Enterprise

Beveiliging heeft in de afgelopen jaren een steeds grotere rol gekregen in de ICT-wereld. Een andere trend in ICT-land is de opkomst van Service-Oriented Architecture (SOA). In dit artikel gaat het om de vraag of beveiliging in een SOA-omgeving anders benaderd moet worden dan bij 'traditionele' systeemontwikkeling. Ook worden enkele veelvoorkomende beveiligingsmechanismen bekeken en wordt besproken in hoeverre de Enterprise Service Bus (ESB) geschikt is om beveiliging in een SOA te regelen.

Beveiliging in een SOA-omgeving

Geen triviale taak

Ronald van Luttikhuizen
Approach Alliance

Waarom is beveiliging van belang in een SOA-omgeving? Bijna iedereen zal het belang van beveiliging binnen systeemontwikkeling onderkennen. Vanuit deze aanname is de vraag of beveiliging van een SOA-omgeving nodig is nauwelijks meer interessant te noemen. Iedereen zal deze stelling namelijk beamen. Belangrijker is het antwoord op de vraag of beveiliging *anders* aangepakt moet worden? En zo ja, waarom dan? Om deze vraag te kunnen beantwoorden, kijken we eerst naar de relevante verschillen tussen traditionele systeemontwikkeling en SOA, en hoe deze het beveiligingsvraagstuk beïnvloeden.

De volgende karakteristieken van een SOA-omgeving spelen in dit kader een rol:

- Naast mens-applicatie-interactie is er vaker sprake van applicatie-applicatie-interactie. Voor deze koppelingen is veelal een geautomatiseerde vorm van authenticatie, autorisatie, encryptie, enzovoort vereist.
- Er zijn meer tussenstations (bijvoorbeeld een ESB), en dus zijn er meer mogelijkheden om de inhoud van berichten in te zien. Niet elk tussenstation zal dit mogen. In deze situatie is beveiliging van transport niet genoeg.
- Hoe controleer je of een applicatie data mag ophalen en/of services mag aanroepen? Niet iedereen zal een service om te wisselen van verzekeraar mogen gebruiken.
- Er is vaker sprake van 'Straight Through Processing' (STP) waarbij een proces geheel elektronisch afgehandeld wordt zonder menselijke tussenkomst. Goede beveiliging is erg belangrijk aangezien mogelijke gevolgen eventueel pas laat ontdekt worden. Het hele proces verloopt immers automatisch.
- Services worden gebruikt door verschillende interne en externe afnemers. In de eerste plaats wordt het beveiligingsniveau van een service bepaald door de eigenaar; vaak afhankelijk of afnemers intern dan wel extern zijn. Beveiliging ligt voor afnemers dus deels buiten hun eigen 'span of control'. Belangrijk is dat de mate van beveiliging het vertrouwen van afnemers bepaalt: "Wat gebeurt er met mijn data als de service niet beveiligd is?", "Kan ik het antwoord van een service wel vertrouwen?", enzovoort.

Deze eigenschappen zorgen er voor dat het beveiligingsvraagstuk in een SOA-omgeving anders benaderd moet worden.

Voordat we ingaan op de vraag waar beveiliging afgedwongen kan worden in een SOA-omgeving, bespreek ik eerst hoe een SOA beveiligd kan worden. Er worden twee veel voorkomende beveiligingsmechanismen uitgewerkt.

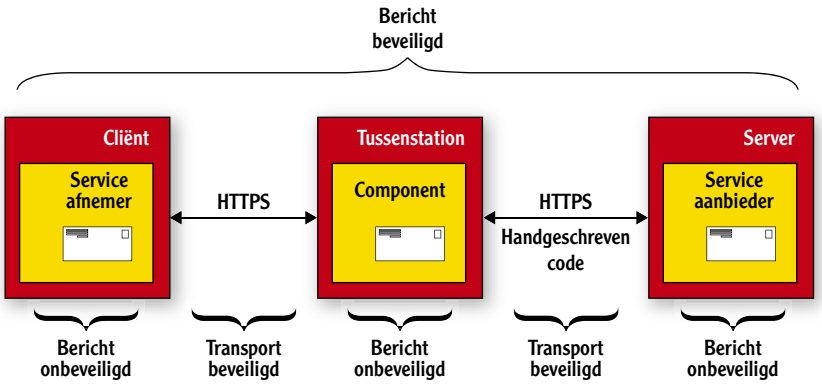
Beveiliging van transport versus beveiliging van bericht

Beveiliging kan op verschillende manieren toegepast worden. Vaak gebruikte mechanismen zijn de beveiliging van transport en de beveiliging van bericht.

