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

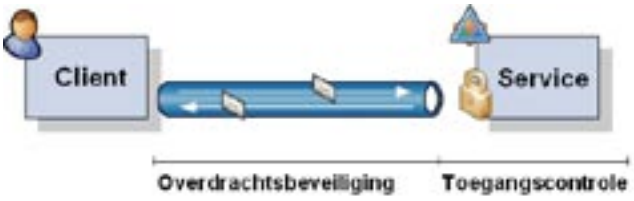
Services beveiligen met Windows Communication Foundation

CREDENTIALS VORMEN DE BASIS VAN BEVEILIGING

In een eerder artikel in het .NET Magazine [WSE] is al uitvoerig behandeld hoe we de beveiliging van webservices kunnen realiseren met Microsoft Web Services Enhancements (WSE) toolkit. Van deze toolkit is momenteel versie 3.0 beschikbaar. WSE 3.0 is de manier om webservices te beveiligen op basis van open standaarden (zoals WS-Security). Met de komst van Windows Communication Foundation (WCF) - als onderdeel van WinFX - ontstaat een compleet nieuwe servicecommunicatie-infrastructuur.

In een vorig artikel over WCF [PIM] is een algemeen overzicht gegeven van de mogelijkheden van deze nieuwe servicecommunicatie-infrastructuur. In dit artikel gaan we specifiek in op de beveiligingsmogelijkheden van WCF. Beveiliging bestaat uit overdrachtsbeveiliging en toegangscontrole. Het woord 'beveiliging' wordt vaak als een algemene term gebruikt. Met beveiliging bedoelen we een verzameling maatregelen die getroffen wordt om bepaalde bedreigingen tegen te gaan. Het is dan ook belangrijk om op voorhand aan te geven wat we precies verstaan onder beveiliging in de context van services. We delen het begrip beveiliging op in twee delen: overdrachtsbeveiliging en toegangscontrole. Met overdrachtsbeveiliging bedoelen we het nemen van maatregelen om bedreigingen tegen te gaan tijdens de overdracht van berichten tussen service-aanroeper (client) en service-aanbieder (service).

Tijdens de overdracht van een bericht kan er bijvoorbeeld de bedreiging zijn dat de inhoud van het bericht door derden wordt gewijzigd. Daarnaast is het belangrijk op te merken dat er in een servicegeoriënteerde oplossing niet altijd een directe koppeling bestaat tussen een client en service. Communicatie kan plaatsvinden tussen één of meer tussenpersonen (zogenaamde *intermediaries*).



Afbeelding 1. De onderdelen van beveiliging: overdrachtsbeveiliging en toegangscontrole

Afbeelding 2. Interfacedefinitie van de gebruikte service

In dit geval zullen we dus maatregelen moeten treffen die op het bericht zelf van toepassing zijn. Dit geldt zelfs wanneer het bericht over verschillende kanalen via verschillende tussenpersonen wordt afgeleverd op de uiteindelijke eindbestemming. Wanneer het bericht de eindbestemming bereikt, om vervolgens een bepaalde serviceoperatie uit te voeren, is het nog maar de vraag of de oorspronkelijke aanroeper hiertoe gerechtigd is. Eerst wordt gecontroleerd of de afzender daadwerkelijk diegene is wie hij zegt dat hij is (*authenticatie*). Vervolgens wordt gecontroleerd of de afzender de serviceoperatie mag uitvoeren. Deze toegangscontrole vormt het tweede onderdeel van beveiliging en wordt ook wel *access control* of *autorisatie* genoemd. Beide onderdelen van beveiliging zijn weergegeven in afbeelding 1.

Credentials vormen de basis van identiteit

De basis van beveiliging wordt gelegd door zogenaamde *credentials*. Een credential is iets dat iemand kan overhandigen aan een ander om zijn identiteit te bewijzen. Als voorbeeld kan ik mijn rijbewijs aan iemand overhandigen, zodat de ander kan controleren dat ik daadwerkelijk ben wie ik zeg dat ik ben. Een rijbewijs is dus een vorm van credential die mijn identiteit tegenover een ander kan bewijzen. Het rijbewijs bevat bepaalde echtheidskenmerken (foto, stempel, aluminiumfolie, et cetera) die de ander controleert om vast te stellen of het rijbewijs inderdaad is afgegeven door het *Koninkrijk der Nederlanden*. Vervolgens *vertrouwt* de ander erop dat dit rijbewijs met zorg is afgegeven aan de juiste persoon. Welke kenmerken hierbij gecontroleerd worden en wie verantwoordelijk is voor de afgifte van de credential bepalen dus eigenlijk de manier waarop authenticatie plaatsvindt. Deze verzameling van principes wordt dan ook gezamenlijk een *authenticatiemodel* genoemd. WCF biedt standaard ondersteuning voor drie bekende authenticatiemo-

Bedreiging	Maatregel
Integriteit (integrity)– de inhoud van het bericht is gewijzigd door onbevoegden	Ondertekening
Vertrouwelijkheid (confidentiality)– de inhoud van het bericht wordt gelezen door onbevoegden	Versleuteling
Onweerlegbaarheid (non-repudation)– de afzender ontkent het bericht te hebben verzonden	Ondertekening
Herhaling (replay)– het bericht wordt opgenomen en opnieuw verzonden door een onbevoegde	Ondertekening met hash vergelijking

