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

.NET Role-Based en Code Access Security

Een complete en open set van security-mechanismen

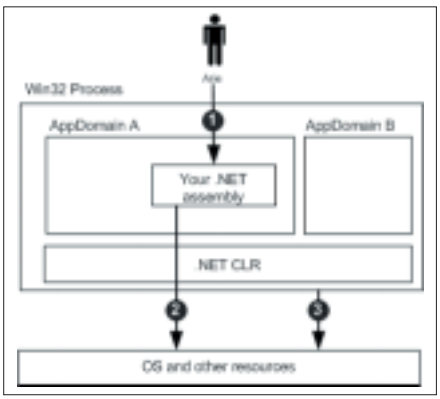
Op dit moment zijn er op het gebied van software-ontwikkeling twee belangrijke trends ontstaan. Ten eerste zien we dat applicaties steeds meer verbonden worden met internet (via webservices). De andere trend is die van automatische code deployment (over internet). Deze twee bewegingen zullen niet alleen het ontwikkelen van software sterk veranderen maar betekenen tevens dat er opnieuw nagedacht moet worden over de beveiliging van systemen. In veel projecten wordt security beschouwd als een afterthought. Hier willen we echter stil staan bij de die aspecten van security die je vanaf het begin van je project zou moeten meenemen.

Als we naar security kijken kunnen we drie niveaus onderscheiden:

1. Role based security
2. Code access security
3. OS security

Op het eerste niveau wordt gecontroleerd of een persoon (of identiteit) wel geautoriseerd is om bepaalde functionaliteit aan te roepen. Het mechanisme om dit in .NET te regelen wordt role-based security genoemd: iemand mag een stuk functionaliteit aanroepen omdat hij een bepaalde rol heeft. Het tweede niveau legt vast wat een bepaald stuk code mag. Dit wordt in .NET code access security genoemd: een bepaalde stuk code mag naar het filesysteem schrijven omdat het afkomstig is van een betrouwbare partij. De twee verschillende niveaus van beveiliging zijn complementair: ze vullen elkaar volledig aan en kunnen vaak niet zonder elkaar. Het uitvoeren van betrouwbare code door een ongeautoriseerd persoon is immers minstens zo gevaarlijk als het uitvoeren van onbetrouwbare code door een wel geautoriseerd persoon. Als laatste onderkennen we nog de security services die onder de .NET runtime liggen en door het besturingssysteem worden aangeboden.

Wanneer we .NET code uitvoeren op een secure besturingssysteem zoals NT, Windows 2000 of XP kunnen we niet zomaar resources als NTFS aanspreken. Om dit te kunnen moet het proces waarin je .NET assembly in is geladen wel de juiste credentials hebben. Afbeelding 1 geeft aan op welk niveau security wordt gecontroleerd.



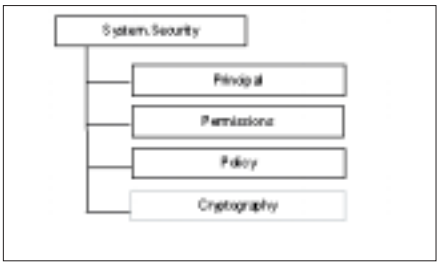
Afbeelding 1. De drie niveaus van beveiliging[1]

Ik geef eerst een overzicht van de .NET security namespace. Daarna leg ik de .NET security-niveaus 1 en 2 uit. Beveiliging op het gebied van het besturingssysteem (niveau 3) laat ik buiten beschouwing.

Namespace overzicht

De System.Security namespace is onderverdeeld in vier subnamespaces. De eerste subnamespace bevat type definities voor het omgaan met principals. Een principal is een identiteit (zoals bijvoorbeeld de naam van een persoon als Arie of Bert) met zijn bijbehorende rollen. De principal vormt het

1 Een appdomain kan je beschouwen als een logisch proces in .NET binnen een proces van het besturingssysteem.



Afbeelding 2. System.Security namespace overzicht

hart van de .NET role-based security. De volgende twee namespace, permissions en policy worden gebruikt voor het tweede niveau van security, namelijk code access security. De permissions vormen het hart van code access security. De laatste namespace bevat managed implementaties van een aantal standaard symetrische (zoals 3DES) en asymmetrische (zoals RSA) encryptie-algoritmen. Deze laatste namespace behandel ik verder niet in dit artikel.

.NET Role-based security

Zoals we hierboven reeds hebben aangegeven vormt de principal het hart van role-based security. De type definitie van een principal staat in afbeelding 3. Deze interface is vrij eenvoudig en kan dus ook makelijk zelf geïmplementeerd worden om je eigen principal te maken. Dit zal later worden toegelicht. De identity stelt een geauthenticeerde gebruiker voor. Deze gebruiker kan bijvoorbeeld geauthenticeerd zijn door Windows, een .NET passport of een zelf ontwikkeld authenticatie-mechanisme.

```
namespace System.Security.
   Principal {
public interface IPrincipal {
IIdentity identity{get;}
bool IsInRole(string role);
  }
}
}
```

Afbeelding 3.

