Gijs de Jong
Consultant – Microsoft Services

GXA standaarden en webservices enhancements

De toekomst van webservices

Webservices zijn de afgelopen jaren steeds populairder geworden. Vele softwareleveranciers hebben toolkits op de meest uiteenlopende platformen op de markt gebracht. Webservices zijn voornamelijk geïmplementeerd in 'business-to-business' (B2B) scenario's waar twee of meer bedrijven met elkaar integreren over het internet. Ook worden webservices veel in dezelfde organisatie gebruikt om platformverschillen of zelfs devices-verschillen te kunnen overbruggen. Maar wat zijn nou eigenlijk de succesfactoren van webservices?

Webservices maken gebruik van reeds bestaande industriestandaarden zoals XML, XML Schema (XSD), SOAP en HTTP. Een groot voordeel hiervan is dat de reeds opgebouwde expertise de 'learning curve' drastisch verkleind. Alhoewel SOAP transportprotocol onafhankelijk is, zijn nagenoeg alle webservices op het HTTP-protocol geïmplementeerd. Op deze manier wordt voor een groot deel al van bestaande infrastructuur gebruik gemaakt. Dit maakt de implementatie van webservices relatief goedkoop.

Ondanks het succes van webservices zijn er nog de nodige aspecten die niet zijn geadresseerd in de huidige implementatie van SOAP en WSDL. Security is waarschijnlijk hét voorbeeld dat voor velen het meest tot de verbeelding zal spreken. Alle huidige mogelijkheden op het gebied van webservices-beveiliging worden gefaciliteerd door het onderliggende transportprotocol. In het geval van HTTP betekent dit concreet: Windows Integrated (NTLM/Kerberos), Basic Authentication & SSL/TLS of Client Certificates.

In de steeds meer eisende wereld van gedistribueerde applicaties is dit vaak niet voldoende meer. Het gevolg hiervan is dat eigen geïmproviseerde oplossingen worden bedacht en geïmplementeerd. Dat is tijdrovend en complex, en doordat het niet op een standaard manier gebeurt is de interoperabiliteit met andere oplossingen nihil. De mogelijkheden van webservices zullen dus uitgebreid moeten worden om wel aan de eisen van de nieuwe generatie gedistribueerde applicaties te kunnen voldoen.

Deze uitbreiding wordt geboden in de vorm van GXA. De rest van dit artikel zal ingaan op wat GXA nu precies is, en zullen de belangrijkste onderdelen nader toegelicht worden. Aan de hand van een aantal voorbeelden wordt getoond hoe je vandaag de dag al concreet gebruik kan maken van GXA door middel van de Webservices Enhancements SDK.

GXA

GXA staat voor *Global XML Web Services Architecture* en is een framework van protocollen. Deze protocollen verrijken het SOAP-protocol met zaken zoals routing, security, transactions en eventing. De specificaties van deze protocollen beschrijven twee zaken. Hoe SOAP-messages er precies uit moeten zien en hoe ze verwerkt dienen te worden. Deze nieuwe specificaties zijn uitbreidingen op SOAP, die gedefinieerd zijn door gebruik te maken van SOAP-headers. Elk GXA-protocol zal zich dus manifesteren in de vorm van een of meer extra headers in de *SOAP-message*.

Een cruciaal aspect is dat GXA niets zegt over het programmeermodel. Dit is mede een kritische succesfactor gebleken bij de acceptatie van XML. De specificatie beschrijft slechts hoe een geldig XML-document er uit moet zien. Het maakt niets uit of je nou DOM, SAX of System.Xml gebruikt. Uiteindelijk gaat het er alleen om dat het XML-document aan de specificatie voldoet. Dit zelfde geldt voor GXA.

GXA-protocollen

Een aantal marktleiders, waaronder Microsoft, IBM en VeriSign, is betrokken bij het definiëren van de verschillende GXA-protocollen. Ze zijn ook verantwoordelijk voor de oprichting van de Web

Service Interoperability (WS-I) Organization. De WS-I is onder andere in het leven geroepen om de vele ontwikkelingen op het gebied van webservices te coördineren. Er zijn inmiddels al de nodige GXA-protocollen gepubliceerd Hier volgt een overzicht van de meest fundamentele protocollen, die vandaag de dag ook al te gebruiken zijn.