Tabel 1. Beveiligingsbedreigingen met hun corresponderende maatregel

```
[ServiceContract()]
public interface IBankingServices
{
   [OperationContract]
   string TransferMoney(string debitAccount, string creditAccount,
      double amount);
}

public class BankingServices : IBankingServices
{
   #region IBankingServices Members

   string IBankingServices.TransferMoney(string debitAccount,
        string creditAccount, double amount)
   {
      return string.Format("{0} dollars have been transferred from
      account '{1}' to account '{2}'", amount, debitAccount,    creditAccount);
   }

   #endregion
}
```

Codevoorbeeld 1.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <client>
         <endpoint
            address="http://server/host2/BankingServices.svc"
            binding="basicHttpBinding"
            contract="IBankingServices">
         </endpoint>
      </client>
   </system.serviceModel>
</configuration>
```

Codevoorbeeld 2.

Stap 1 – Basisconfiguratie (client-configuratie)

```
<configuration>
   <system.diagnostics>
      <trace autoflush="true"/>
      ...
   </system.diagnostics>
   <system.serviceModel>
      <diagnostics>
         <messageLogging logEntireMessage="true"
            logMalformedMessages="false"
            logMessagesAtServiceLevel="false"
            logMessagesAtTransportLevel="true"
            maxMessagesToLog="1000"
            maxSizeOfMessageToLog="2147483647">
         </messageLogging>
      </diagnostics>
      <services>
         <service type="SecureServices.BankingServices">
            <endpoint
               address="http://server/host2/BankingServices.svc"
               binding="basicHttpBinding"
               contract="SecureServices.IBankingServices">
            </endpoint>
         </service>
      </services>
   </system.serviceModel>
</configuration>
```

Codevoorbeeld 3.

Stap 1 – Basisconfiguratie (serviceconfiguratie)

dellen: *UserName / Password*, *Certificate* en *Windows*. Het *UserName* credential-type is eenvoudigweg een combinatie van een gebruikersnaam met een wachtwoord. Vanwege de eenvoud (van beheer) is dit een veelgebruikt authenticatiemodel op het internet. In het *Certificate*-authenticatiemodel worden X509-certificaten gebruikt als credential die zijn uitgegeven door een *Certificate Authority*. In het *Windows*-authenticatiemodel wordt een Kerberos-ticket gebruikt als credential die is afgegeven door een Kerberos Authority (of *Key Distribution Center*). Naast ondersteuning voor deze basismodellen biedt WCF mogelijkheden voor geavanceerde identiteitmetasystemen (zoals *InfoCard*-gebaseerde oplossingen). Het reikt echter te ver om dit model ook in dit artikel te behandelen.

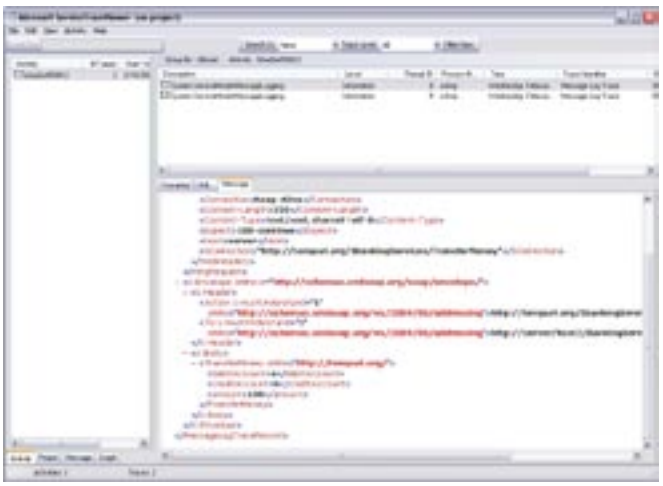
Het nut van overdrachtsbeveiliging

Overdrachtsbeveiliging gaat dus over het beveiligen van het berichtenverkeer tussen client en service. Tijdens overdracht van een bericht kan een aantal bedreigingen optreden. Deze bedreigingen met hun corresponderende maatregel zijn weergegeven in tabel 1

We zullen nu aan de hand van een voorbeeld laten zien hoe we maatregelen kunnen treffen om de bedreigingen uit bovengenoemde tabel het hoofd te bieden. We hebben een eenvoudige service geïmplementeerd met de naam *BankingServices*. Deze service implementeert de *IBankingServices*-interface met daarop één methode *TransferMoney*. Het klassendiagram en de code zijn te vinden in afbeelding 2 en codevoorbeeld 1.

Om de service te kunnen aanroepen zullen we deze beschikbaar maken in een host. We hosten onze service in Internet Information Services 6.0 en definiëren een endpoint op deze service, zodat we deze kunnen benaderen via het http-protocol. We gebruiken hiervoor een *basicHttpBinding* (1). De bijbehorende configuratie van de client en service is respectievelijk te vinden in codevoorbeeld 2 en 3 (2).

Om goed te begrijpen wat er daadwerkelijk gebeurt bij het aanroepen van de service gaan we kijken naar de exacte berichtenuitwisseling tussen client en service. Hiervoor gebruiken we de service trace-viewer. Deze tool is een onderdeel van de Windows SDK [SDK] en is standaard te vinden in '..*Program Files**Microsoft SDKs**Windows**v1.0**Bin*' onder de naam *SvcTraceViewer*. We configureren de service nu zodat alle binnenkomende en uitgaande berichten op de service getraced worden en zichtbaar worden in de trace-viewer. Dit gebeurt via het *system.ServiceModel / diagnostics*-element; zie codevoorbeeld 3 (3). Let op dat we alleen berichten op *transport*-niveau tracen, omdat we willen zien hoe het ruwe bericht eruit ziet tijdens transport voordat het is geïnterpreteerd door WCF. Om ervoor te zorgen dat alle traces direct naar het bestand worden geschreven, zetten we de waarde *system.diagnostics / trace @ autoflush* op *true*. Wanneer we nu de service aanroepen zien we een onbeveiligde berichtuitwisseling tussen de client en de service. Het *request*-bericht is weergegeven in afbeelding 3.