Bij transportbeveiliging zijn berichten *alleen tijdens transport* beschermd; bijvoorbeeld via een transportprotocol of het gebruik van eigen

communicatielijnen. Beveiliging vindt plaats vanaf verzending tot aan ontvangst, niet als deze berichten opgeslagen zijn of verwerkt worden. Gaan we uit van webservices die aangeroepen worden via HTTP –een veelgebruikte variant in een SOA-landschap– dan is de meest voorkomende vorm van transportbeveiliging HTTPS, ofwel HTTP over Secure Socket Layers (SSL) of Transport Layer Security (TLS). Deze 'point-to-point' beveiliging is geïmplementeerd in het transportprotocol met encryptie van het bericht. Via HTTPS kan een bepaalde mate van integriteit (wie heeft het bericht verstuurd?), authenticatie (wie mag de service aanroepen?) en vertrouwelijkheid (heeft niemand anders het bericht gelezen?) gegarandeerd worden. Over deze vorm van beveiliging kan meer gelezen worden in het Java Magazine artikel "Secure Sockets in Java" van maart 2008.

Bij berichtbeveiliging wordt *het bericht zelf* beveiligd. Beveiliging vindt dus op een ander niveau plaats en is daarmee onafhankelijk van het transportprotocol. Een belangrijke consequentie is dat beveiliging niet alleen tijdens transport plaatsvindt, maar toegepast wordt wanneer dat nodig is. Zo kan men bijvoorbeeld berichten met creditcardgegevens beveiligen terwijl deze in tussenstations of bepaalde delen van verzender of ontvanger 'in rust' zijn. Berichtbeveiliging biedt –in tegenstelling tot transportbeveiliging– de mogelijkheid tot 'end-to-end' beveiliging. De volgende afbeelding geeft dit verschil grafisch weer.



Afbeelding 1. Het verschil tussen transportbeveiliging en end-to-end berichtbeveiliging

Voorbeeld

We gaan uit van een webservice met de operatie echoMe. Deze operatie retourneert de input als output. Een SOAP-aanroep zonder beveiliging ziet er als volgt uit:

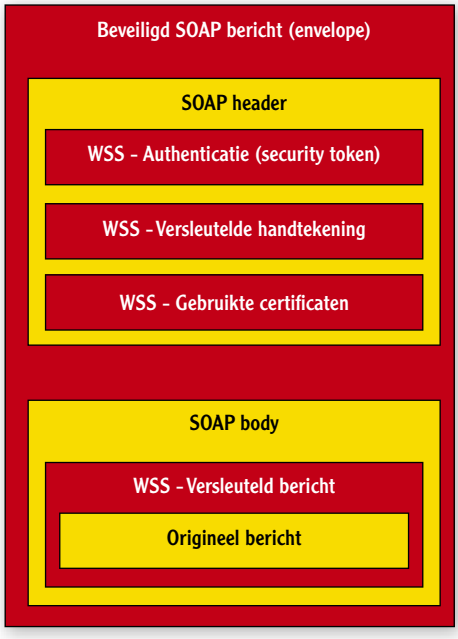
```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/enve-
lope/"
xmlns:foo="http://www.foo.com/">
   <!-- bericht header -->
   <soapenv:Header/>
   <!-- bericht inhoud -->
   <soapenv:Body>
      <foo:echoMe>
         <foo:echoText>Hello world!</foo:echoText>
      </foo:echoMe>
   </soapenv:Body>
</soapenv:Envelope>
```