- *WS-Routing* – Maakt het mogelijk een routingpad voor SOAP-messages te definieren. Dit pad kan uit een of meerdere SOAP-intermediars bestaan, voordat het werkelijke endpoint bereikt wordt. Het retourpad voor de response-message is echter optioneel. WS-Routing biedt tevens ondersteuning voor de correlatie tussen SOAP-request- en response-messages.
- *WS-Referral* – Biedt de mogelijkheid tot het dynamisch routeren van SOAP-messages. Dit gebeurt door middel van routinginstructies die zijn gebaseerd op de URI van de binnenkomende SOAP-message.
- *WS-Security* – Maakt integriteit en vertrouwelijkheid van SOAP-messages mogelijk. Integriteit wordt bewerkstelligd door *signing* van de message. Signing wordt gebruikt ter verificatie of de message (of een onderdeel daarvan) onderweg niet is aangepast door derden. Vertrouwelijkheid wordt gegarandeerd door encryptie van de message. Encryptie zorgt ervoor dat de SOAP-message (of een onderdeel daarvan) alleen leesbaar is voor de eigenaar van de correcte (private of symmetrische) key. WS-Security maakt het tevens mogelijk om security-tokens met een SOAP-message mee te sturen.
- *WS-Attachments* – Biedt de mogelijkheid tot het toevoegen van attachments aan een SOAP-message.

Architectuurrichtlijnen GXA

Er is een aantal architectuurrichtlijnen dat als uitgangspunt heeft gediend tijdens het ontwerp van alle GXA-protocollen.

- *Decentraal / Federated* – Er zijn geen centrale servers nodig; de SOAP-message bevat alle benodigde informatie. Dit bevordert schaalbaarheid en maakt een organisatieoverstijgend security-model mogelijk.
- *Modulair* – De GXA-specificaties zijn gericht op een specifiek probleemdomein (bijvoorbeeld routing of security). Je hoeft alleen díe protocollen te gebruiken die van toepassing zijn. Ze fungeren ook als bouwstenen, waar het ene protocol als basis voor een ander protocol kan dienen.
- *Applicatiedomeinonafhankelijk* – GXA-specificaties zijn algemeen en houden dus geen rekening met een specifieke applicatie/industrie; dit maakt standaardisatie eenvoudiger. Het kan uiteraard wel als basis voor een specifiek applicatiedomein dienen door de uitbreidbare aard van de protocollen.
- *Transportonafhankelijk* – Alle GXA-specificaties gaan uit van volledige transportonafhankelijkheid, door alles op SOAP-message-niveau te definiëren. Dit betekent ook dat er niet van de semantiek van een bepaald transportprotocol mag worden uitgegaan. Een goed bijvoorbeeld hiervan is het request/response-model van HTTP.

Tot op heden zijn webservices veelal gebruikt in de vorm van een RPC (Remote Procedure Call) model over HTTP. Het HTTP-protocol stimuleert dit programmeermodel door het request/response-mechanisme. Aangezien SOAP een transportonafhankelijk protocol is kan niet worden uitgegaan van dit request/responsmodel. Bij gebruik van het SMTP-protocol bijvoorbeeld zou dit niet mogelijk zijn.

Een ander aspect van het RPC-model is de 'point-to-point'-communicatie. Hiermee wordt bedoeld dat de consument van de webservice een rechtstreekse verbinding maakt met de webservice. Voor veel gedistribueerde applicaties is dit vaak niet meer voldoende en/of haalbaar. Het is mogelijk dat een SOAP-message gerouteerd moet worden naar een andere server; misschien wel een server die zich op een intern netwerk achter een firewall bevindt. Een bijkomend aspect is dat de transportprotocollen tussen de verschillende schakels niet hetzelfde hoeven te zijn. Zoals in afbeelding 1 is weergegeven, wordt tussen de client en de SOAP-router HTTP gebruikt, en TCP tussen de SOAP-router en de webservice. Dit zal er toe leiden dat webservices steeds meer in de richting van een asynchroon message-georiënteerde manier van communicatie gaan in plaats van het RPC-model.