De binding die we hier gebruiken – basicHttpBinding- is de enige binding in WCF die standaard niet is beveiligd. Alle overige bindings in WCF ondersteunen standaard een of meer vormen van beveiliging. De basicHttpBinding is ontworpen om compatibel te zijn met de eerste generatie webservices (ASMX) conform de WS-I Basic Profile [BP] en WS-I Basic Security Profile. Omdat het berichtenverkeer niet beveiligd is, kunnen we een willekeurige http-sniffer gebruiken om de berichten op te nemen, te wijzigen en weer opnieuw af te spelen. Een handige tool hiervoor is Fiddler (http://www.fiddlertool.com/fiddler/). Met deze tool nemen we het SOAP-request op, wijzigen het en spelen het naar gelieven



Afbeelding 3. Inkomend bericht bij stap 1

Binding	Niveau	Maatregel	Server Credentials	Client Credentials
basicHttpBinding	Transport	Versleuteling	Certificaat	Geen

Tabel 2.

opnieuw af, zodat ons saldo gestaag groeit. Hiermee is dus duidelijk gemaakt dat bij dit soort gevoelige transacties het berichtenverkeer tussen client en server op een aantal punten beschermd dient te worden: integriteit, vertrouwelijkheid en herhaalbaarheid.

Deze binding biedt de mogelijkheid van transport-level-beveiliging via https. Het implementeren van transport-level-beveiliging is eenvoudig en vaak een adequaat middel om beveiliging te realiseren. Onder de volgende voorwaarden is dit dan ook prima te gebruiken:

1) de berichten dienen alleen beveiligd te worden tijdens transport. Immers, wanneer het bericht uit het SSL-kanaal komt, is de beveiliging verdwenen.
2) er is sprake van een directe point-to-point-verbinding tussen client en service.

In de volgende paragraaf zullen we laten zien hoe we deze beveiliging configureren met WCF.

Overdrachtsbeveiliging configureren met bindings en behaviors

Beveiliging op transportniveau met https

We laten nu zien welke stappen er genomen moeten worden om vertrouwelijkheid te realiseren op basis van https. In hoofdlijnen moeten we twee stappen nemen. In de eerste stap configureren we een servercertificaat dat we willen gebruiken in IIS ten behoeve van serverauthenticatie. En als tweede stap zullen we in de WCF-configuratie van zowel de client als de service moeten aangeven dat we gebruiken willen maken van https in plaats van http.
De eerste stap is eigenlijk niets bijzonders aangezien dit de standaard manier is waarop we een server side SSL-kanaal configureren in IIS. Via de CA hebben we een certificaat verkregen dat geschikt is voor serverauthenticatie. Vervolgens gebruiken we de *Web Server Certificate Wizard* om het certificaat aan onze webserver te koppelen. Deze wizard is te bereiken via MMC / Internet Information Services (IIS) Manager / Web Sites / <naam web site> / Properties / Directory Security / Server Certificate.
In de tweede stap zullen we het adres in het endpoint van onze service moeten vervangen door een https-adres [4]. Het realiseren van beveiliging binnen WCF gebeurt volledig op basis van bindings en behaviors, dit is dus volledig onafhankelijk van het programmeermodel (services, contracten enzovoort.). Hierdoor kunnen we zonder ook maar een enkele codewijziging te maken een uitgebreide set van beveiligingsmaatregelen treffen. In WCF is de beveiliging geabstraheerd tot twee kernbegrippen: enerzijds het definiëren van de maatregel die getroffen dient te worden (ondertekening of versleuteling) en anderzijds welke credentials daarvoor gebruikt dienen te worden. Deze instellingen in combinatie met de gekozen binding resulteren automatisch in een beveiligde berichtenstroom tussen client en service. De gemaakte keuzes zijn weergegeven in tabel 2.
Om https te kunnen gebruiken zullen we ook in de binding moeten aangeven dat we gebruik willen maken van transport-level-beveiliging. Hiervoor wordt het *mode*-attribuut gebruikt dat onderdeel is van het *security*-element; zie codevoorbeeld 4 en 5. De belangrijkste mogelijke waarden van dit attribuut worden weergegeven in tabel 3. Wanneer je deze stap achterwege laat krijg je een

Security @ Mode attribuut waarde voor basicHttpBinding	Betekenis
None	Berichten zijn niet beveiligd tijdens overdracht
Transport	Beveiliging wordt geboden door https. Bericht worden versleuteld verstuurd door middel van het SSL-kanaal. De service dient geconfigureerd te worden met een server side SSL-certificaat. Serverauthenticatie vindt plaats middel dit certificaat. De authenticatie van de client wordt bepaald door het *ClientCredentialType*-attribuut van het *security/transport*-element.
Message	Beveiliging wordt geboden op SOAP-berichtniveau. Standaard worden berichten versleuteld en ondertekend.
TransportWith MessageCredential	Integriteit, vertrouwelijkheid en serverauthenticatie worden geboden door https met een server side-certificaat. Client-authenticatie wordt vervolgens geboden door op bericht niveau te ondertekenen. Deze combinatie van transport en berichtbeveiliging is met name relevant om username credentials over een bestaande https-kanaal te kunnen sturen.

Tabel 3.

