Erik S.C. van de Ven is senior consultant bij Microsoft Services Nederland

Webservices beveiligen wordt nog eenvoudiger met WSE 3.0 en Visual Studio 2005

WEBSERVICES BEVEILIGEN MET BEHULP VAN X509-CERTIFICATEN ZONDER TE PROGRAMMEREN

In een servicegeoriënteerde oplossing worden (autonome) services ontwikkeld die met elkaar communiceren op basis van berichten. Wanneer deze berichten gevoelige informatie bevatten en er bovendien gecommuniceerd moet worden met services van andere partijen, betekent dit dat er veel eisen worden gesteld aan beveiliging. Hoe kan worden voorkomen dat berichten door ongewenste personen worden gelezen? Hoe kan met zekerheid worden vastgesteld van wie het bericht afkomstig is?

Web Services Enhancements (WSE) is een Microsoft-toolkit die is gebaseerd op de open WS*-standaarden om bovengenoemde problemen eenvoudig op te lossen. Binnenkort zal alweer de derde versie van deze toolkit beschikbaar komen. Deze versie is gebaseerd op het .NET Framework 2.0 en is goed geïntegreerd met Visual Studio 2005. Momenteel is een CTP-versie (*Community Technology Preview*) van dit product beschikbaar op de Microsoft website [1]. De drie belangrijkste thema's in deze versie zijn eenvoud, beveiliging en toekomstzekerheid. De eenvoud komt tot uitdrukking in de goede integratie met Visual Studio 2005 en het .NET Framework 2.0. Daarnaast is het gebruik van beveiliging sterk vereenvoudigd. Zeker voor de toekomst betekent dat het product al is afgestemd op Indigo (de toekomstige web services-infrastructuur binnen Longhorn). Deze versie van de toolkit bevat reeds Indigo-concepten en is op berichtenniveau uitwisselbaar met Indigo. Dit betekent dat WSE 3.0 vandaag de dag dé manier is om webservices te beveiligen op het .NET Framework 2.0 en tevens dé manier is om je voor te bereiden op Indigo. In dit artikel richten we ons op de thema's eenvoud en beveiliging. We zullen laten zien hoe WSE 3.0 gebruikt kan worden om een oplossing te bieden voor een aantal veelvoorkomende beveiligingsrisico's.

```
[WebService(Namespace = "http://www.mybank.com/services/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Banking : System.Web.Services.WebService
{
  //constructor
  public Banking ()
  {
  }

  [WebMethod]
  public string TransferMoney(string debitAccount,
    string creditAccount, double amount)
  {
    return (string.Format("{0} dollars have been transferred
        from account '{1}' to account '{2}' on {3}",
      amount, debitAccount, creditAccount,
      DateTime.Now.ToString()));
  }
}
```

Codevoorbeeld 1. Een eenvoudige webservice van een bank voor het overmaken van geld

Configureren in plaats van programmeren

Bij de introductie van WSE 2.0 is er in .NET Magazine #4 aandacht besteed aan de basisprincipes voor het beveiligen van webservices [2]. In het genoemde artikel wordt versleuteling gerealiseerd door een oplossing te programmeren met behulp van testcertificaten. Dit artikel zal laten zien dat met WSE 3.0 programmeren kan worden vervangen door configureren. Tevens laten we zien hoe je in de praktijk op basis van echte X509-certificaten een veilige oplossing kan bouwen met WSE 3.0 en het .NET Framework 2.0. Om duidelijk te kunnen aangeven welke stappen je moet nemen, werken we een volledig voorbeeld uit. In dit voorbeeld gaan we uit van een server van een fictieve bank die een webservice aanbiedt voor het doen van financiële transacties. We beginnen met het ontwikkelen van de webservice in Visual Studio 2005. In codevoorbeeld 1 zie je een eenvoudige webservice van een bank voor het overmaken van geld van de ene rekening naar een andere.