Voorbeeld 1. SOAP-aanroep zonder beveiliging

De belangrijkste webservice-standaard voor berichtbeveiliging is WS-Security (WSS) en wordt beheerd door OASIS. WSS biedt onder andere het volgende:

1. **Authenticatie** - Toevoegen van authenticatiegegevens aan een bericht via 'security-tokens': naam/wachtwoord, X.509 certificaat, SAML, Kerberos, enzovoort. Dit garandeert dat alleen bepaalde partijen een service mogen afnemen.
2. **Vertrouwelijkheid** - Encryptie van het bericht via de public key van de aanbiedende partij en een algoritme zoals RSA. Niemand anders dan de eigenaar van de bijbehorende private key –de aanbieder van de service– kan het bericht lezen.
3. **Integriteit** - Ondertekenen van het bericht via de private key van de afnemer. Als de aanbieder via de public key van de afnemer de handtekening kan 'ontsleutelen' weet men dat de eigenaar van de private key –de afnemer– het bericht ondertekend heeft. Doordat deze handtekening een hash is van de berichtinhoud weet men dat er niet geknoeid is met het bericht.

Een met WS-Security beveiligd SOAP-bericht heeft de volgende structuur: (zie Afbeelding 2)



Afbeelding 2. Structuur van een met WS-Security beveiligd SOAP-bericht

De serviceaanbieder retourneert het volgende foutbericht aangezien beveiliging vereist is:

```
<env:Envelope   xmlns:env="http://schemas.xmlsoap.org/
soap/envelope/">
    <env:Header/>
    <env:Body>
       <!-- fout -->
       <env:Fault
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd">
          <faultcode>wsse:InvalidSecurity</faultco-
de>
        <faultstring>Missing <wsse:Security> in SOAP
Header</faultstring>
          <faultactor/>
       </env:Fault>
    </env:Body>
</env:Envelope>
```

Voorbeeld 2. Foutbericht

Beveiliging van het bericht via WSS resulteert in een tweetal wijzigingen. De berichtinhoud bevat versleutelde gegevens in plaats van leesbare tekst:

```
<env:Body
wsu:Id="iwKErOG8Cf7pWtsQx4MR5w22"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd">
     <!-- header die aangeeft dat het om versleutelde
data gaat -->
      <xenc:EncryptedData
xmlns="http://www.w3.org/2001/04/xmlenc#"
xmlns:xenc=http://www.w3.org/2001/04/xmlenc#
Type="http://www.w3.org/2001/04/xmlenc#Content"
Id="_Dff7ySASsISfb2H31osV8A22">
          <!-- encryptie algoritme -->
          <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/
xmlenc#tripledes-cbc"/>
          <!-- versleutelde data -->
          <xenc:CipherData>

<xenc:CipherValue>Eifd1gVl41qNg48…qBAYZQI4UJU2s4=</
xenc:CipherValue>
          </xenc:CipherData>
      </xenc:EncryptedData>
 </env:Body>
```

Voorbeeld 3. Berichtinhoud met versleutelde gegevens

Daarnaast is een WS-Security header toegevoegd. Deze bevat onder andere authenticatiegegevens, een handtekening als XML Digital Signature en de gebruikte certificaten voor versleuteling en ondertekening. Aangezien de header te groot is om in zijn geheel op te nemen, zijn bepaalde details weggelaten.

```
<env:Header>
       <wsse:Security>
       <!-- X.509 certificaat gebruikt voor encryptie
-->
                          <wsse:BinarySecurityToken
ValueType="wsse:X509v3"/>
             MIICUTCCAbqgA…awxekHKkTWS2az
          </wsse:BinarySecurityToken>
          <!-- encryption key gebruikt voor de ver-
sleuteling van het bericht -->
          <xenc:EncryptedKey>… </xenc:EncryptedKey>
     <!-- X.509 certificaat gebruikt voor handtekening
-->
          <wsse:BinarySecurityToken ValueType="wsse:
X509v3"/>
             MIICRzCCAbCgA…9ssBsDFmgT2ASO=
          </wsse:BinarySecurityToken>
          <!-- handtekening -->
          <dsig:Signature>
             <dsig:SignedInfo>
          <!-- normalisatiemethode (canonicalization)
-->
    <dsig:CanonicalizationMethod Algorithm="http://www.
w3.org/2001/10/xml-exc-c14n#"/>
                <!-- hash van het bericht (digest)
-->
     <dsig:DigestValue>odVpOoTtu7BRBJhAxgxSMQssRdI=</
dsig:DigestValue>
                 <!-- "daadwerkelijke" handtekening
-->
                  <dsig:SignatureValue>H7MoXu2JdPx2…
HOVdTqrylXDAg=</dsig:SignatureValue>
          </dsig:Signature>
           <!-- authenticatie, ter simplificering in
"clear text" -->
          <wsse:UsernameToken>
             <wsse:Username>noot</wsse:Username>
             <wsse:Password>mies</wsse:Password>
          </wsse:UsernameToken>
          <!-- timestamp -->
          <wsu:Timestamp>
                <wsu:Created>2008-04-16T21:10:09Z</
wsu:Created>
                <wsu:Expires>2008-04-17T05:10:09Z</
wsu:Expires>
          </wsu:Timestamp>
      </wsse:Security>
   </env:Header>
```

