven feit is, heeft men WS-Security geïntroduceerd. De WS-Security-specificatie definieert hoe we beveiliging op berichtniveau kunnen gebruiken via verschillende mechanismen afhankelijk van de gestelde eisen. Dit houdt in dat het SOAP-bericht wordt beveiligd in tegenstelling tot het transportprotocol. Uiteraard kun je nog steeds het transportprotocol beveiligen wanneer je beveiliging op berichtniveau toepast.

WS-Security specificatie

De WS-Security specificatie is één van de vele deelgebieden binnen de vele specificaties die voorhanden zijn binnen de wereld van webservices. De specificaties zijn zodanig opgebouwd dat ze over elkaar heen en in combinatie met elkaar



Figuur 1. Overzicht van webservices specificatie stack.

kunnen worden gebruikt. Een overzicht van de belangrijkste specificaties is weergegeven in Figuur 1.

Het doel van de WS-Security specificatie is om een flexibele set van mechanismen te definiëren die ons in staat stellen beveiliging op berichtniveau toe te passen. De specificatie is zo opgezet dat verschillende vormen van encryptie, beveiliging-tokens en domeinen zijn toe te passen. Daarnaast zijn er aanvullingen op deze specificatie beschikbaar die specifiek ingaan op het gebruik van Kerberos en andere specifieke beveiligingsoplossingen. De specificaties kun je vinden op de website[1] van OASIS. OASIS is een niet-commerciële organisatie die zich inzet voor het definiëren van e-business standaarden.

Wanneer je de aspecten van beveiliging voor enterprise applicaties uiteenzet dan kunnen we de volgende niveaus onderkennen:

- *Integriteit*, zekerheid geven dat niemand kan knoeien met het bericht
- *Privacy*, zorg dragen dat het bericht niet door andere kan worden gelezen
- *Authenticatie*, aanleveren van bewijs van identiteit
- *Autorisatie*, het beschermen van resources

Binnen WS-Security vindt je drie basismechanismen die in combinatie met elkaar kunnen worden gebruikt voor het toepassen van de eerder beschreven beveiliging niveaus. Welke mechanismen je nodig hebt, hangt af van de eisen die worden gesteld qua beveiliging.

- Digitale handtekeningen worden gebruikt voor integriteit.
- Encryptie van het bericht dient voor privacy.
- Gebruikersnaam/wachtwoord voor het verzenden van gegevens voor identificatie en autorisatie.

Bij het gebruik van WS-Security zie je vaak dat men kiest tussen combinaties van signatures, encryptie van het bericht of het doorgeven van gebruikersnaam en wachtwoord. Dit geeft ons een krachtig mechanisme waarmee we de mate van beveiliging flexibel in het bericht kunnen toepassen. Een uitwerking die veel wordt toegepast is bijvoorbeeld het toevoegen van een digitale handtekening bij het bericht en delen van het bericht encrypten. Voorzichtigheid is geboden bij het maken van de keuzes welke mechanismen je gebruikt. Encryptie van het bericht is bijvoorbeeld bijzonder duur qua processorgebruik en beïnvloedt de performance van verwerking aanzienlijk.

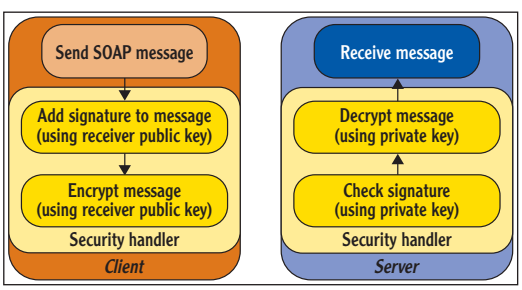
WS-Security en SOAP

Wanneer je de SOAP-specificatie leest dan zie je al snel dat SOAP-berichten kunnen worden uitgebreid met extra informatie. Wanneer we WS-Security gebruiken dan wordt er aan de SOAP-envelop een zogenaamde SOAP-header toegevoegd. De SOAP-header beschrijft welke vorm van security binnen het bericht is toegepast. De ontvanger zal door middel van de header in het bericht de nodige acties verrichten of een bericht terugsturen dat de headers niet kunnen worden verwerkt. Het verwerken van SOAP-headers gebeurt vaak door zogenaamde intermediaries. Dit zijn tussenstations die het bericht kunnen bewerken voordat het fysiek wordt bezorgd bij de webservice. Intermediaries zijn binnen de meeste Java webservices frameworks geïmplementeerd via Handlers.