```
<services>
  <service type="SecureServices.BankingServices">
    <endpoint
      address="https://server/host2/BankingServices.svc"
      binding="basicHttpBinding"
      bindingConfiguration="secured"
      contract="SecureServices.IBankingServices">
    </endpoint>
  </service>
</services>
<bindings>
  <basicHttpBinding>
    <binding name="secured">
      <security mode="Transport">
        <transport clientCredentialType="None"/>
      </security>
    </binding>
  </basicHttpBinding>
</bindings>
```

Codevoorbeeld 4.

Stap 2 – Transport niveaubeveiliging met serverauthenticatie (serviceconfiguratie)

```
<system.serviceModel>
  <client>
    <endpoint
      address="https://server/host2/BankingServices.svc"
      binding="basicHttpBinding"
      bindingConfiguration="secured"
      contract="IBankingServices">
    </endpoint>
  </client>
  <bindings>
    <basicHttpBinding>
      <binding name="secured">
        <security mode="Transport">
          <transport clientCredentialType="None"/>
        </security>
      </binding>
    </basicHttpBinding>
  </bindings>
</system.serviceModel>
```

Codevoorbeeld 5.

Stap 2 – Transport niveaubeveiliging met serverauthenticatie (client-configuratie)

foutmelding van de WCF-runtime die aangeeft dat https-verkeer niet is toegestaan. Omdat we in dit geval alleen server side-authenticatie willen gebruiken en dus geen client-credentials meegeven, zetten we de waarde van het attribuut *transport @ clientCredentialType* op *None*. Dit attribuut wordt in meer detail besproken in de volgende paragraaf.

Wanneer we nu onze service opnieuw aanroepen en met een geavanceerde tool als Netmon kijken naar het verkeer tussen client en server, dan zien we alleen nog maar SSL-versleuteld http-verkeer tussen beide. Aan de servicekant wordt het verkeer dat uit het kanaal komt vervolgens onversleuteld doorgegeven aan de WCF-runtime. Afbeelding 4 toont het bericht dat op transportniveau binnenkomt op de service. Hieraan is duidelijk te zien dat het bericht zijn versleuteling verliest zodra het uit het SSL-kanaal naar buiten komt. Dit is inherent aan het gebruik van transportniveaubeveiliging. Wanneer je de versleuteling wilt behouden, zal je dus maatregelen moeten nemen op het bericht zelf, in plaats van op het transportkanaal. In de volgende paragraaf laten we zien hoe we op berichtniveau beveiliging kunnen realiseren.

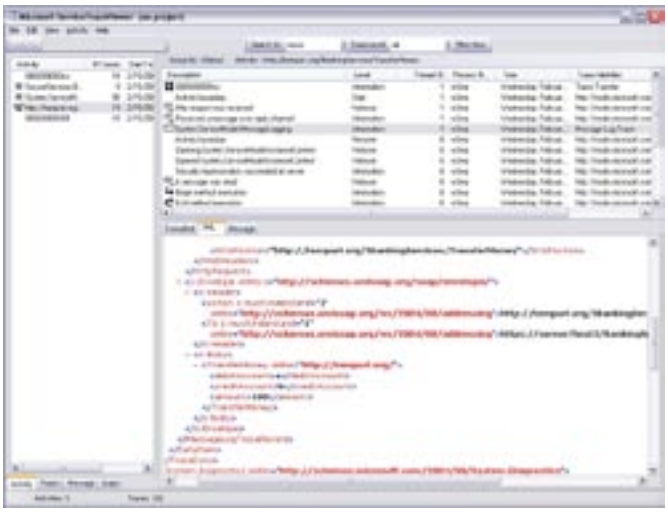
Beveiliging op berichtniveau met WS-Security

Zoals we in de vorige paragraaf hebben gezien, biedt transportbeveiliging een eenvoudige en doeltreffende manier voor het beschermen van het berichtenverkeer. In sommige gevallen (zie vorige paragraaf) biedt transportniveaubeveiliging echter onvoldoende bescherming en zullen we een stap hoger moeten grijpen door gebruik te gaan maken van berichtniveaubeveiliging. De open standaard die hiervoor ontwikkeld is heet WS-Security [WSS]. WS-Security is een open standaard om individuele berichten te versleutelen en te ondertekenen. We gebruiken nu de *wsHttpBinding* in plaats van de *basicHttpBinding*.