Voorbeeld 4. WS-Security header toegevoegd

De keuze tussen verschillende beveiligingsmechanismen, standaarden en implementaties zoals HTTPS en WS-Security is meestal geen gemakkelijke. Dit komt vooral door het grote aantal randvoorwaarden, zoals de (on)mogelijkheden van tools, platformen en te integreren legacy-systemen, de beschikbare kennis en eisen van (externe) afnemers en aanbieders van services. Om dit te illustreren volgt een tweetal voorbeelden.

Voorbeeld 1 - Een legacy-systeem wordt afnemer van een nieuwe service. De programmeertaal waarin het legacy-systeem ontwikkeld is, ondersteunt geen webservices maar wel fixed-length berichtenverkeer over HTTP. Transportbeveiliging via HTTPS is in dit geval een goede keuze.

Voorbeeld 2 - Men neemt diensten af van een externe serviceaanbieder die geen HTTP(S) ondersteunt, maar wel SMTP en RMI. In dit geval is transportbeveiliging via HTTPS geen optie, maar berichtbeveiliging via WS-Security wellicht wel.

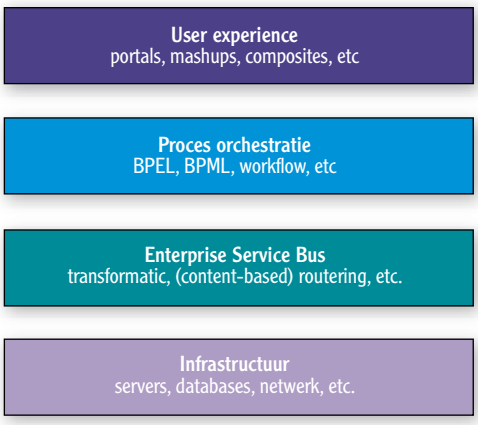
Aandachtspunten

Om implementatiedetails te scheiden van beveiligingsbelangen bevat een WSDL geen WS-Security en HTTPS-details. Dit kan verwarrend zijn als men op basis van contract (WSDL) ontwikkelt en een serviceaanbieder niet vertelt hoe een certificaat gebruikt moet worden, alleen dat het gebruikt moet worden. Vanwege de heterogeniteit van SOA-omgevingen speelt de interoperabiliteit van webservices een grote