Handlers kun je vergelijken met interceptors of decorators die het inkomende of uitgaande bericht kunnen bewerken voordat het bij het eindstation wordt bezorgd. Intermediaries kunnen zowel op de client als de server worden gebruikt, denk bijvoorbeeld aan een client die een bericht ontvangt waarop encryptie is toegepast. Naast de Handlers die door de frameworks worden geleverd kun je gemakkelijk je eigen Handlers schrijven en gebruiken, vooral voor het realiseren van niet-functionele eisen zijn Handlers vaak een goede oplossing. Hiermee kun je een mooie scheiding maken tussen de business-logica en niet-functionele randvoorwaarden zoals loggen, beveiliging en het bewaren van berichten als bewijs van ontvangst (non-repudiation). Het grootste voordeel bij deze opzet is het hergebruik en de flexibiliteit om handlers aan elkaar te koppelen zonder dat je business-logica hoeft te wijzigen (Handler chain pattern GoF[2]).
De benodigde beveiligingsinformatie wordt toegevoegd aan het SOAP bericht, maar hoe weet een client wat voor mate van beveiliging de server verwacht. Dit is opgelost in de WS-Policy specificatie die een profiel definieert waarmee beveiligingsaspecten in de WSDL kunnen worden toegevoegd.

Toepassen van WS-Security met XFire

Het toepassen van beveiliging voor XFire webservices wordt volledig via de configuratie gerealiseerd. Je kunt daarnaast kiezen voor verschillende



Figuur 2. Gebruik van Handlers bij de aanroep en ontvangst van bericht

implementaties voor het fysiek afhandelen van de beveiliging. Dit wordt mogelijk gemaakt door de flexibele opzet van XFire en het gebruik van dependency-injectie via de configuratie bestanden. Ik zal voor dit artikel gebruikmaken van de WSS4J bibliotheek van Apache welke ook bijvoorbeeld in Axis 1.x wordt gebruikt.

Wanneer je beveiliging wilt gebruiken zal je een intermediary moeten definiëren in de configuratiebestanden, die kan dan worden gekoppeld aan de gewenste webservice. In het onderstaande voorbeeld definieer ik een Bean voor het afhandelen van signatures en encryptie. Het soort beveiliging wat je wilt toepassen kun je specificeren in de ACTION parameter. Je kunt hierin verschillende beveiligingsniveaus combineren. De configuratie van de handler voor WS-Security ziet er als volgt uit:

```
<bean id="domInHandler"
               class="org.codehaus.xfire.util.dom.
DOMInHandler"/>

<bean id="wss4jInHandlerEncSign"
         class="org.codehaus.xfire.security.wss4j.
WSS4JInHandler">
      <property name="properties">
         <props>
              <prop key="action">Signature Encrypt</
prop>
            <prop key="user">clientalias</prop>
            <prop key="encryptionuser">clientalias</
prop>
            <prop key="decryptionPropFile">
META-INF/xfire/insecurity.properties
      </prop>
    <prop key="signatureKeyIdentifier">IssuerSerial</
prop>
            <prop key="signaturePropFile">
META-INF/xfire/insecurity.properties
      </prop>
            <prop key="passwordCallbackClass">
      highview.util.PasswordHandler
      </prop>
         </props>
      </property>
</bean>
```

De voorgaande code toont hoe we de `wss4jInHandlerEncSign` Bean configureren voor het toepassen van decryptie en het controleren van de in het bericht opgenomen signature op basis van certificaten op de server. De property-bestanden die worden opgegeven bevatten de informatie die nodig is voor deze Handler om de daadwerkelijke klasse te instantiëren die de acties kan uitvoeren en deze te voorzien met de benodigde configuratie argumenten. Het configuratiebestand dat ik gebruik ziet er als volgt uit:

```
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=keystorePass
org.apache.ws.security.crypto.merlin.alias.
password=keystorePass
org.apache.ws.security.crypto.merlin.keystore.
alias=clientalias
org.apache.ws.security.crypto.merlin.file=META-INF/
xfire/privateStore.jks
```

Dit soort configuratiebestanden zijn afhankelijk van welke implementatie je gebruikt. In het voorbeeld gebruiken we de WSS4J-implementatie van Apache. Andere bibliotheken zullen een andere aanpak vereisen.
Uiteraard zul je de benodigde certificaatbestanden aanmaken en toegankelijk maken voor je applicatie. Je kunt dit doen met de key-tool die wordt meegeleverd met de JDK. Binnen een productie-omgeving zul je de certificaten via een leverancier als Verisign laten aanleveren. Het aanmaken van de benodigde bestanden (certificaten en keystores) kun je via dit voorbeeldscript in realiseren:

```
del *.jks
del *.rsa
keytool -genkey -alias clientalias -keystore priva-
teStore.jks
-storepass keystorePass -dname "cn=clientalias"
-keyalg RSA
keytool -selfcert -alias clientalias -keystore pri-
vateStore.jks
-storepass keystorePass -keypass keystorePass
keytool -export -alias clientalias -file key.rsa
-keystore
privateStore.jks -storepass keystorePass
keytool -import -alias clientalias -file key.rsa
-keystore
publicStore.jks -storepass keystorePass
```

Dit script levert een bestand op voor een publieke en geheime sleutel. De laatste moet worden gebruikt door de server en de publieke door de client.

In het eerste codevoorbeeld is daarnaast een `domInHandler` geconfigureerd. Deze is nodig om de XML te standaardiseren naar een formaat waarmee de klasse die de beveiliging implementeert kan omgaan.