WindowsPrincipal

In System.Security.Principal vinden we naast deze interface definitie ook twee standaard implementaties voor IPrincipal. De eerste is WindowsPrincipal. Deze principal stelt je in staat om in code te controleren of de huidige windows user wel in een bepaalde windows-groep voorkomt. In een eenvoudig voorbeeld (zie afbeelding 4) schrijven we een bestand naar het onderliggende NTFS filesysteem. We onderzoeken wat de huidige principal is. Deze is op te vragen met de CurrentPrincipal property van de huidige thread.

```
AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.
  WindowsPrincipal);
Console.WriteLine("Identity van huidige thread " +
  Thread,CurrentPrincipal.Identity.Name);
```

Afbeelding 4.

```
if (Thread.CurrentPrincipal.IsInRole(@"MACHINE_OR_DOMAIN\Managers"))
WriteToFile();
 else
   Console.WriteLine("U bent niet bevoegd!");
```

Afbeelding 5.

```
[PrincipalPermission(SecurityAction.Demand, Role=
  @" MACHINE_OR_DOMAIN\Managers")]
static void WriteToFile()
{
  // implementatie hier
}
```

Afbeelding 6.

Om de WindowsPrincipal te kunnnen opvragen moeten we eerst in ons appdomain de PrincipalPolicy zetten. Hierdoor zal de CLR een WindowsPrincipal-object construeren op basis van de huidige OS thread credentials. Standaard komt deze overeen met de process credentials en de credentials van de huidige ingelogde windowsgebruiker. Vervolgens wordt de principal aan de huidige logische .NET thread gehangen en kunnen we hem dus opvragen.
Nu we de principal hebben kunnen we vragen of we in de rol van Managers zitten. Dit gaat eenvoudigweg met de methode IsInRole. In dit voorbeeld gaan we er van uit dat de opgegeven rol `MACHINE_OR_DOMAIN \Managers` op de machine of binnen het windows domein bekend is. Hoewel we op deze manier dus eenvoudig role-based security kunnen doen, is het nog maar de vraag of het schrijven naar het bestand daadwerkelijk lukt. Dit hangt immers af van het derde niveau van security dat we onderkend hadden: het besturingssysteem. Het proces dat deze code executeert zal op NTFS-niveau schrijfrechten moeten hebben in de juiste directory. Met het voorbeeld in afbeelding 5 is duidelijk te zien dat je beveiliging integraal op verschillende niveaus moet beschouwen om tot een goede oplossing te komen.
Naast deze imperatieve schrijfwijze voor role based security is er ook een declaratieve stijl mogelijk. Hiervoor is een speciale permission beschikbaar in de System.Security.Permissions namespace. Doordat we de voor de functie WriteToFile() een attribuut opnemen die de rol van Managers vereist kunnen we nu de functie eenvoudigweg aanroepen zonder de if-then constructie (zie afbeelding 6). In het onderdeel '.NET Code access security' wordt verder uitgelegd hoe permission gebruikt kan worden.

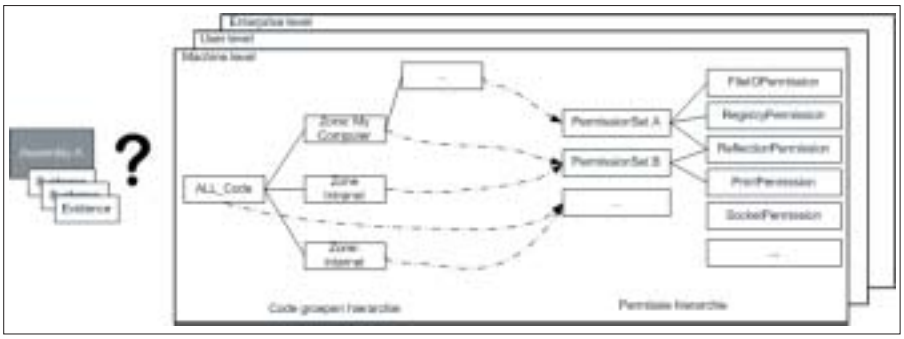
Wanneer je de principal slechts eenmaal nodig hebt in je code is het efficienter om deze niet via een AppDomain.SetPrincipalPolicy aan je thread te hangen, maar om deze zelf te creëeren op de manier in afbeelding 7 en vervolgens IsInRole te gebruiken. Declaratieve security is op deze manier helaas niet mogelijk.

