Mark Blomsma (MVP)
is softwarearchitect en medeoprichter van Omnext.NET bv (www.omnext.net). Omnext.NET bv specialiseert in het migreren en renoveren van legacy-systemen naar .NET-applicaties (www.software-renovation.net). Naast zijn werkzaamheden voor Omnext.NET is Mark ook de C# Groupleader voor Software Developers Group Netherlands (www.sdgn.nl).

Tooltime: Web Services Enhancements 2.0

VEILIG WERKEN MET OPEN PROTOCOLLEN

Wie kent ze niet? De programma's op tv waar in 30 minuten een huis compleet wordt gerenoveerd en de bouwlui ook nog tijd hebben voor een bak koffie. Doe je het zelf, dan duurt het toch meestal wat langer. Hoe kan dat toch? Opvallend is dat behalve ervaring de mannen op tv voor elk stapje dat ze uitvoeren een ander stuk gereedschap hebben. En het wordt al snel duidelijk dat met het juiste gereedschap alles een stuk eenvoudiger wordt. Dit geldt ook voor het beveiligen van webservices.

Webservices zijn bedoeld om systemen open te maken. De integratie van systemen via een open netwerk zoals Internet, op basis van open protocollen, legt de lat voor beveiliging hoog, erg hoog. Security is dus niet eenvoudig, maar we zullen zien dat het met de Web Service Enhancements 2.0 wel makkelijker wordt. Dit artikel is gebaseerd op de Web Services Enhancements 2.0 Technology Preview (WSE 2.0) die kan worden gedownload op de MSDN-website. De Web Services Enhancements vormen een add-on voor Visual Studio.NET en het .NET Framework. De WSE vormen een tussentijdse update op het .NET Framework, bedoeld om op deze manier bij te blijven met de ontwikkelingen die gaande zijn op het gebied van webservice-standaarden.

Veilig communiceren

Wat is veilig communiceren? In ieder willekeurig spionageverhaal kun je lezen dat de veiligste manier om met je informant te communiceren is om hem op een geheime plek te ontmoeten, verzekerd te zijn dat er geen afluisterapparatuur is en hem recht in de ogen te kijken als hij zijn informatie overdraagt. In de wereld van open systemen, waar computers automatisch met elkaar moeten communiceren, zullen we het iets anders moeten aanpakken. We willen echter wel verzekerd zijn van de integriteit van de boodschap. Dit betekent dat we zeker willen zijn dat de afzender is wie hij zegt dat hij is en dat de boodschap - tussen versturen en aankomst - niet is gelezen en/of gehackt door een derde partij.
In dit artikel werken we uit welke zaken we moeten regelen om te komen tot een veilige webservice.
De client-applicatie zal de volgende functionaliteit implementeren:
1. Aanmaken van een bericht.
2. Aanmaken van een security token.
3. Ondertekenen van het bericht met het security token.
4. Encrypten van het bericht met het security token.
5. Versturen van het bericht naar de webservice.

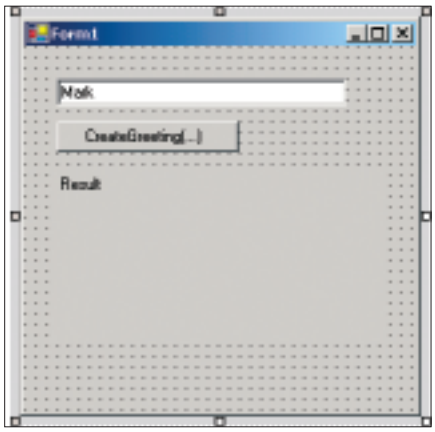
De webservice moet dus in staat zijn om:
1. Het bericht te decrypten en hierbij te controleren dat het bericht niet is gewijzigd is.
2. De afzender te valideren en authenticeren op basis van de handtekening.