Vanuit onze client voegen we vervolgens een webreferentie naar onze webservice toe. Hierdoor wordt er automatisch een webservice proxy gegenereerd. Omdat we op dit moment nog geen beveiligingsbeleid hebben afgedwongen, kunnen we onze webservice eenvoudig aanroepen door gebruik te maken van de gegenereerde webservice proxy; zie codevoorbeeld 2.

Hiermee hebben we de basis van onze oplossing gedefinieerd zonder enige vorm van beveiliging. We willen nu de volgende additionele eisen invullen:

- de identiteit van de server moet gewaarborgd zijn (server-authenticatie)

```
// construct the proxy
Banking bankingProxy = new Banking();

// invoke the web service method
MessageBox.Show( bankingProxy.TransferMoney("account A", "account B", 100));
```

Codevoorbeeld 2. Een gegenereerde webservice proxy

Afbeelding 1. De gebruikte infrastructuurcomponenten

- de identiteit van de gebruiker moet gewaarborgd zijn (client-authenticatie)
- informatie die we versturen moet alleen leesbaar zijn voor de bank (encryptie)
- de inhoud van het bericht moet niet kunnen worden gewijzigd door derden (*non-tampering*)
- de identiteit van de afzender van een bericht moet onomstotelijk bewijsbaar zijn (onweerlegbaarheid of *non-repudiation)*

Om dit te kunnen realiseren moeten we twee stappen nemen. Eerst richten we een omgeving in voor het gebruik van certificaten. Vervolgens definiëren we een beveiligingsbeleid dat we toepassen op onze bestaande code. De volgende paragraaf beschrijft het inrichten van de omgeving en geeft een korte introductie in het gebruik van certificaten in een PKI-omgeving. Wanneer je al bekend bent met PKI-technologie kan je deze paragraaf overslaan en direct doorgaan met de tweede stap in paragraaf Beveiliging configureren met WSE 3.0.

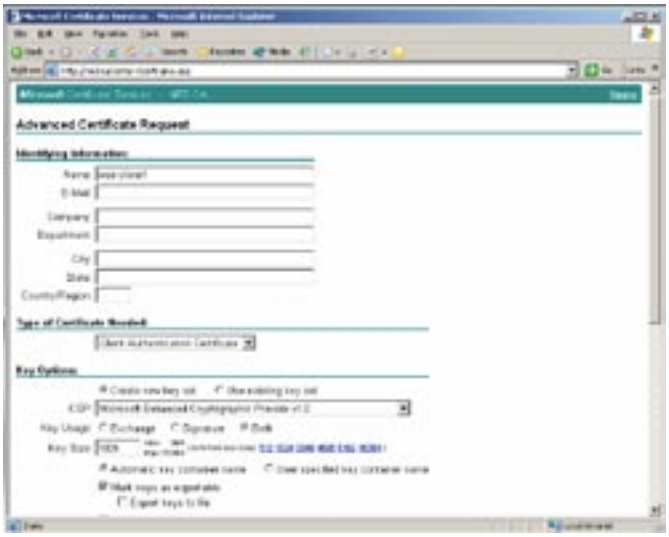
INRICHTEN OMGEVING VOOR X509-CERTIFICATEN

Om aan de eisen te kunnen voldoen maakt de oplossing in ons voorbeeld gebruik van een Public Key Infrastructure (PKI). Een PKI is een systeem van Certificate Authorities (CA's) waarmee public key-certificaten kunnen worden uitgegeven en beheerd. Een public key-certificaat is een digitaal ondertekende verklaring waarmee een publieke sleutel wordt gekoppeld aan de identiteit van de persoon of service met een bijbehorende persoonlijke sleutel. Om een PKI flexibel in te zetten plaatsen we de CA's in een hiërarchie. Hoe hoger een CA in de hiërarchie is geplaatst, hoe meer vertrouwen er is. De CA bovenin een PKI-hiërarchie wordt de 'root-CA' genoemd. De root-CA moet als zogenaamde 'Trusted Root' worden geregistreerd op alle systemen die certificaten van de PKI willen gebruiken. In een praktijksituatie kan een eigen PKI worden gebruikt of die van een commerciële partij (zoals Verisign). In het voorbeeld gebruiken we een PKI in eigen beheer op basis van de standaardvoorzieningen in Windows Server 2003, zodat we in detail kunnen laten zien hoe alles werkt. De gebruikte infrastructuurcomponenten zijn weergegeven in afbeelding 1. Zoals is te zien, beschikt de client over een certificaat (1) met de bijbehorende private sleutel. Ook de server beschikt over een certificaat (2) met bijbehorende private sleutel. Daarnaast beschikken zowel client als server over elkaars certificaat met publieke sleutel (maar uiteraard zonder private sleutel!). Om dit te bewerkstelligen moeten we op beide systemen de volgende stappen doorlopen:

1. Aanvragen van een certificaat bij de CA
2. Registreren van het certificaat in de local certificate store
3. Registreren van het rootcertificaat van de CA als 'trusted root'

Certificaat aanvragen

We beginnen met het aanvragen van een clientcertificaat. Dit is eenvoudig te doen door via de browser een aanvraag te plaatsen op de client. We typen het webadres in van de certificate server



Afbeelding 2. Het aanvragen van een clientcertificaat

(in ons geval http://nrd-ca/certsrv) en volgen de aanwijzingen op het scherm. We kiezen *wse-client1* als naam van de aanvrager; zie afbeelding 2.

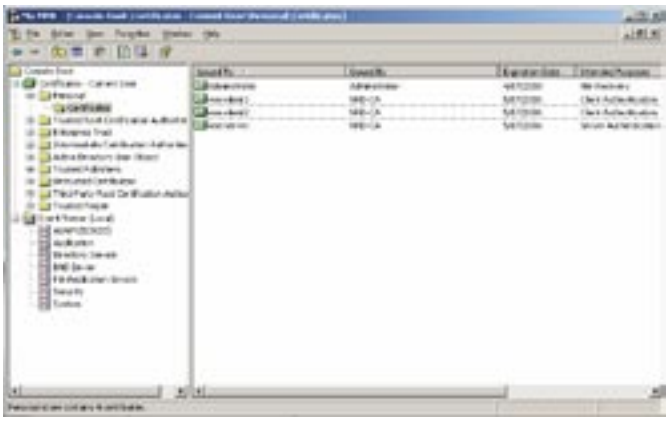
Hiermee wordt een certificaat aangevraagd bij onze CA. Op de CA accepteren we de aanvraag en creëren het certificaat (*MMC/Certification Authority/NRD-CA/Pending Requests/Issue*). Vanaf de client browsen we opnieuw naar de CA-webserver en zien we dat het aangevraagde certificaat beschikbaar is.

Certificaat registeren

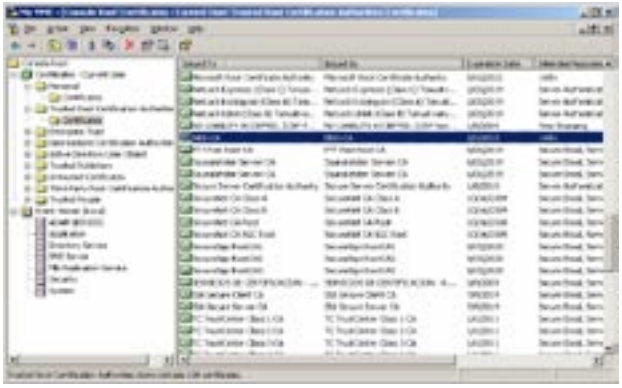
Nu kunnen we het certificaat registreren op de client. Dit kan eenvoudig worden gedaan door het certificaat te selecteren via de browser en de optie *Install this certificate* te kiezen. Hiermee wordt het certificaat geregistreerd in de persoonlijke certificate-store van de huidige gebruiker (*MMC/Certificates – Current User/Personal/Certificates,* zie afbeelding 3).