We zijn nu klaar om de daadwerkelijke webservice te configureren, zodat onze security Handler bij aanroep wordt gebruikt om de nodige acties uit te voeren. De configuratie, het toevoegen van de security handler aan de service ziet er als volgt uit:

```
bean id="RegistrationBeanTarget"
   class="registration.impl.RegistrationImpl">
</bean>

<bean id="bookService"
  class="org.codehaus.XFire.spring.remoting.
XFireExporter">
                <property     name="serviceBean"
ref="RegistrationBeanTarget"/>
```

In de ACTION parameter kun je hierin verschillende beveiligingsniveaus combineren

```
   <property name="serviceClass"
 value="registration.Registration"/>
   <property name="inHandlers">
      <list>
         <ref bean="domInHandler"/>
         <ref bean="wss4jInHandlerEncSign"/>
      </list>
   </property>
</bean>
```

Je kunt zien dat we een property `inHandlers` hebben toegevoegd met daarin op volgorde de handlers zoals ze moeten worden doorlopen. Voor uitgaande handlers hanteer je hetzelfde principe. We zijn nu klaar om onze applicatie te deployen naar de applicatieserver. Je zult alleen wel merken dat dit niet zomaar gaat. Je zult een fiks aantal JAR-bestanden moeten toevoegen en in het geval dat je WSS4J gebruikt moet je ook Xalan beschikbaar stellen.

Toevoegen van beveiliging aan de client

Aangezien onze service nu encryptie en een signature verwacht, zullen we wat extra werk moeten verrichten op de client. Het principe is gelukkig hetzelfde. Het is meer een kwestie uit te zoeken hoe we een Handler toevoegen aan onze client:

```
RegistrationClient      client      =      new
RegistrationClient();
RegistrationPortType  serv  =  client.getRegistrati-
onHttpPort();
Client client = Client.getInstance(serv);

// Configure outgoing security (REQUEST message)
client.addOutHandler(new DOMOutHandler());
Properties properties = new Properties();
// Set properties for WSS4JHandler
properties.setProperty(WSHandlerConstants.ACTION,
                                WSHandlerConstants.
SIGNATURE+
                            " "+WSHandlerConstants.
ENCRYPT);
properties.setProperty(WSHandlerConstants.USER,
"clientalias");
properties.setProperty(WSHandlerConstants.SIG_PROP_
FILE,
                                       "outsecurity.
properties");
properties.setProperty(WSHandlerConstants.SIG_KEY_
ID,
                                    "IssuerSerial");
properties.setProperty(WSHandlerConstants.USER,
                                       "clientalias");
properties.setProperty(WSHandlerConstants.PW_
CALLBACK_CLASS,
                                PasswordHandler.class.
getName());
properties.setProperty(WSHandlerConstants.ENC_PROP_
FILE,
                                       "outsecurity.
properties");
client.addOutHandler(newWSS4JOutHandler
(properties));
Member member = new Member();
String reply = serv.registerMember(member);
```

Wellicht heb je opgemerkt dat ik zowel op de server als op de client een property definieer voor passwordCallbackClass. Deze klasse moet je meeleveren en wordt gebruikt door de Handlers om het daadwerkelijke password te krijgen. De code voor deze klasse ziet er zo uit:

```
public    class    PasswordHandler    implements
CallbackHandler {

  private Map passwords = new HashMap();

  public PasswordHandler() {
    passwords.put("clientalias", "keystorePass");
  }

   public void handle(Callback[] callbacks) throws
IOException,
                    UnsupportedCallbackException {
      WSPasswordCallback pc = (WSPasswordCallback)
callbacks[0];
    String id = pc.getIdentifer();
    pc.setPassword((String) passwords.get(id));

  }

}
```

De applicatie is na het doorvoeren van de wijzigingen klaar om te testen. Het is raadzaam om een tool als TCPMon te gebruiken zodat je kunt zien wat de client verstuurt naar de server. Problemen die we kunnen tegenkomen zitten meestal in het feit dat de server of client JAR-bestanden mist of de benodigde .JKS bestanden niet kan vinden. Axis 1.x hanteert dezelfde techniek. Wanneer je het gebruik van WSS4J kent, kun je het dus in meerdere oplossingen gebruiken.

Ik hoop dat via dit artikel het toepassen van WS-Security binnen Java webservices is verduidelijkt. Voor meer informatie zou ik willen adviseren om eerst de relevante WS-* specificaties te lezen die beschikbaar zijn via OASIS en W3C en daarna pas te bepalen hoe je dit gaat toepassen door middel van XFire of andere producten zoals Axis.

De toekomst voor X-Fire

De volgende (2.0) release van X-Fire slaat een nieuwe weg in. Deze versie zal een samenvoeging zijn van Celtix en X-Fire. Het resultaat zal publiekelijk beschikbaar zijn via het CFX project van Apache. De actie heeft als hoofddoel om de kern van X-Fire grondig aan te pakken en beter te confirmeren aan nieuwe specificaties zoals WS-* en JAX-WS en gebruik te maken van Celtix elementen.

Referenties

1. http://www.oasis-open.org/specs/index.php
2. Design patterns elements of reusable Object-Oriented Software, Gamma

De volgende release van X-Fire zal een samenvoeging zijn van Celtix en X-Fire