rol. Het heeft immers weinig zin om functionaliteit als webservices te verpakken als een Java-programma geen Microsoft webservice kan aanroepen. De Web Services Interoperability Organization (WS-I) heeft als doel om interoperabiliteit van webservices en standaarden te bevorderen. Het WS-I heeft in zijn "Basic Profile" vastgelegd aan welke standaarden webservices moeten voldoen. Basic Profile omvat echter geen beveiligingsstandaarden. Hiervoor heeft WS-I vorig jaar "Basic Security Profile" opgeleverd. Hierin is ook standaardisatie en naleving van WS-Security opgenomen.
Het testen van een SOA-omgeving omvat zowel complete integratietesten als unittesten van services. Bij unittesten is het belangrijk dat serviceafnemers en serviceaanbieders los van elkaar getest kunnen worden. Het testen van serviceafnemers is mogelijk door services te 'mocken'. Dit houdt in dat een testtool op basis van contract (WSDL en XSD) een service nabootst. Vice versa kunnen serviceaanbieders getest worden doordat de testtool zich als afnemer gedraagt. Ook dit gebeurt op basis van het servicecontract. Behalve een betere kwaliteit heeft dit ook tot gevolg dat services en hun afnemers sneller geïmplementeerd kunnen worden. Voor unittesten van webservices, inclusief ondersteuning voor HTTPS en WSS, is SoapUI een goede keus.
Nu we meer weten over enkele veelvoorkomende beveiligmechanismen gaan we verder met de vraag waar beveiliging in een SOA-omgeving toegepast kan worden.

De rol van een ESB bij beveiliging van een SOA

Hoewel SOA-platformen onderling verschillen, ziet een 'doorsnee' SOA er grofweg als volgt uit:



Afbeelding 3. Een 'doorsnee' SOA-platform

Services kunnen –afhankelijk van hun definitie– op elk van bovenstaande lagen voorkomen. Verder moet opgemerkt worden dat niet elke SOA-omgeving een orchestratielaag bevat.
Meerdere van deze componenten kunnen beveiliging toepassen:

- Services die beveiliging zelf regelen.
- Een aparte beveiligingscomponent zoals AmberPoint SOA Management System en Oracle Web Service Manager.
- Een ESB die beveiliging toepast.

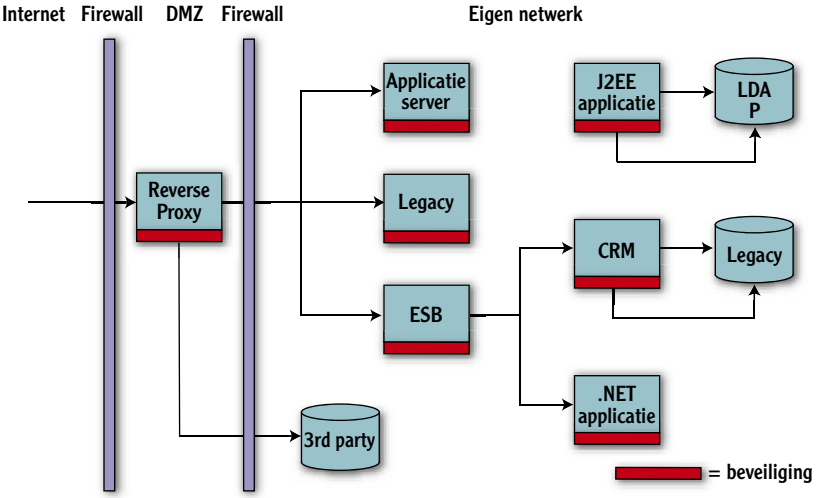
Afbeelding 4 op de pagina hiernaast illustreert verschillende vormen van beveiliging in een SOA-omgeving.