Vertrouwen van certificaten

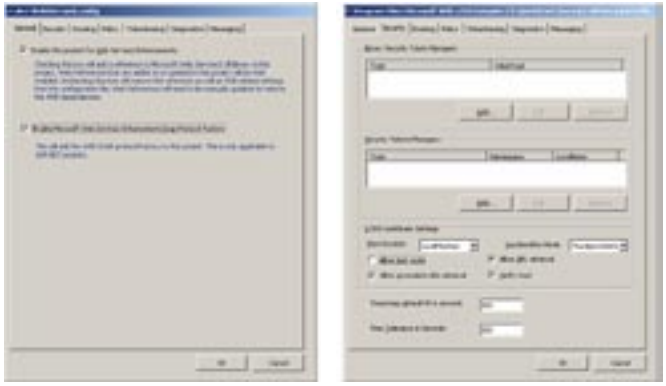
De authenticiteit van een identiteit is gebaseerd op vertrouwen. Dit betekent dat we met zekerheid de identiteit kunnen vaststellen wanneer we (1) kunnen bewijzen dat het certificaat is uitgegeven door een bepaalde partij en (2) we deze partij vertrouwen. Om het beheer van vertrouwen te vereenvoudigen is het mogelijk om een complete hiërarchie te bouwen (trust hierarchy). Hiermee ontstaat een ouder-kind-relatie tussen certificaten (het zogenaamde *certification path)*. Een certificaat is dan ook alleen geldig wanneer het complete pad van onder tot boven uit geldige vertrouwde certificaten bestaat die niet zijn ingetrokken (zie volgende paragraaf). In ons voorbeeld bestaat dit pad slechts uit twee certificaten (*wse-client1 -> nrd-ca*). Het pad zal moeten eindigen in een certificaat



Afbeelding 3. De registratie van het certificaat



Afbeelding 4. De registratie van het nrd-ca-certificaat als een vertrouwde CA



Afbeelding 6. Het kiezen van de juiste instellingen voor de webservice

dat afkomstig is van een partij die we vertrouwen. In ons geval betekent dat, dat we het *nrd-ca*-certificaat als een vertrouwde CA moeten registreren (zie *MMC/Certificates-Current User/Trusted Root Certification authorities/Certificates;* zie afbeelding 4. Dit certificaat is beschikbaar op de CA.

Intrekken van certificaten

Een certificaat is iets dat je aan iemand geeft om zijn identiteit te kunnen bewijzen. Wanneer een certificaat eenmaal is uitgegeven, is het lastig dit zomaar weer in te nemen. Zolang je het certificaat in je bezit hebt, kan je het gebruiken zo vaak en zolang je wilt. In sommige situaties is het echter toch wenselijk een certificaat van iemand ongeldig te kunnen verklaren. Dit wordt intrekking (of *revocation*) genoemd. Een certificaat kan worden ingetrokken door de CA. Dit is eenvoudig te doen door op de CA het uitgegeven certificaat te selecteren en de actie *Revoke Certificate* te kiezen. Hiermee komt het certificaat op een *blacklist* te staan (de zogenaamde *Certificate Revocation List (CRL)*). Deze lijst wordt op gezette tijden gepubliceerd door de CA en kan door iedere client worden gedownload via HTTP. Het http-adres is te vinden als eigenschap op het certificaat (attribuut *CRL-distribution points*). Om vast te stellen of een certificaat wel of niet is ingetrokken, kan je Windows laten controleren of het certificaat voorkomt op de lijst. Hiervoor is het dus nodig dat er (http-)toegang is tot de CA.

Hiermee zijn de belangrijkste stappen genomen om een certificaat te kunnen gebruiken op de client. Vervolgens doorlopen we dezelfde stappen op de server, maar met de volgende kleine verschillen:

- het certificaat dat we aanvragen is bedoeld voor serverauthenticatie in plaats van voor clientauthenticatie

Afbeelding 5. Leesrechten geven tot de private sleutel

- we registeren het certificaat niet in een specifieke user-store maar in de local machine store, omdat het certificaat toegankelijk moet zijn voor onze service.