We zullen het een en ander doorlopen aan de hand van een eenvoudige webservice. In Visual Studio.NET starten we een nieuwe solution, in deze solution creëren we een nieuw project van het type ASP.NET Web Service. In het project maken we een webservice die lijkt op HelloWorld() maar dan een persoonlijke boodschap geeft. We noemen de service 'Hello' en de methode 'CreateGreeting'. Zie codevoorbeeld 1.

```
[WebMethod]
public string CreateGreeting( string name )
{
 return "Hello " + name + ". Welcome to my web service!";
}
```

Codevoorbeeld 1

Om de webservice te testen kun je op de class file gaan staan en kiezen voor 'View in Browser'. ASP.NET zal een fraaie testpagina genereren waarin een parameter kan worden ingevoerd en waar, na een druk op de knop, een XML-file wordt getoond met het resultaat van de webservice. We moeten nu een bericht gaan samenstellen om deze methode aan te roepen. Als we het hebben over het aanroepen van een webservice wordt er met het SOAP-protocol een XML-document verstuurd van de client naar de server. Dit XML-document wordt het bericht genoemd. We hebben een client-applicatie nodig om de webservice aan te roepen. We voegen aan onze solution een Windows Application toe. Op ons scherm (zie figuur 1) zetten we een textbox, button en label. Om de webservice aan te roepen moeten we een Web Reference toevoegen. Klik op het winform-project en kies 'Add Web Reference'. Selecteer nu op de local host machine de webservice die we eerder hebben aangemaakt. Als die nog niet zichtbaar is, compileer dan eerst het webservice-project.



Afbeelding 1. Form1

```
private void btnCreateGreeting_Click(object sender, System.EventArgs e)
{
  MyHelloService.Hello service = new MyHelloService.Hello();
  this.lblResult.Text = service.CreateGreeting( this.txtName.Text );
}
```

Codevoorbeeld 2

```
<?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
  <CreateGreeting xmlns="http://tempuri.org/">
    <name>Mark</name>
  </CreateGreeting>
  </soap:Body>
</soap:Envelope>
```

Codevoorbeeld 3

```
using Microsoft.Web.Services.Security;
using Microsoft.Web.Services.Security.X509;
```

Codevoorbeeld 4

```
// Maak een security token
UsernameToken token;

// Vul username en wachtwoord in, geef aan dat het wachtwoord
// hashed verstuurd moeten worden en niet in plaintext.
string username = "Mark";
string password = "Password";
token = new UsernameToken( username, password,
PasswordOption.SendHashed );
```

Codevoorbeeld 5

```
// Toevoegen van een security token
service.RequestSoapContext.Security.Tokens.Add( token );

// Ondertekenen van de request op basis van het token
service.RequestSoapContext.Security.Elements.Add( new Signature( token));

// Maak een greeting met als parameter de inhoud van de textbox
this.lblResult.Text = service.CreateGreeting( this.txtName.Text );
```

Codevoorbeeld 6

Je kunt de webservice een alias geven. In dit voorbeeld noem ik hem 'MyHelloService'. Onder de knop kunnen we nu de volgende code plaatsen om de webservice aan te roepen. Zie codevoorbeeld 2.

Codevoorbeeld 2 werkt nu al, maar zoals je ziet is er nog niets aan security gedaan. Zie codevoorbeeld 3 als we nu de SOAP-request bekijken die wordt verzonden.

Deze data kunnen met een tool als 'Advanced HTTP Packet Sniffer' zo van de lijn geplukt worden. Handig voor testen en debuggen, maar voor security natuurlijk een drama. Tijd om wat aan de beveiliging te gaan doen.

We zullen UsernameSigning gebruiken om de aanroep van onze webservice te beveiligen. Hiervoor moeten we, voordat we de webservice gaan aanroepen, een security token hebben. We maken gebruik van een UsernameToken uit Microsoft.Web.Services. Om gebruik te kunnen maken van deze namespace moeten we wel eerst een referentie opnemen naar de .NET-component 'Microsoft.Web.Services.dll'.

Vervolgens voegen we in de header van onze pagina de volgende using-statements toe. Zie codevoorbeeld 4.

We beschikken nu over de benodigde referenties om een security token aan te maken. Met onderstaande code maken we een token met username 'Mark' en wachtwoord "Password". Helaas beschikken we bij het gebruik van de Technology Preview van WSE 2.0 nog

niet over IntelliSense-help in Visual Studio.NET. Hopelijk zullen we hierover in de releaseversie wel kunnen beschikken. De geïnstalleerde helpfiles zijn overigens, alhoewel niet compleet, wel al zeer bruikbaar. Zie codevoorbeeld 5.