Afbeelding 1. HTTP tussen client en SOAP-router – TCP tussen SOAP-router en webservice

```
<configuration>
  <system.web>
    <webServices>
      <soapExtensionTypes>
        <add type="Microsoft.Web.Services.WebServicesExtension,
        Microsoft.Web.Services, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=31bf3856ad364e35"
        priority="1"
      group="0" />
      </soapExtensionTypes>
    </webServices>
  </system.web>
</configuration>
```

Afbeelding 2. Web.config van de webservice

Web Services Enhancements (WSE)

De WSE is een software-developmentkit die developers in staat stelt op ASP.NET gebaseerde webservices te verrijken met GXA-protocollen. De huidige versie (1.0) van de WSE biedt ondersteuning van de volgende protocollen: WS-Routing, WS-Referral, WS-Security en WS-Attachments. De WSE is de kortetermijn-implementatie van GXA. Met zekere regelmaat zullen nieuwe versies te downloaden zijn, waarin de laatste

specificaties zijn verwerkt. Er wordt momenteel ook gewerkt aan een langetermijn-implementatie van GXA. Het GXA-platform zal uiteindelijk namelijk volledig in het .NET Framework worden opgenomen. De WSE wordt overigens wel ondersteund door Microsoft.

WSE-configuratie

Om gebruik te kunnen maken van de WSE in een standaard ASP.NET webservice moet een aantal configuratiestappen worden uitgevoerd.
De WSE is geïmplementeerd als een SOAP-extension. Deze moet geconfigureerd worden in het web.config-bestand van de webservice, zoals in afbeelding 2 is weergegeven.

Aan de kant van de webservice-client moet ook een configuratiehandeling worden uitgevoerd. Het proxy-object, dat door het toevoegen van een Web Reference (of WSDL.exe) is gegenereerd, moet worden aangepast. Standaard is de base-class van dit object System.Web.Services.Protocols.SoapHttpClientProtocol, dit moet worden gewijzigd naar Microsoft.Web.Services.WebServicesClientProtocol. Deze class bevindt zich in de WSE-assembly Microsoft.Web.Services.dll. WebServicesClientProtocol biedt door middel van de SoapContext toegang tot de GXA-specifieke SOAP-headers in de message.

Om dit soort WSE-configuratiehandelingen niet handmatig uit te hoeven voeren is hier een configuratietool voor: de WSE Settings Tool. De WSE Settings Tool is een Visual Studio .NET add-in en faciliteert in de configuratie van alle mogelijke WSE-settings. De huidige versie van de settings tool wordt niet ondersteund door Microsoft. In de volgende versie van de WSE zal de settings tool een geïntegreerd onderdeel zijn.

Routing

Het doel van routing is het virtualiseren van de netwerktopologie. Clients sturen SOAP-messages naar een router in plaats van het werkelijke endpoint. De SOAP-message wordt vervolgens door de router geforward naar een eventueel volgende router of naar het endpoint. Het is dus mogelijk dat de client niet weet wat het uiteindelijke endpoint is. Deze kan wel achter een firewall op het interne netwerk staan, waar de client helemaal geen toegang heeft; zie afbeelding 1.

Door de WSE worden twee vormen van SOAP-routing geboden: URI-gebaseerd en contentgebaseerd.
Voor de router zal eerst een HttpHandler geconfigureerd moeten worden, dit gebeurt in het web.config-bestand; zie afbeelding 3.
De routering van SOAP-messages op basis van de URI gebeurt door middel van een *ReferralCache*. De ReferralCache is een configuratiebestand waarin staat beschreven waarnaar SOAP-messages moeten worden geforward. Het configureren van deze cache gebeurt in een aantal stappen. Eerst moet er een configSection worden opgenomen in het

```
<configuration>
  <system.web>
    <HttpHandlers>
      <add  type="Microsoft.Web.Services.Routing.RoutingHandler,
      Microsoft.Web.Services, Version=1.0.0.0,
      Culture=neutral, PublicKeyToken=31bf3856ad364e35"
      path="*.asmx"
        verb="*" />
    </HttpHandlers>
  </system.web>
</configuration>
```

Afbeelding 3. Configuratie HttpHandler in web.config

```
<configuration>
  <configSections>
    <section name="microsoft.web.services"
    type="Microsoft.Web.Services.Configuration.
    WebServicesConfiguration,
      Microsoft.Web.Services, Version=1.0.0.0,
      Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
  </configSections>
</configuration>
```

Afbeelding 4. configSection in web.config

```
<?xml version="1.0" ?>
<r:referrals
    xmlns:r="Http://schemas.xmlsoap.org/ws/2001/10/referral">
    <r:ref>
        <r:for>
            <r:exact>
               http://server1/dotNETMagazine/Router/EndPoint.asmx
            </r:exact>
        </r:for>
        <r:if/>
        <r:go>
            <r:via>
          http://server2/dotNETMagazine/EndPoint/EndPoint.asmx
            </r:via>
        </r:go>
        <r:refId>uuid:fa469956-0057-4e77-962a-81c5e292f2ae</r:refId>
    </r:ref>
</r:referrals>
```