Leestoegang geven tot de private sleutel

Op de server moet nog één additionele stap worden doorlopen: de application-pool waarin de webservice draait moet toegang krijgen tot de private key van het servercertificaat. Dit kan met behulp van de *WseCertificate3.exe*-tool; zie afbeelding 5. Deze is te vinden in de WSE-toolkit onder *..\Program Files\Microsoft WSE\Tools*. Met de tool selecteer je het servercertificaat (*CN=wse-server)* en klik je op *View Private Key File Properties* om leesrechten te geven aan het account waaronder de application-pool draait.

BEVEILIGING CONFIGUREREN MET WSE 3.0

In de vorige paragraaf zijn alle stappen besproken die nodig zijn om X509-certificaten te gebruiken voor de communicatie tussen client en server. In deze paragraaf zullen we de certificaten daadwerkelijk gaan gebruiken om berichten tussen client en server te versleutelen en te ondertekenen. We gaan terug naar ons voorbeeld (codevoorbeeld 1) en zullen dit scenario nu gaan beveiligen met de geregistreerde X509-certificaten. We beginnen met de webservice-code op onze server. In Visual Studio klikken we rechts op ons project in de *Solution Explorer* en kiezen de actie '*WSE Settings 3.0...*'. Op het eerste tabblad (zie afbeelding 6) selecteren we beide opties en vervolgens kiezen we het tabblad *Security*. Onder het kopje '*X.509 Certificate Settings*' kiezen we de juiste instellingen voor onze webservice; zie afbeelding 6.

Een uitleg van de belangrijkste instellingen worden hieronder weergegeven:

- **Store location –** de inkomende webservice-request is versleuteld met de public key van het servercertificaat. Bij binnenkomst van het bericht op onze server zal de WSE-runtime het juiste certificaat met de bijbehorende private sleutel zoeken om het bericht te kunnen ontsleutelen. De *store location* geeft aan in welke certificate-store wordt gezocht (mogelijke waarden *Current User* of *LocalMachine*). Op de server kiezen we hier de optie *LocalMachine*, omdat het servercertificaat in de *LocalMachine* store is opgeslagen.
- **KeyIdentifier mode –** bepaalt het algoritme dat gebruikt wordt om een certificaat uniek te identificeren (mogelijke waarden zijn *ThumbPrintSHA1, Capi* en *RFC3280*).
- **Allow test roots –** omdat we met certificaten van een echte CA werken laten we deze optie uit. Deze optie wordt alleen gebruikt in een testomgeving als certificaten worden gebruikt die zijn gegenereerd met behulp van bijvoorbeeld SELFCERT.EXE.
- **Allow URL retrieval –** hiermee wordt bepaald of de WSE-runtime dynamisch de juiste certificaten ophaalt van de CA om een volledig trust-pad van certificaten te kunnen opbouwen.
- **Allow revocation URL retrieval –** hiermee wordt bepaald of de WSE-runtime dynamisch een CRL zal ophalen van de CA

wanneer er geen (geldige) CRL lokaal aanwezig is.
- **Verify trust –** hiermee wordt bepaald of de WSE-runtime bij het controleren van een digitale ondertekening een volledig trust-pad van certificaten opbouwt en de geldigheid van alle betrokken certificaten controleert.

We herhalen deze stappen voor ons project op de client, met de volgende twee verschillen:
- In afbeelding 6 kiezen we alleen de bovenste optie.
- Bij de keuzemogelijkheid voor *StoreLocation* kiezen we nu *CurrentUser* store in plaats van *LocalMachine*.