Het token moet worden toegevoegd aan de request en wordt vervolgens gebruikt om de request te ondertekenen. Hiervoor is in de Web Service Enhancements een toevoeging gedaan op de webservice proxy. We maken gebruik van de RequestSoapContext-property om de eigenschappen van het SOAP-bericht aan te passen. Zie codevoorbeeld 6.

Tip: Soms zijn de WSE-properties nog niet beschikbaar op de gegenereerde proxy. Om hiervan toch gebruik te maken moeten we een aanpassing doen in de gegenereerde proxy-code. Dit doen we door op de declaratie van onze service te staan (MyHelloService.Hello) en met de rechtermuisknop te kiezen voor 'Go to Definition'. We komen nu in de source van de proxy. We moeten de proxy nu laten erven van Microsoft.Web.Services.WebServicesClientProtocol.

Even recapituleren:
1. We hebben een soap request (= het bericht).
2. We hebben een security token.
3. We hebben het security token toegevoegd aan het bericht.
4. We hebben het bericht ondertekend.
5. We versturen het bericht.

Als we nu de solution in debug-mode opstarten, zien we dat er automatisch een foutmelding ontstaat op de server en wel met de volgende melding: "The security token could not be authenticated or authorized". Bij het ontvangen van een signed request gaat WSE 2.0 automatisch controleren of de tokens in het bericht wel geldig zijn. We hebben echter nog helemaal niet aangegeven hoe dit moet gebeuren. We hadden al eerder besloten om te gaan autoriseren op basis van username en wachtwoord. Aan de serverkant moet nu geregeld worden dat de webservice 'weet' hoe de controle van username en wachtwoord moet plaatsvinden. Hiervoor voegen we in het webservice project een nieuwe class toe met de originele naam 'MijnSecurityTokenManager'. Zie codevoorbeeld 7.

```
using Microsoft.Web.Services;
using Microsoft.Web.Services.Timestamp;
using Microsoft.Web.Services.Security;
using Microsoft.Web.Services.Security.Tokens;

namespace NETMagazine4Demo
{
  /// <summary>
  /// MijnSecurityTokenManager implementeert een eigen algoritme voor
  /// het bepalen van het wachtwoord op basis van de username.
  /// </summary>
  public class MijnSecurityTokenManager : UsernameTokenManager
  {
    /// <summary>
    /// Returns het wachtwoord dat bij de username hoort.
    /// </summary>
    /// <param name="token">The username token</param>
    /// <returns>The password for the username</returns>
    protected override string AuthenticateToken( UsernameToken token )
    {
        // Iedereen die het wachtwoord weet mag binnenkomen.
        // Voor ons voorbeeld controleren we niet eens of
        // token.User == "Mark"
        return "Password";
    }
  }
}
```

Codevoorbeeld 7

```
<configSections>
 <section name="microsoft.web.services"
    type="Microsoft.Web.Services.Configuration.WebServicesConfiguration,
    Microsoft.Web.Services, Version=2.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
</configSections>

<microsoft.web.services>
 <security>
  <securityTokenManager
    type="NETMagazine4Demo.MijnSecurityTokenManager, NETMagazine4Demo"
    xmlns:wsse=http://schemas.xmlsoap.org/ws/2002/12/secext
    qname="wsse:UsernameToken" />
  </security>
</microsoft.web.services>
```

Codevoorbeeld 8

Hoe werkt dit? De classes uit Microsoft.Web.Services zullen bij het controleren van de tokens in ons bericht ons security token tegenkomen. Vervolgens moet worden gecontroleerd of het wachtwoord dat in het token is meegestuurd wel klopt. Hiervoor wordt het token aangeboden aan MijnSecurityTokenManager. Als het wachtwoord in het token overeenkomt met het resultaat van de functie 'AuthenticateToken(UsernameToken token)' dan is het token valide. Alleen de class aanmaken is nog niet voldoende. We moeten ook nog configu-

reren dat de webservice de class 'MijnSecurityTokenManager' als token manager gaat gebruiken. Dit doen we in de web.config van de webservice. Hiervoor voegen we een nieuwe sectie, 'microsoft.web.services', toe aan de config-file. Hierin verwijzen we naar de assembly en classname waar de token manager kan worden gevonden. Zie codevoorbeeld 8.