```
<services>
  <service type="SecureServices.BankingServices"
      behaviorConfiguration="credentials">
    <endpoint
      address="http://server/host2/BankingServices.svc"
      binding="wsHttpBinding"
      bindingConfiguration="secured"
      contract="SecureServices.IBankingServices">
    </endpoint>
  </service>
</services>
<bindings>
  <wsHttpBinding>
    <binding name="secured">
      <security mode="Message">
        <message
          clientCredentialType="Certificate"
          defaultProtectionLevel="Sign"
          negotiateServiceCredential="false"/>
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
<behaviors>
  <behavior name="credentials">
    <serviceCredentials>
      <serviceCertificate
        findValue="server"
        storeLocation="LocalMachine"
        storeName="My"
        x509FindType="FindBySubjectName"/>
    </serviceCredentials>
  </behavior>
</behaviors>
```

Codevoorbeeld 6.

Stap 3 – Berichtniveaubeveiliging met ondertekening (serviceconfiguratie)



Afbeelding 4. Inkomend bericht bij stap 2

Binding	Niveau	Maatregel	Server Credentials	Client Credentials
wsHttpBinding	Bericht	Ondertekening met wederzijdse authenticatie	Certificaat	Certificaat

Tabel 4

Deze binding biedt de meeste brede ondersteuning voor de WS-* standaarden zoals *WS-SecureConversation*.
We beginnen met een voorbeeld waarin we ondertekening van berichten gebruiken om te waarborgen dat berichten niet kunnen worden gewijzigd (*integriteit* of *non-tampering*) en de afzender niet kan ontkennen dat hij het bericht heeft verzonden (*onweerlegbaarheid* of *non-repudiation*). Als client credential-type gebruiken we een certificaat. De genomen beslissingen zijn weergegeven in tabel 4.

Authenticatie van de server vindt plaats door de het servercertificaat te valideren en te controleren of de *subject-name* van het certificaat overeenkomt met de DNS-naam van de server waarmee we een verbinding maken (in dit geval is de naam 'server'). Client-authenticatie vindt plaats op de server waar wordt gecontroleerd of het certificaat afkomstig is van een vertrouwde CA en of het ondertussen niet is ingetrokken door de CA. Ook wordt de handtekening op het bericht gecontroleerd om te valideren of het bericht niet is gewijzigd tijdens transport. Als deze controles worden automatisch uitgevoerd door de WCF-runtime door eenvoudig de volgende instellingen te configureren. Op de server geven we aan dat we een *server*certificaat beschikbaar hebben die de identiteit van de server bevat. Op de client geven we aan wat het clientcertificaat is en wat het servercertificaat is. Omdat we dit certificaat op de client beschikbaar hebben, willen we verder geen onderhandeling over het te gebruiken servicecertificaat (*negotiateServiceCredential* = "false"). We kiezen in dit geval alleen voor ondertekening van de berichten en zetten dus de *defaultProtectionLevel* op *Sign* (mogelijke waarden zijn *None, Sign* en *EncryptAndSign*). De resulterende configuratie is weergegeven in codevoorbeeld 6 en 7.
Als laatste stap moeten we er nog voor zorgen dat de service toegang heeft tot de private key van het servercertificaat. Zoals wellicht bekend worden private keys van certificaten in Windows bewaard in een bestand. De toegang tot deze sleutels wordt afgedwongen via standaard Windows ACL's. We moeten dus zorgen dat het account, dat gebruikt wordt om de service te hosten, toegang heeft tot deze private sleutel. In de Windows SDK [SDK] kan je de tool *FindPrivateKey*

gebruiken om de private sleutel te vinden die bij het servercertificaat hoort. Met het volgende commando wordt de naam van het bestand en de folder weergeven die de bijbehorende private sleutel bevat:

```
findprivatekey.exe My LocalMachine -n "CN=Server, OU=test, O=MS,
L=Epe, S=Gelderland, C=nl"
```

Omdat in dit geval het IIS-workerproces onder de identiteit van NETWORK Service draait, geven we dat account leesrechten tot het gevonden bestand.
Wanneer we nu de service aanroepen dan zien we inderdaad dat er een bericht wordt verstuurd dat ondertekend is conform de WS-Security-standaard; zie afbeelding 6.
Standaard wordt ook gecontroleerd door de service of het certificaat van de client ondertussen niet is ingetrokken door de CA. Deze controle wordt gedaan door een lijst met ingetrokken certificaten (*Certificate Revocation List (CRL)*) op te vragen bij de CA. Of deze lijst ook daadwerkelijk opgehaald dient te worden of niet, kan je beïnvloeden met het configuratieattribuut *clientCertificate / revocationMode*. Dit attribuut kent de volgende waarden *NoCheck, Offline* en *Online*. Bij *Online* wordt contact gezocht met de CA om een recente versie van de CRL op te halen wanneer de huidige versie is verlopen, terwijl bij *Offline* eenvoudigweg de laatste locale versie wordt gebruikt. Willen we naast ondertekening ook nog versleuteling van de berichten, dan kunnen we dat eenvoudig configureren door zowel in de client- als de serviceconfiguratie het attribuut *message @ defaultProtectionLevel* te wijzigen in de waarde *EncryptAndSign* in plaats van *Sign*. Hiermee hebben we dus gezien dat we zowel op transport- als berichtniveau eenvoudig kunnen configureren welke beveiligingsmaatregelen we willen treffen. De configuratiekeuze van het beschermingsniveau (ondertekening eventueel in combinatie met versleuteling) en de binding leidt automatisch tot de juiste berichtenstroom tussen client en service. In de volgende stap zullen we laten zien hoe we op basis van het binnenkomende bericht kunnen bepalen of een gebruiker überhaupt gerechtigd is om een serviceoperatie uit te voeren.

Toegang tot services regelen

In de vorige paragraaf hebben we gezien hoe we het berichtenverkeer tussen een client en service kunnen beveiligen met ondertekening en versleutelen op basis van credentials. In deze paragraaf zullen we laten zien hoe we de credentials van de client kunnen gebruiken om te bepalen of een aanroeper wel gerechtigd (geautoriseerd) is om een service aan te roepen.
De *ServiceSecurityContext* bevat de identiteit van de aanroeper
Zoals we al eerder hebben besproken, ondersteunt WCF verschillende credential-typen (Windows, certificaten en gebruikersnaam). Al deze credential-typen kunnen worden gebruikt om de identiteit van de gebruiker te bepalen om daarmee vervolgens de autorisatie

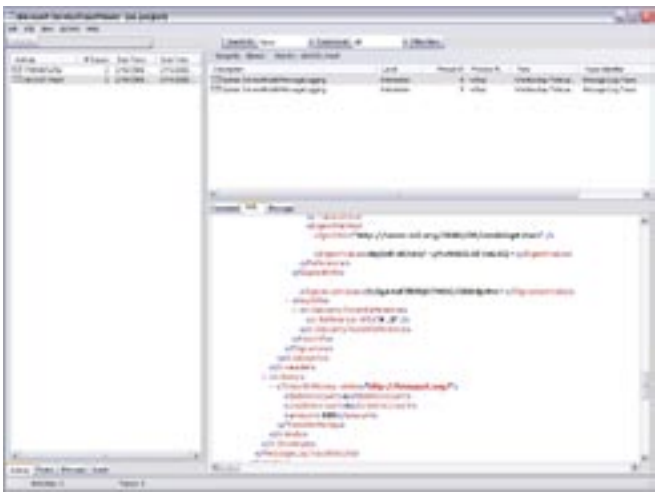
```
<system.serviceModel>
<client>
  <endpoint
    address="http://server/host2/BankingServices.svc"
    binding="wsHttpBinding"
    bindingConfiguration="secured"
    behaviorConfiguration="credentials"
    contract="IBankingServices">
  </endpoint>
</client>
<bindings>
  <wsHttpBinding>
    <binding name="secured">
      <security mode="Message">
        <message
          clientCredentialType="Certificate"
          defaultProtectionLevel="Sign"
          negotiateServiceCredential="false"/>
      </security>
    </binding>
    </wsHttpBinding>
</bindings>
<behaviors>
  <behavior name="credentials">
    <clientCredentials>
      <clientCertificate findValue="john"
                storeLocation="CurrentUser"
                storeName="My"
                x509FindType="FindBySubjectName"/>
      <serviceCertificate findValue="server"
                storeLocation="LocalMachine"
                storeName="My"
                x509FindType="FindBySubjectName"/>
    </clientCredentials>
  </behavior>
</behaviors>
</system.serviceModel>
```

Codevoorbeeld 7.

Stap 3 – Berichtniveaubeveiliging met ondertekening (clientconfiguratie)

te regelen. Naast custom-oplossingen integreert WCF uiteraard ook met het standaard .NET security-mechanisme (*PrincipalPermission)* en het rollenbeheer zoals we dat kennen in *Active Directory*.
We beginnen opnieuw met een eenvoudig voorbeeld dat gebaseerd is op een *Username*-credential. In dit scenario gebruiken we een combinatie van transport en berichtniveaubeveiliging. We maken een beveiligd kanaal op basis van een server side SSL-verbinding waarin het verkeer wordt versleuteld. Dit kanaal biedt bescherming ten behoeve van vertrouwelijkheid. Daarnaast willen we op

Afbeelding 5. Leestoegang geven tot de private sleutel



Afbeelding 6. Inkomend bericht stap 3

Binding	Niveau	Maatregel	Server Credentials	Client Credentials
wsHttp-Binding	Transport en Bericht	Ondertekening tbv integriteit en versleuteling op transport- en ondertekening tbv authenticatie op berichtniveau	Certificaat	Username

Tabel 5.

```
proxy = new BankingServicesProxy();

proxy.ClientCredentials.UserNamePassword.UserName = @"SECURE\john";
proxy.ClientCredentials.UserNamePassword.Password = "pass@word1";
```

Codevoorbeeld 8.

```
<system.serviceModel>
  <client>
    <endpoint
      address="https://server/host2/BankingServices.svc"
      binding="wsHttpBinding"
      bindingConfiguration="secured"
      contract="IBankingServices">
    </endpoint>
  </client>
  <bindings>
    <wsHttpBinding>
      <binding name="secured">
        <security mode="TransportWithMessageCredential">
          <message clientCredentialType="UserName"/>
        </security>
      </binding>
    </wsHttpBinding>
  </bindings>
</system.serviceModel>
```

Codevoorbeeld 9.

Stap 4 – Transportniveauversleuteling gecombineerd met berichtniveau-

```
<services>
  <service type="SecureServices.BankingServices">
    <endpoint
      address="https://server/host2/BankingServices.svc"
      binding="wsHttpBinding"
      bindingConfiguration="secured"
      contract="SecureServices.IBankingServices">
    </endpoint>
  </service>
</services>
<bindings>
  <wsHttpBinding>
    <binding name="secured">
      <security mode="TransportWithMessageCredential">
        <message clientCredentialType="UserName" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

Codevoorbeeld 10.