Configureer de tokens en het gewenste beschermingsniveau

Nu de algemene beveiligingsinstellingen zijn gedefinieerd, kan het berichtenverkeer worden versleuteld en ondertekend. In het eerdergenoemde artikel over WSE 2.0 werd dit gedaan door code te introduceren. Deze mogelijkheid is ook in WSE 3.0 nog aanwezig voor geavanceerdere scenario's. Het is echter gebleken dat de meeste toepassingen van WS-Security eigenlijk terug zijn te voeren op een zestal basisscenario's. Deze scenario's zijn eenvoudig te implementeren door gebruik te maken van één van de zes standaard security-assertions die de WSE 3.0-toolkit biedt:
- AnonymousOverX509
- KerberosSecurity
- MutualX509Security
- UserNameOverTransportSecurity
- UsernameOverX509Security
- X509MutualAuthenticationProfile

Tabel 1 bevat een beschrijving van de standaard assertions met hun typische toepassingsgebied.

Deze assertions (of beweringen) kunnen op dezelfde manier worden gebruikt als de *Integrity* en *Confidentiality* assertions die we kennen uit de WS-SecurityPolicy specificatie [3]. Dit betekent dat eerst in een policy-bestand het gewenste beveiligingsniveau wordt aangegeven en de runtime er vervolgens automatisch voor zorgt dat het gewenste niveau wordt gerealiseerd.
Wat in vergelijking met de WS-SecurityPolicy direct opvalt is dat deze assertions de beveiliging op een hoger niveau definiëren dan we gewend zijn. Hierdoor wordt de configuratie sterk vereenvoudigd. We zullen dit laten zien aan de hand van ons voorbeeld waarin we wederzijdse authenticatie willen realiseren met de client- en servercertificaten. Wanneer we dit scenario met WS-SecurityPolicy zouden realiseren, wordt het configuratiebestand door de mate van detail uiteindelijk zeer complex. Met de komst van WSE 3.0 is dit echter eenvoudig te configureren door gebruik te maken van een standaard *assertion* met de naam *mutualX509Security*. Dit wordt beschreven in de volgende paragrafen.

```
<policies>
  <extensions>
    <extension name="mutualX509Security"
           type="Microsoft.Web.Services3.Design.MutualX509Assertion,
           Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral,
           PublicKeyToken=31bf3856ad364e35" />
    <extension name="x509"
           type="Microsoft.Web.Services3.Design.X509TokenProvider,
           Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral,
           PublicKeyToken=31bf3856ad364e35" />
  </extensions>
  <policy name="MyPolicy">
    <mutualX509Security establishSecurityContext="false"
           renewExpiredSecurityContext="true"
           signatureConfirmation="true"
           protectionOrder="SignBeforeEncrypting"
           deriveKeys="true" actor="">
      <clientToken>
        <x509 storeLocation="CurrentUser" storeName="My"
          findValue="CN=wse-client1"
          findType="FindBySubjectDistinguishedName" />
      </clientToken>
      <serviceToken>
        <x509 storeLocation="CurrentUser" storeName="My"
          findValue="CN=wse-server"
          findType="FindBySubjectDistinguishedName" />
      </serviceToken>
      <protection>
        <request signatureOptions="IncludeAddressing,
          IncludeTimestamp, IncludeSoapBody" encryptBody="true" />
        <response signatureOptions="IncludeAddressing,
          IncludeTimestamp, IncludeSoapBody" encryptBody="true" />
      <fault signatureOptions="IncludeAddressing, IncludeTimestamp,
          IncludeSoapBody" encryptBody="false" />
      </protection>
    </mutualX509Security>
  </policy>
</policies>
```

Codevoorbeeld 3. Het gegenereerde WSE 3.0 policy-bestand

CONFIGUREER EEN BEVEILIGINGSBELEID

We beginnen met het configureren van de beveiliging van de client. In de *Solution Explorer* van onze client-applicatie selecteren we het project en klikken we rechts om de *WSE Settings 3.0* configuratietool te starten. Op het tabblad *Policy* geven we aan dat we een nieuwe policy willen definiëren met de naam *MyPolicy*; zie afbeelding 7. Met behulp van een wizard kunnen we in een aantal stappen het beleid configureren; zie figuur 8. Het resultaat van de wizard is een policy-configuratiebestand dat automatisch wordt toegevoegd aan het project. Dit bestand is weergegeven in codevoorbeeld 3.