Eén van de richtlijnen in een SOA-omgeving is het scheiden van businesslogica en infrastructuur. Dit vereenvoudigt het definiëren, implementeren en orchestreren van services. Vanuit dit perspectief is het wenselijk om applicaties – lees: services – los te koppelen van hun beveiliging. Dit vereist dat de infrastructuur beveiliging aanbiedt en zorgt voor de naleving ervan. Deze aanpak maakt het mogelijk om beveiliging als – mogelijk herbruikbare – set van services beschikbaar te stellen. Deze kunnen net als alle andere services gecombineerd worden om het juiste resultaat te bereiken. Dit wordt ook wel 'mediation' genoemd: het toepassen van een keten van verschillende services, ofwel 'mediators'. Zo kan men op een bericht encryptie toepassen (beveiligingsservice), gevolgd door ondertekening van het bericht (beveiligingsservice) om ten slotte het bericht te versturen naar een afnemer (routeringservice).
Waarom zou de ESB geschikt zijn om deze beveiligingsservices aan te bieden? Om deze vraag te beantwoorden kijken we eerst naar de rol van deze component in een SOA. Een ESB is een infrastructuurcomponent die applicaties en services integreert. Hij biedt functionaliteit voor datatransformatie, routering, messaging, virtualisatie en het aanbieden van adapters voor verschillende (legacy) systemen, technologieën en protocollen. Een ESB verbergt hiermee de onderliggende heterogeniteit van verschillende systemen. Aangezien services veelal via een ESB aangeroepen en/of beschikbaar gesteld worden, is dit een logische plaats om zogenaamde 'cross-cutting concerns' te regelen. Denk hierbij aan logging, auditing, monitoring en ook beveiliging. Dit maakt de ESB een geschikte plek voor beveiligingsservices. Een ESB past beveiliging vaak toe via 'agents' en 'gateways'. Een agent bevat de beveiligingsconfiguratie – ook wel policies genoemd – van één enkele service, terwijl een gateway de policies bevat die op meerdere (of alle) services van toepassing zijn. Een gateway kan een mogelijke 'single point of failure' of performancebottleneck creëren. Een mix van gateways en agents is over het algemeen een

goede zaak: gateways voor veelvoorkomende configuraties en agents voor afwijkende beveiligingseisen. (Zie afbeelding 5 op p. 52)

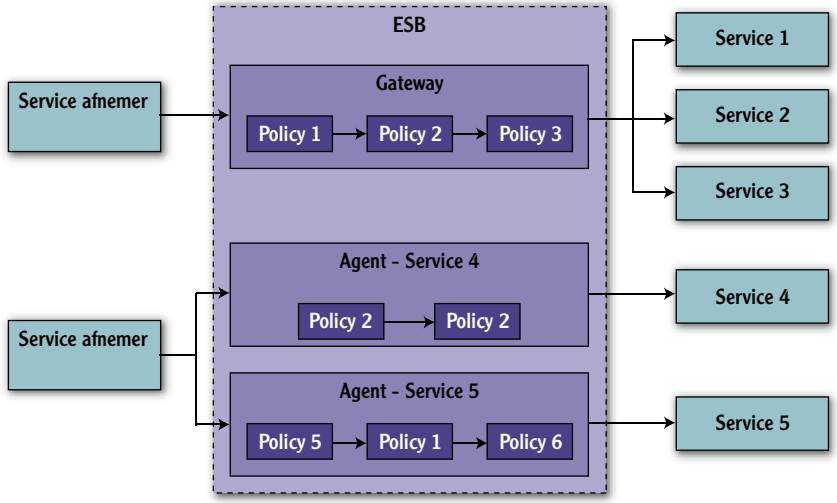
De transformatie van applicatiegecentreerde beveiliging naar beveiligingsservices via een ESB gaat vaak via een aantal stadia:

- **Applicatiebeveiliging** – Applicaties implementeren hun eigen beveiliging en passen deze zelf toe. Een eerste stap is om applicaties zo veel mogelijk gebruik te laten maken van gestandaardiseerde beveiligingsaanbieders via een API; bijvoorbeeld het gebruik van LDAP en JAAS.
- **Demilitarized Zone (DMZ)** – Het toevoegen van beveiligingscomponenten aan de rand van het netwerk. Componenten als een reverse proxy bieden over het algemeen een grove vorm van beveiliging om binnenkomend HTTP-verkeer te filteren op basis van authenticatie. Dit is een eerste vorm van een gateway.
- **Gateways** – Het toevoegen van gateways die onderdeel zijn van de ESB en beveiliging bieden voor andere transportprotocollen dan HTTP en daarnaast berichtbeveiliging kunnen toepassen.
- **Gateways en agents** – Specifieke beveiligingseisen van services toepassen via agents en gedeelde beveiliging toepassen via gateways. Zowel gateways als agents zijn onderdeel van een ESB en kunnen gecombineerd worden met overige services als transformatie- en content-based routing services.