De webservice weet nu hoe het security token gevalideerd moet worden, en onze request voor een greeting werkt nu met behulp van autorisatie. Natuurlijk zul je in een productieomgeving de username en het wachtwoord niet hardcoded in de sources opnemen, maar eerder uitlezen uit de principal op de current thread (Environment.UserName). Voor de controle van het wachtwoord kun je het wachtwoord opzoeken in de database of een XML-bestand. Voor het gebruik van certificaten worden ook functies en classes beschikbaar gesteld in WSE 2.0.

Encryptie

De autorisatie is nu geregeld, maar we sturen de data nog steeds als leesbare XML over de lijn. Behalve het ondertekenen van ons bericht willen we het ook nog versleutelen. Zo wordt voorkomen dat derden het bericht kunnen inzien of aanpassen. Om een bericht te kunnen encrypten hebben we niet genoeg aan een UsernameToken. Dit token is niet geschikt voor encryptie. We zullen gebruik moeten maken van een X509SecurityToken. Hiervoor moeten we een X509-certificaat inlezen en hiermee de data in het bericht encrypten. Een alternatief zou kunnen zijn om zelf een custom token te maken en hiermee zelf onze encryptie te regelen. In de set-up van WSE2.0 wordt een aantal voorbeeldcertificaten meegeleverd. Deze zullen we voor dit voorbeeld gaan gebruiken. Je moet ze wel zelf installeren op je machine. Kijk in 'C:\Program Files\Microsoft WSE\v2.0\Samples\Sample Test Certificates\readme.htm' voor meer informatie over hoe dit moet.

Met onderstaande functies kunnen we een X509SecurityToken maken op basis van een certificaat. We maken voor het inpakken van het bericht daarbij gebruik van de public key van het certificaat. Zie codevoorbeeld 9.

```
// (Base64String) Key waaronder het certificaat is opgeslagen
public static string ClientBase64KeyId = "gBfo0147lM6cKnTbbMSuMVvmFY4=";

/// <summary>
/// Lees het WSE2.0 sample token.
/// </summary>
/// <returns></returns>
public static X509SecurityToken GetSampleToken()
{
  return GetX509SecurityToken( Convert.FromBase64String(
     ClientBase64KeyId ) );
}

/// <summary>
/// Lees een X509 token uit de CurrentUserStore.
/// </summary>
/// <param name="certificateKey">Key waaronder het
/// certificate in de  store staat</param>
/// <returns></returns>
public static X509SecurityToken GetX509SecurityToken(
    byte[] certificateKey )
{
  X509SecurityToken token = null;

  // Open the CurrentUser Certificate Store
  X509CertificateStore store;
  store = X509CertificateStore.CurrentUserStore(
    X509CertificateStore.MyStore );
  // Exit als er niet gelezen mag worden
  if( !store.OpenRead() ) return null;

  // Zoek in de CurrentUserStore naar het gewenste certificate
  X509CertificateCollection certs =
    store.FindCertificateByKeyIdentifier( certificateKey );

  if (certs.Count > 0)
  {
    // Lees het certificaat en maak er een token van
    token = new X509SecurityToken( ((X509Certificate) certs[0]) );
  }

  return token;
}
```

Codevoorbeeld 9

We moeten dit token nu wederom toevoegen aan ons bericht. Dit gebeurt op dezelfde wijze als het ondertekenen van het bericht. Zie codevoorbeeld 10.

Als we nu het voorbeeld opstarten krijgen we de volgende foutmelding "An invalid security token was provided --> The certificate's trust chain could not be verified. The CERT_CHAIN_POLICY_STATUS return code is 2148204809." Dit komt omdat we de webservice nog niet hebben geconfigureerd om certificaten te kunnen controleren. Klik nu in de Solution Explorer met de rechtermuisknop en kies 'WSE Settings 2.0'. Op het tabblad 'Security' kunnen we aangeven hoe de webservice moet zoeken naar certificaten.
We geven aan dat de X509-certificaten ten behoeve van decryptie te vinden zijn in de

```
// Het versleutelen van de data kan alleen als het token
// dit ook ondersteunt. Hiervoor hebben we een X509 token
// nodig, want met een UsernameToken kan dit niet.

// Via een util-functie even het WSE sample certificate uitlezen
X509SecurityToken x509Token = Util.GetSampleToken();

// Het token dat wordt gebruikt voor de encryptie
// moet worden meegezonden
service.RequestSoapContext.Security.Tokens.Add( x509Token );

// Geef aan dat de data ge-encrypt moet worden
service.RequestSoapContext.Security.Elements.Add(
   new EncryptedData( x509Token ) );
```