Naam	Betekenis	Typisch toepassingsgebied
anonymousOverX509Security	De client wordt niet geauthenticeerd. De server wordt geauthenticeerd op basis van zijn X509-certificaat. Beveiliging op berichtniveau wordt gerealiseerd op basis van het X509-token van het servercertificaat.	Client- en webservice- applicatie die over het internet communiceren
kerberosSecurity	De client en server wordt geauthenticeerd op basis van een Kerberos-token.	Client- en webservice- applicatie die over het intranet communiceren
mutualX509Security	Zowel de client als de server worden geauthenticeerd op basis van een X509-certificaat. Beveiliging op berichtniveau wordt gerealiseerd op basis van het X509-token van het certificaat.	Applicatie die over het internet communiceert
usernameOverTransportSecurity	De client wordt geauthenticeerd op basis van een username en password. De server wordt geauthenticeerd door een server SSL-certificaat te gebruiken op transportniveau.	Applicatie die over het internet communiceert
usernameOverX509Security	De client wordt geauthenticeerd op basis van een username en password. De server wordt geauthenticeerd op basis van X509-certificaat op berichtniveau.	Applicatie die over het internet communiceert
x509MutualAuthenticationProfile	Gelijk aan mutualX509Security maar nu op basis van de WS-Security 1.1 specificatie.	Applicatie die over het internet communiceert

Tabel 1. Standaard assertions met hun typische toepassingsgebied.

Afbeelding 7. De definiëring van een nieuwe policy

In dit bestand kunnen 0 of meer policies worden gedefinieerd. Een policy bestaat uit een <*policy*>-element met een naam (attribuut *name*). Binnen een <policy> vinden we een *assertion*. In dit geval maken we gebruik van een standaard *assertion* met de naam *mutualX509Security*. Daarbinnen geven we aan wat het clientcertificaat (<*clientToken*>) is dat we willen gebruiken voor ondertekening en het servercertificaat (<*serviceToken*>) dat we gebruiken voor versleuteling. Vervolgens geven we met het <*protection*>-element aan welke onderdelen van de berichtenstroom tussen client en server ondertekend en/of versleuteld dienen te worden. Dit zelfde beleid dient ook te worden gebruikt op de server. Dit kunnen we op twee manieren doen. De eerste mogelijkheid is om de wizard opnieuw uit te voeren op ons serverproject. Daarbij kiezen we dan overeenkomstige opties zoals op het clientproject. Uiteraard met de uitzondering dat we nu kiezen voor het configureren van een server (*Secure a service application*) in plaats van een cliënt; zie afbeelding 8. Een andere manier is om direct het policy-bestand te kopiëren naar de server. In ons geval hoeven dan alleen de *storeLocation* te veranderen, omdat het servercertificaat op de server in de *machineStore* is opgeslagen in plaats van de *userStore*. *Hiermee is de configuratie van het beleid geregeld. Het laatste dat we nu nog moeten doen is het daadwerkelijk toepassen van het beleid. Dit wordt uitgelegd in de volgende paragraaf.*

Pas het beveiligingsbeleid toe

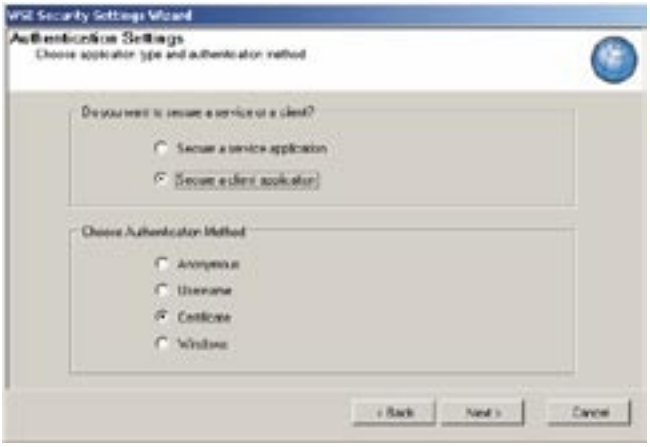
Zowel de client als server beschikken over een beveiligingsbeleid met de naam *MyPolicy*. Op zowel de client als de server moeten we nu aangeven dat we dit beleid willen gaan gebruiken. Op de client is hiervoor de WSE 3.0 proxy (*Microsoft.Web.Services3::WebServices-*

```
// construct the proxy
BankingWse bankingProxy = new BankingWse();

// set the policy
bankingProxy.SetPolicy("MyPolicy");

// invoke the web service method
MessageBox.Show( bankingProxy.TransferMoney("account A", "account B", 100));
```