Afbeelding 4. Verschillende vormen van beveiliging in een SOA-omgeving

Om te achterhalen welke beveiligingsservices nodig zijn, en hoe en op welke services deze toegepast moeten worden, is een antwoord op een



Afbeelding 5. Een mix van gateways en agents voor beveiliging is een goede zaak

aantal – eerder in dit artikel gestelde – vragen nodig.

Betreft het externe services, dan moet een ESB zich houden aan de beveiligingseisen van deze services en deze ondersteunen. Als het type beveiliging niet overeenkomt met die van de afnemer(s) van deze service dan zal een ESB de transformatie van beveiligingsmechanismen moeten ondersteunen; bijvoorbeeld transformatie van WS-Security berichtbeveiliging naar authenticatie via HTTPS. Gaat het echter om nieuw te ontwikkelen services die via de ESB beschikbaar gemaakt worden, dan zal het gebruik van standaarden om interoperabiliteit te waarborgen een belangrijke drijfveer zijn.

Kunnen we nu concluderen dat beveiliging in een SOA-omgeving altijd en alleen maar in de ESB toegepast moet worden? Nee. Wel dat een ESB vooral geschikt is voor beveiliging van applicatiecomponenten (services) en processen in een SOA-omgeving. Componenten die buiten de ESB om benaderd worden, zoals databases en user-interfaces dienen via een andere weg beveiligd te worden.

Vergelijking tussen verschillende ESB's

Om beveiliging in de ESB wat concreter te maken, volgt hieronder een korte vergelijking tussen de beveiligingsmechanismen van Oracle ESB, BEA AquaLogic Service Bus (ALSB) en Open ESB:

- **Oracle ESB** – Oracle ESB biedt zelf geen beveiligingsmechanismen. Beveiliging zoals HTTPS en WS-Security wordt toegepast en geconfigureerd via de onderliggende Oracle Application Server. Dit kan op webserviceniveau of via agents en gateways die door Oracle Web Service Manager (OWSM) geconfigureerd worden. OWSM is een externe component.
- **BEA AquaLogic Service Bus** – ALSB bevat een aantal functionele lagen waarvan de 'Security Layer' er één is. Naast standaard beveiligingscomponenten ten behoeve van transportbeveiliging, berichtbeveiliging, enzovoort kunnen ook eigen beveiligingscomponenten gedefinieerd worden. ALSB-beveiliging maakt gebruik van het beveiligingsframework van de onderliggende BEA WebLogic Server. Configuratie gebeurt zowel in ALSB als WebLogic.
- **OpenESB** – OpenESB is een ESB-implementatie op basis van Java Business Integration (JBI). Beveiliging in OpenESB is verdeeld over de 'Service Engines' (businesslogica) en 'Binding Components' (communicatielogica) componenten. Zo biedt de HTTP Binding Component (HTTP BC) zowel transport- als berichtbeveiliging. HTTP BC ondersteunt zowel Basic Profile als Basic Security Profile. Bovendien biedt de onderliggende applicatieserver –Glassfish– net als bij Oracle ESB en BEA ALSB ook verschillende beveiligingsmechanismen.

Niet triviaal

Het beveiligen van een SOA-omgeving is geen triviale taak. Door de specifieke karakteristieken van SOA is een andere blik op beveiliging nodig dan bij 'traditionele' systeemontwikkeling. Daarom is het belangrijk om beveiliging als apart onderwerp, en bijtijds, in een SOA-traject te behandelen. Een manier om dit te bereiken is de beveiligingsarchitectuur als een aparte view in een SOA-architectuur te definiëren.

Standaardisatie is een belangrijke vereiste in een SOA-omgeving. Dit geldt ook voor standaardisatie van beveiligingsmechanismen. Transportbeveiliging via HTTPS en berichtbeveiliging via WS-Security zijn voorbeelden van gestandaardiseerde beveiligingsmechanismen.

In een SOA-omgeving is de ESB een geschikte component om beveiliging van applicatiecomponenten (services) en processen toe te passen. Beveiliging via een ESB alleen is echter niet voldoende. Beveiliging kan via een ESB ontkoppeld worden van applicaties en als set van beveiligingsservices aangeboden worden. Deze kunnen door de ESB toegepast worden op services via gateways en agents. «