Afbeelding 5. ReferralCache.config

```
<configuration>
  <microsoft.web.services>
    <referral>
      <cache name="ReferralCache.config" />
    </referral>
  </microsoft.web.services>
</configuration>
```

Afbeelding 6. Referral-element in web.config

```
<configuration>
  <microsoft.web.services>
    <diagnostics>
      <trace enabled="true" input="InputTrace.webinfo"
      output="OutputTrace.webinfo" />
    </diagnostics>
  </microsoft.web.services>
</configuration>
```

Afbeelding 7. Neem deze setting op in web.config

```
<wsrp:path soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
soap:mustUnderstand="1" xmlns:wsrp="http://schemas.xmlsoap.org/rp">
      <wsrp:to>
      http://server1/dotNETMagazine/Router/EndPoint.asmx
  </wsrp:to>
</wsrp:path>
```

Afbeelding 8. Trace-bestand

```
<wsrp:path soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
soap:mustUnderstand="1" xmlns:wsrp="http://schemas.xmlsoap.org/rp">
      <wsrp:to>
      http://server1/dotNETMagazine/Router/EndPoint.asmx
  </wsrp:to>
      <wsrp:fwd>
        <wsrp:via>
          http://server2/dotNETMagazine/EndPoint/EndPoint.asmx
      </wsrp:via>
       </wsrp:fwd>
</wsrp:path>
```

Afbeelding 9. De SOAP-router heeft een extra element <wsrp:fwd> toegevoegd aan de Path-header

web.config-bestand; zie afbeelding 4. Maar allereerst moet een configuration section-handler worden toegevoegd aan het web.config-bestand van de router-webservice.

Vervolgens kunnen we een referral cache-bestand aanmaken, in dit geval ReferralCache.config genaamd; zie afbeelding 5. Hierin definiëren we dat alle SOAP-messages naar http://server1/dotNETMagazine/Router/EndPoint.asmx geforward worden naar http://server2/dotNETMagazine/EndPoint/EndPoint.asmx.

Het ReferralCache.config-bestand moet nog wel geladen worden. Dit kan worden bewerkstelligd door een referral-element op te nemen in het web.config-bestand; zie afbeelding 6.

De URL van de Webservice Proxy op de client die naar de *EndPoint* webservice verwijst, moet worden gewijzigd in http://server2/dotNETMagazine/Router/EndPoint.asmx . Het makkelijkste is om in VS.NET de property *URL Behavior* van de proxy op *Dynamic* te zetten. De URL wordt nu uit het app.config-bestand gelezen. Ook hier kan dus de URL worden aangepast.

Nu alles goed geconfigureerd is kunnen we de webservice aanroepen. Het is interessant om te zien welke specifieke SOAP-headers er door de WSE worden toegevoegd. Voor de tracing van SOAP-messages zijn de nodige tools beschikbaar. De SOAP Toolkit 2.0 bevat een handige tool: MSSoapT. De WSE heeft echter ook zelf trace-functionaliteit. Dit kan worden geconfigureerd door in het web.config-bestand de volgende setting op te nemen; zie afbeelding 7. De WSE logt vervolgens alle inkomende SOAP-messages naar InputTrace.webinfo en alle uitgaande berichten naar OutTrace.webinfo.

Als we na het aanroepen van de webservice het trace-bestand (afbeelding 8 en 9) bekijken, zien we dat de SOAP-router een extra element aan de Path-header heeft toegevoegd, namelijk <wsrp:fwd>. Hierin wordt het pad gedefinieerd dat de SOAP-message moet gaan afleggen. Elke stap – in SOAP-terminologie ook wel een 'SOAP intermediary' genoemd – zal worden beschreven in de vorm van een <wsrp:via> element. (Elementen die niet relevant zijn voor het voorbeeld zijn verwijderd.)

Routing van SOAP-messages is ook mogelijk op basis van de inhoud van een SOAP-message; dit wordt ook wel Content Based Routing (CBR) genoemd. De inhoud kan zich zowel in een SOAP-header als in de body bevinden. Voor CBR moet een eigen RoutingHandler geschreven worden. Dit is een class die moet zijn afgeleid van Microsoft.Web.Services.Routing.RoutingHandler. In de RoutingHandler moet de ProcessRequestMessage-methode worden overschreven. Op basis van de inhoud van de SOAP-message ('message' input parameter) kan het te volgen pad van de message worden gemanipuleerd ('outgoingPath' input parameter).