Codevoorbeeld 4. Definieer welk beleid gebruikt dient te worden op de client



Afbeelding 8. WSE Security Settings Wizard: settings authenticatie

ClientProtocol) uitgebreid met een extra methode *SetPolicy*; zie afbeelding 9. Een van de overloads van de deze methode accepteert een string waarmee we de naam kunnen aangeven van het beleid dat we willen toepassen. Om het eerdergenoemde geconfigureerde beveiligingsbeleid toe te passen, geven we de naam aan van het beleid zoals weergegeven in codevoorbeeld 4. Omdat we nu WSE- functionaliteit gaan gebruiken instantiëren we een proxy van het type *BankingWSE* in plaats van *Banking*; zoals in codevoorbeeld 2. Deze nieuwe proxy is automatisch gegenereerd door de WSE-toolkit.

Als alternatief kunnen we ook gebruik maken van een *Policy*-attribuut dat we op de proxy-class kunnen toevoegen. Met behulp van de partial classes-syntax in het .NET Framework 2.0 kunnen we dus eenvoudig de volgende code toevoegen aan onze clientcode.

```
namespace wse_client2.Banking
{
   [Microsoft.Web.Services3.Policy("MyPolicy")]
   public partial class BankingWse
         :Microsoft.Web.Services3.WebServicesClientProtocol{};
}
```

Codevoorbeeld 5. Een alternatieve manier om op de client aan te geven welk beleid gebruikt dient te worden

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using Microsoft.Web.Services3;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[Policy("MyPolicy")]
public class Banking : System.Web.Services.WebService
{
   public Banking()
   {
   }

   [WebMethod]
   public string TransferMoney(string debitAccount,
      string creditAccount, double amount)
   {
      return (string.Format("{0} dollars have been transferred from
            account '{1}' to account '{2}' on {3}",
            amount, debitAccount, creditAccount, DateTime.Now.ToString()));
   }
}
```

Codevoorbeeld 6. Definieer welk beleid gebruikt dient te worden op de server

Afbeelding 9. De WSE 3.0 proxy is uitgebreid met de extra methode SetPolicy

De laatste stap die we moeten nemen is het toepassen van het beleid op de serverkant. Dit kunnen we doen door het *Policy*-attribuut toe te passen op onze webservice-class. Dit is weergegeven in codevoorbeeld 6.

Hiermee is de volledige beveiligingsconfiguratie afgerond. De berichten die we nu tussen client en server heen en weer sturen, worden automatisch versleuteld en ondertekend. Met de introductie van WSE 3.0 is het beveiligen van webservices dus sterk vereenvoudigd en is de hoeveelheid benodigde code tot een minimum beperkt.

Erik S.C. van de Ven is senior consultant bij Microsoft Services Nederland. Zijn e-mailadres is erikven@microsoft.com

Referenties

[1] WSE 3.0 CTP en hands-on labs: http://msdn.microsoft.com/webservices/building/wse/default.aspx

[2] .NET Magazine for developers #4, december 2003, Tooltime: Web Services Enhancements 2.0, Veilig werken met open protocollen, Mark Blomsma. http://www.microsoft.nl/netmagazine4

[3] WS-SecurityPolicy: http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-securitypolicy.asp#ws-securitypolicy_toc27450548

[4] WS-Security: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[5] Introduction to Certificate Services: http://www.microsoft.com/technet/itsolutions/wssra/raguide/CertificateServices/default.mspx