Stap 4 – Transportniveauversleuteling gecombineerd met berichtni

berichtniveau de afzender kunnen vaststellen en voorkomen dat de inhoud van het bericht kan worden gewijzigd. Hiervoor gebruiken we ondertekening op berichtniveau.

Hier zien we dus een combinatie van maatregelen op transport- en berichtniveau. Deze combinatie wordt in WCF aangegeven door de securitymode *TransportWithMessageCredential* (*bindings / wsHttpBinding / binding / security @ mode*. Om transportniveauversleuteling te realiseren roepen we onze service nu opnieuw aan via een https-adres in plaats van http. Dit scenario is vooral geschikt om van buiten het trust realm van een Windows-domein veilig te communiceren met een service die binnen dit domein staat. Op de client worden dan UserName-credentials (gebruikersnaam en wachtwoord) meegegeven over een SSL-kanaal naar een service waar authenticatie plaatsvindt tegen bijvoorbeeld een Windows AD (of een eigen SQL Server-authenticatiedatabase). Het meegeven van de cliënt-credentials is eenvoudig te programmeren via de proxy; dit is weergegeven in codevoorbeeld 8. De uiteindelijke configuratie van client en service is weergegeven in codevoorbeelden 9 en10.

Wanneer de client binnen het trust realm van het servicedomein draait, kan ook gebruik worden gemaakt van Windows-credentials. Dit is te configureren door op zowel de client als de service het client credential-type te wijzigen van *UserName* in *Windows*. In dat geval wordt automatisch de Windows securitycontext waar de clientapplicatie in draait, gebruikt voor ondertekening van berichten. Welk credential-type ook wordt gebruikt, in de service zijn deze op te vragen via het type *System.Servicemodel.ServiceSecurityContext*. Dit type bevat een eigenschap met de naam *WindowsIdentity*. Deze wordt automatisch gevuld met de identiteit van de client wanneer de corresponderende Windows-identiteit van de aanroeper direct of indirect is te bepalen. In dit geval geven we via code een UserName-credential ('Secure\John' met wachtwoord 'pass@word1') mee dat correspondeert met een Windows-account (5). In de service kunnen we deze vervolgens weer uitvragen via het *ServiceSecurityContext* type; zie codevoorbeeld 11.

Wanneer een *WindowsIdentiteit* beschikbaar is, wordt ook automatisch een *WindowsPrincipal*-object geconstrueerd dat aan de huidige .NET-thread wordt gekoppeld. Hierdoor is het direct mogelijk om gebruik te maken van de standaard .NET role-based autorisatiefaciliteiten. We kunnen nu de toegang tot de serviceoperatie beperken tot gebruikers die in de rol *Financial* zitten door eenvoudigweg een *PrincipalPermission*-attribuut te plaatsen; zie codevoorbeeld 12.

Wanneer 'Secure\John' in active directory geen lid is van de groep 'Secure\Financial', dan wordt automatisch een SOAP fault-bericht geretourneerd aan de client met daarin de melding *Access Denied*. Naast het gebruik van de standaard Windows-groepen is het ook mogelijk om dit rollenbeheer te laten uitvoeren door de nieuwe ASP.NET roleprovider-functionaliteit in het .NET Framework 2.0. Hiermee is het mogelijk om rollenbeheer via een SQL serverdatabase of via AzMan te beheren. De ASP.NET-roleprovider kunnen we gebruiken door het serviceconfiguratieattribuut *behavior / serviceAuthorization @ principalPermissionMode* de waarde *UseAspNetRoles* te geven. Andere mogelijke waarden van dit attribuut zijn *None, UseWindowsGroups* (default) en *Custom*. De benodigde configuratiewijzigingen en installatiestappen voor het gebruik van AzMan zijn uitvoerig beschreven in [AZM].

Impersonation van de client

In het vorengenoemde voorbeeld hebben we laten zien hoe de WCF-runtime ervoor zorgt dat automatisch de cliënt-identiteit (met zijn bijbehorende security-context) gekoppeld wordt aan de huidige thread in de service. Het gaat hier echter om een logische .NET-thread en niet een fysieke thread van het onderliggende besturingssysteem. Wanneer we dus vanuit onze service een

bestand naar het bestandssysteem willen schrijven, gebeurt dat onder de security-context vanuit het hostingproces. In dit geval is dat dus de security-context van onze applicationpool in IIS (default het NETWORK service-account). Wanneer je daadwerkelijk de security-context van de aanroeper wilt gebruiken bij het benaderen van resources op de service zal je de client-identiteit moeten overnemen. Dit wordt impersonation genoemd en kan eenvoudig worden gerealiseerd met WCF. Wanneer je de complete serviceoperatie wilt impersoneren, kan je het volgende attribuut opnemen bij de implementatie van je serviceoperatie.

```
[OperationBehavior(Impersonation=ImpersonationOption.Required)]
```

Wanneer we nu vanuit onze service bijvoorbeeld een bestand schrijven, zal dit gebeuren onder de security-context van de aanroeper 'SECURE\John'.