Codevoorbeeld 10

```
<soap:Body>
 <xenc:EncryptedData Id=
    "EncryptedContent-b36fcbeb-3111-4e6a-b671-6d5b70410c16"
  Type=http://www.w3.org/2001/04/xmlenc#Content
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod Algorithm=
    "http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
  <xenc:CipherData>
   <xenc:CipherValue>
DWDox9HxQJN7nQCQyY+GiZO3YNBFmT7m/Dm8D1KW985ZJ0ETTifyk1Jw9dHjtSHKb2y2
D6QzoVkVUonC5z4G6Yptlh3DzFX/8Lq5OXHHDcmsViWoHzEExA==
   </xenc:CipherValue>
  </xenc:CipherData>
 </xenc:EncryptedData>
</soap:Body>
```

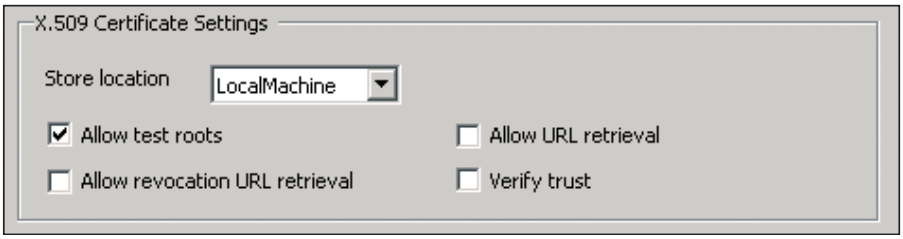
Codevoorbeeld 11

Local Machine store. Omdat we werken met testcertificaten zetten we het vinkje voor 'Allow test roots' aan. Als we nu de applicatie starten zullen we zien dat alles prima werkt. Er is een kans dat je een exception krijgt met hierin de melding 'keyset does not exist'. Indien dit het geval is, kan de webservice de decryptie-sleutel niet vinden of niet laden. De webservice draait onder het ASPNET account. Dit account moet full access hebben op de decryption keys in de directory 'C:\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA\MachineKeys'. Controleer of de ASPNET accountrechten heeft, zo niet: deel die dan uit.

We kijken nog even naar de body van het SOAP-bericht dat over de lijn gaat. We willen natuurlijk zeker weten of het allemaal inderdaad werkt. We laten hier alleen de body zien, want inmiddels is de SOAP-header bijna twee A4-tjes lang geworden. Zie codevoorbeeld 11.

De WSE 2.0 classes pakken het bericht eigenlijk automatisch uit aan de webservice-kant. Dit gebeurt met behulp van de private key van de server. Er wordt gebruik gemaakt van het standaard PKI-mechanisme, zoals dit ook bij bijvoorbeeld HTTPS wordt gebruikt. Is het uitpakken van het bericht succesvol, dan zal het bericht niet zijn aangepast tijdens het transport. De WSE 2.0 classes gebruiken hiervoor overigens de digitale signature.

We hebben nu gezien dat we een bericht kunnen maken, ondertekenen, encrypten en versturen naar een webservice. Op de webservice hebben we onze eigen username / wachtwoordroutine geïmplementeerd en door middel van configuratie hebben we aangegeven hoe de decryptie met behulp van X509-certificaten plaats moet vinden. Het aantal regels code dat we voor dit alles nodig hebben is eigenlijk bijzonder klein; en dit geldt eens te meer als we in ogenschouw nemen hoe complex de XML van de SOAP-message is.



Afbeelding 2. X.509 Certificate settings

Relevante links

http://msdn.microsoft.com/webservices
http://www.w3c.org
http://www.oasis-open.org
http://www.link-rank.com/sniffer/
http://www.ws-i.org/

Noot:

Een laatste opmerking over de WSE 2.0 technologie preview is wel op zijn plaats in verband met de toekomst. Op de Professional Developers Conference oktober 2003 is meer duidelijkheid gegeven over WSE en de implementatie van webservices in de volgende generatie Windows; codenaam Longhorn. De WSE technologie preview kit is primair bedoeld om kennis op te doen over de laatste ontwikkelingen op gebied van webservice-standaarden.