Security

Tot nu toe maakten clients altijd rechtstreeks een verbinding met de werkelijke webservice (point-to-point). Aan-

gezien we steeds meer naar een message-georiënteerde vorm van communicatie gaan, mogelijk met verschillende SOAP-intermediary's is een transportprotocol niet meer gegarandeerd. Zoals op afbeelding 1 zichtbaar is, kan de communicatie van de client met de uiteindelijke webservice wel over verscheidene protocollen verlopen. Ook is het heel goed mogelijk dat de client geen directe toegang heeft tot het uiteindelijke endpoint, omdat deze zich achter een firewall bevindt. De enige manier om een 'end-to-end'-beveiliging van de SOAP-message te kunnen garanderen is door het bericht zelf te beveiligen en niet het transportkanaal.

De WSE ondersteunt de beveiliging van een SOAP-message in de vorm van encryptie en signing. Encryptie zorgt ervoor dat de SOAP-message (of een onderdeel daarvan) alleen leesbaar is voor de eigenaar van de correcte (private of symmetrische) key. Signing van een SOAP-message (of een onderdeel daarvan) wordt gebruikt ter verificatie of de message onderweg niet is aangepast door derden.

Signing

De WSE ondersteunt een aantal manieren van signing van een SOAP-message, namelijk door middel van Username & Password, X.509 certificaten en Custom Binary Token. In afbeelding 10 is te zien hoe een UsernameToken wordt gebruikt voor de signing van een SOAP-message. Een UsernameToken bestaat uit een gebruikersnaam en wachtwoord. De PasswordOption.SendNone enumerator geeft aan dat het wachtwoord níet met de message moet worden meegestuurd. Wanneer een UsernameToken gebruikt wordt om een SOAP-message te signen zal op de webservice een zogenaamde passwordProvider geregistreerd moeten worden. Dit gebeurt in het web.config-bestand, zoals in afbeelding 11 is weergegeven.

De PasswordProvider is een class die de IPasswordProvider-interface implementeert. De GetPassword-methode zal door de WSE automatisch worden aangeroepen en behoort het juiste password voor de specifieke gebruiker te retourneren. Deze gegevens staan normaal gesproken typisch in een database. Het geretourneerde password zal als key dienen voor de creatie van een digital signature van de SOAP-message. Komt deze overeen met de digital signature die de client heeft meegestuurd, dan is de SOAP-message tijdens transport niet gewijzigd en is sprake van een succesvolle authenticatie van de gebruiker; zie afbeelding 12.

```
WebService.Calculator Calc = WebService.Calculator();

UsernameToken token = new UsernameToken("gijs","Str0ng!psw",
PasswordOption.SendNone);
SoapContext requestContext = Calc.RequestSoapContext;
requestContext.Security.Tokens.Add(token);
requestContext.Security.Elements.Add(new Signature(token));

int value1 = 5;
int value2 = 7;
int result = Calc.Sum(value1,value2);
```

Afbeelding 10. Gebruik UsernameToken voor signing van een SOAP-message

```
<configuration>
  <microsoft.web.services>
   <security>
          <passwordProvider
       type="dotNETMagazine.PasswordProvider, MyAssembly" />
    </security>
  </microsoft.web.services>
</configuration>
```

Afbeelding 11. Registratie passwordProvider in web.config

```
using System;
using Microsoft.Web.Services.Security;

namespace dotNETMagazine
{
   public class PasswordProvider:IPasswordProvider
   {
       public string GetPassword (UsernameToken token)
       {
            // Check username
            if (token.Username == "gijs")
                 // Typically perform database lookup and return password
                 return "Str0ng!psw";
            else
                 return null;
       }
   }
}
```

Afbeelding 12. Succesvolle authenticatie gebruiker

Encryptie

Signing dient ter verificatie of een SOAP-message tijdens het transport is aangepast. De inhoud van de message is echter wel gewoon leesbaar. Om te voorkomen dat derden de inhoud van een message kunnen lezen, zal encryptie moeten worden toegepast. De WSE ondersteunt encryptie van SOAP-messa-

```
WebService.Service1 proxy = new WebService.Service1();

DimeAttachment da = new DimeAttachment("image/gif",
TypeFormatEnum.MediaType, @"c:\image.jpg");
proxy.RequestSoapContext.Attachments.Add(da);
proxy.UploadFile();
```

Afbeelding 13. Een bestand wordt toegevoegd aan de attachments-collectie van de SOAP-requestcontext

```
[WebMethod]
public void UploadFile()
{
  SoapContext requestContext = HttpSoapContext.RequestContext;
  DimeAttachment da = requestContext.Attachments[0];
  // SaveFile(da.Stream);
}
```

Afbeelding 14. Webservice zonder DimeAttachment

ges op basis van X.509-certificaten, Shared Secret en Custom Binary Token.