WCF en de toekomstige claims-based autorisatie

Naast de traditionele rolgebaseerde autorisatiemechanismen zoals we deze hebben besproken in dit artikel biedt WCF ook ondersteuning voor de toekomstige claims-based autorisatiemechanismen. Het claims-based autorisatiemodel is gebaseerd op een digitale identiteit (*digital identity*). Een digitale identiteit is niets anders dan een verzameling claims ("mijn naam is Erik", "ik werk bij Microsoft", "ik ben ouder dan 18", "mijn ANWB- lidmaatschapnummer is 1234", enzovoort). In plaats van al deze informatie centraal te bewaren in een soort gemeenschappelijke repository (zoals bij Microsoft Passport het geval is) wordt deze decentraal bewaard in een digitale identiteit. Een gebruiker deelt alleen die informatie met een service die belangrijk is voor de service: voor communicatie met de ANWB gebruik ik een digitale identiteit die mijn lidmaatschapnummer en postcode bevatten. Een digitale identiteit biedt ook de mogelijkheid om op zeer gedetailleerd niveau toegang tot services te regelen door specifieke claims te controleren. Zo kan de toegang tot een service worden beperkt tot die personen die ouder zijn dan 18 jaar. Door specifieke claims te kunnen definiëren wordt een explosie van rollen voorkomen, zoals we die zouden hebben wanneer we dit met een traditioneel rolgebaseerd model zouden oplossen. Om de digitale identiteiten eenvoudig te kunnen beheren en gebruiken, worden deze in Windows gepresenteerd aan de gebruiker als een zogenaamde *InfoCard*. Hoe we digitale identiteiten kunnen gebruiken voor het beveiligen van services, kan je lezen in een nieuw artikel in een volgend nummer van het .NET Magazine.

WCF biedt ruime mogelijkheden om beveiliging te configureren

Hoewel beveiliging toch vaak een lastig onderwerp is en blijft, hebben we gezien dat het is terug te brengen tot een aantal kernbegrippen dat we vervolgens eenvoudig kunnen configureren. De basis wordt gevormd door credentials die we bij zowel client als service gebruiken om een identiteit aan te bieden. Vervolgens hebben we gezien dat we geen codeaanpassingen hoeven te doen, maar slechts configuratiewijzigingen hoeven te maken. De configuratiekeuzes van een binding, een credential-type en het beschermingsniveau leiden automatisch tot een beveiligde berichtenstroom tussen client en service

```
ServiceSecurityContext securityCtx = ServiceSecurityContext.Current;
if (securityCtx != null)
{
   primaryIdentity = string.Format("Primary identity \t {0}",
      securityCtx.PrimaryIdentity.Name); // SECURE\John
   windowsIdentity = string.Format("Windows identity \t {0}",
      securityCtx.WindowsIdentity.Name); // SECURE\John
}
currentThreadIdentity = string.Format("Current thread identity \t {0}",
      Thread.CurrentPrincipal.Identity.Name); // SECURE\John
```

Codevoorbeeld 11.

```
[PrincipalPermission(SecurityAction.Demand, Role=@"SECURE\Financial")]
string IBankingServices.TransferMoney(string debitAccount, string
creditAccount, double amount)
{
   //etc
}
```

Codevoorbeeld 12.

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

Referenties

[WSS]	Web Services Security Standard Specification (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss)
[BP]	Web Services Interoperability Organization, Basic Profile, http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile
[SDK]	Microsoft® Windows® Software Development Kit (SDK) for the February 2006 Community Technology Preview (CTP) for Windows Vista and WinFX Runtime Components, http://www.microsoft.com/downloads/details.aspx?FamilyID=9be1fc7f-0542-47f1-88dd-61e3ef88c402&DisplayLang=en
[PPS]	Web Service Security, Patters & Practices, http://www.microsoft.com/downloads/details.aspx?FamilyId=3E02A6C8-128A-47C2-9F39-4082582F3FE1&displaylang=en
[PMI]	Programmeren met Indigo, Gijs de Jong, Microsoft, .NET Magazine #8, http://download.microsoft.com/download/7/9/8/79802a44-7903-402c-a1a7-e7442aa29382/Net8_p23-27_100.pdf
[WSE]	Webservices beveiligen wordt nog eenvoudiger met WSE 3.0 en Visual Studio 2005, Erik van de Ven, Microsoft, .NET magazine #10, September 2005, http://download.microsoft.com/download/1/d/4/1d4eb592-3b40-454c-bafc-31ba2916ddba/Net10_p60-65.pdf
[AZM]	How To: Use Authorization Manager (AzMan) with ASP.NET 2.0, http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000019.asp

Noten

1 Voor een uitleg over algemene WCF-begrippen zoals interfaces, contracten, endpoints en bindings zie [PMI].
2 Zoals reeds in [PMI] is besproken kan je de svcutil-utility gebruiken om een client proxy en configuratiebestand te laten genereren.
3 Je kunt de serviceconfiguratie-editor gebruiken om niet handmatig de bestanden te hoeven wijzigen. Deze tool kan je ook vinden in de Windows SDK [SDK] onder de naam SvcConfigEditor.
4 In dit geval wijzigen we het bestaande endpoint; in de praktijk kan je er ook voor kiezen om een extra endpoint aan te bieden.
5 In dit geval is de PrimaryIdentity gelijk aan de WindowsIdentity. Wanneer we echter clientauthenticatie met certificaat-credentials toepassen en deze afbeelden op een Windows-account (mapClientCertificateToWindowsAccount = true) zal de PrimaryIdentity verschillen van de WindowsIdentity. De PrimaryIdentity bevat dan een verwijzing naar het clientcertificaat en de WindowsIdentity bevat het corresponderende Windows-acount.