Attachments

De WSE biedt ook de mogelijkheid om DIME-attachments (Direct Internet Message Encapsulation) aan SOAP-messages toe te voegen. DIME is een packaging-mechanisme voor de encapsulatie van data van een willekeurig formaat. In afbeelding 13 is te zien hoe een bestand (image.jpg) als DimeAttachment wordt toegevoegd aan de attachments-collectie van de SOAP-requestcontext.

In afbeelding 14 is de implementatie van de webservice te zien, waar de DimeAttachment uit de SOAP-requestcontext wordt gehaald. Het bestand (image.jpg) is vervolgens te benaderen door de Stream-property (type System.IO.Stream) van het DimeAttachment-object.

GXA verrijkt webservices

GXA is dus een framework van protocollen gebaseerd op SOAP. Deze protocollen verrijken webservices met zaken zoals routing, security, transactions en eventing. De Web Services Enhancements (WSE) SDK stelt developers vandaag de dag in staat GXA-functionaliteit aan hun op ASP.NET gebaseerde webservices toe te voegen. In de toekomst zal GXA volledig geïntegreerd zijn met het .NET Framework.

Nuttige internetadressen

- XML Web Services Developer Center Home - http://msdn.microsoft.com/webservices
- Web Services Interoperability Organization (WS-I) - http://www.ws-i.org

---

Titel:
ISBN: 0-7356-1588-8
Auteur: Michael Howard en David LeBlanc

Kwaadwillende mensen dringen regelmatig andermans toepassingen binnen om zich creditcardnummers toe te eigenen of websites te ontregelen. Deze constante bron van ergernis kost bedrijven elk jaar miljoenen dollars. Beveiligingsfouten die softwarearchitecten, -ontwerpers en -ontwikkelaars maken, zijn daar mede debet aan. WRITING SECURE CODE reikt kant-en-klare remedies aan. Dit boek propvol interessante weetjes beschrijft op een unieke manier de belangrijkste aspecten die bij het maken van beveiligde toepassingen om de hoek komen kijken. Dit beslaat het hele ontwikkelproces, vanaf het ontwerpstadium en het schrijven van robuustecode, tot het testen van toepassingen op beveiligingsrisico's.
WRITING SECURE CODE reikt software-ontwerpers, -architecten, -ontwikkelaars en -testers alle theorieën en praktische technieken aan die zij nodig hebben om een goede beveiliging te garanderen. Aan bod komen o.a. beveiligingsprincipes, hoe je bij het ontwerpen, schrijven van code en het testen rekening moet houden met de beveiliging, hoe je beveiligde code voor Microsoft® .NET API's schrijf, waarom bedrijven zo slordig omspringen met hun beveiliging en de tien onbetwiste regels voor (het beheer van) beveiliging. Ontwikkelaars die dit boek hebben gelezen, kunnen er met een gerust hart van uitgaan dat hun code niet alleen snel, maar ook veilig is. Beide auteurs zijn gerenommeerde beveiligingsdeskundigen van Microsoft. Zij hebben in het verleden meegewerkt aan het oplossen van allerlei lastige computerbeveiligingsvraagstukken.

Over de auteur(s):
Michael Howard is beveiligingsmanager bij het Microsoft Windows XP-team. Hij houdt zich bezig met de beveiligingsaspecten in de ontwerp-, programmeer- en testfases. Hij is de hoofdauteur van DESIGING SECURE WEB-BASED APPLICATIONS FOR MICROSOFT WINDOWS 2000 dat bij Microsoft Press is verschenen. Voordat Michael Howard bij het Windows XP-team kwam, werkte hij aan nieuwe webservertechnologieën en IIS. Hij is sinds 1992 werkzaam bij Windows NT®security.
David LeBlanc is een senior beveiligingsspecialist in ITG bij Microsoft. De verdediging van het Microsoft-netwerk tegen aanvallen van buitenaf is zijn voornaamste taak.