GenericPrincipal

Naast de WindowsPrincipal vinden we in de System.Security.Principal namespace nog een ander standaard type dat IPrincipal implementeert, te weten de GenericPrincipal. Deze bevat een triviale implementatie die als basis kan dienen voor een eigen te ontwikkelen authenticatie-mechanisme. Dit kan handig zijn wanneer je bijvoorbeeld de relatie tussen identiteit en rollen niet door Windows wil laten beheren, maar je ervoor kiest om die zelf te beheren in bijvoorbeeld een SQL server database. Als we bijvoorbeeld de authenticatie wel door Windows willen laten verzorgen maar de rollen in een eigen database willen beheren dan kunnen we dat doen zoals in afbeelding 8 wordt getoond.

Naast deze manieren om volledig je eigen authenticatie en autorisatie (rollenbeheer) te regelen, kan je ook gebruik maken van de faciliteit van COM+ services. Deze diensten zijn onder .NET beschikbaar in de namespace System.EnterpriseServices door je component af te leiden van het ServicedComponent type. Je bent dan beperkter in je mogelijkheden maar hebt wel een kant-en-klare oplossing.

Verder is het goed om je te realiseren dat de principal aan de thread hangt binnen het huidige appdomain. Wanneer je dus om wat voor reden dan ook een appdomain-grens passeert gaat de principal verloren. In de huidige versie van .NET vindt er immers geen contextflow plaats zoals we die in DCOM kennen. Een typisch geval hiervan kom je tegen wanneer je .NET remoting toepast. Nu zal je dus zelf je principal moeten serializeren en opnieuw construeren aan de ontvangende kant.



Afbeelding 9. Configuratiebeheer van policies

.NET Code access security

Naast het afschermen van code voor ongeautoriseerde gebruikers willen we ook op een lager niveau ons systeem afschermen tegen onbetrouwbare code. In een component based-ontwerp zien we steeds vaker dat eigen code wordt gecombineerd met componenten van derde partijen. In deze situatie wil je expliciete rechten (permissions genoemd) kunnen uitdelen aan (groepen van) code. In de System.Security.Permissions namespace vind je standaardimplementaties voor onder meer FileIOPermission, RegistryPermission en ReflectionPermission. Code access security is een fundamentele toevoeging van bestaande technieken zoals authenticode. Authenticode heeft als zwakte dat het is gebaseerd op vertrouwen: we kunnen met zekerheid bepalen van wie de code afkomstig is en daarom vertrouwen we de code. Wanneer deze code echter een fout bevat of foutief wordt gebruikt, is er vrijwel niets dat ons hiertegen kan beschermen. We zullen hierna laten zien dat code access security ons juist in deze gevallen wel bescherming kan bieden. Eerst kijken we hoe we permissies kunnen toekennen aan een assembly en vervolgens hoe we deze kunnen gebruiken in een assembly.

```
WindowsPrincipal currentPrincipal = new
  WindowsPrincipal(WindowsIdentity.GetCurrent());
```

Afbeelding 7.

```
// bepaal de identity van huidige windows gebruiker
WindowsIdentity currentUser = WindowsIdentity.GetCurrent();

// haal rollen op uit database
string[] rollen = HaalRollenVanGebruiker(currentUser.Name);

// construeer principal op basis van huidige Windows
```

Afbeelding 8.

Permissies toekennen aan een assembly

Een assembly mag je pas naar file schrijven wanneer het een FileIOPermission bezit. Met configuratie kan je eenvoudig rechten uitdelen aan assemblies. Om het beheer te vereenvoudigen verbind je permissions niet direct aan een assembly maar gaat dat via het model in afbeelding 9.

Wanneer de assembly wordt geladen in het hosting process is het de taak van de host om het bijbehorende evidence (bewijsmateriaal) te verzamelen en aan de assembly te koppelen. Evidence is een intrinsieke eigenschap van een assembly die iets vertelt over de identiteit. Voorbeelden van evidence zijn een hash of een public key waarmee hij gesigned is. Vervolgens wordt het verzamelde evidence aangeboden aan de CLR security manager om de permissies (rechten) van de assembly te bepalen. Met reflectie kan je uitvragen wat het verzamelde evidence is van een assembly. In afbeelding 10 is daar een voorbeeld van gegeven.

De output van de code uit het voorbeeld van afbeelding 10 is weergegeven[2] in

2 Om de leesbaarheid te vergroten is de hash-waarde ingekort.

```
using System.Reflection;

Assembly myAssembly = Assembly.GetExecutingAssembly();

foreach(object evidence in  myAssembly.Evidence)
{
Console.WriteLine(evidence);
}
```

Afbeelding 10.

```
<System.Security.Policy.Zone version="1">
   <Zone>MyComputer</Zone>
</System.Security.Policy.Zone>

<System.Security.Policy.Url version="1">
   <Url>file://D:/Personal/Software Development/Microsoft.NET/Secu
rity/WriteToFile/bin/Debug/WriteToFile.exe</Url>
</System.Security.Policy.Url>

<StrongName version="1"
    Key="0024000004800000940000000602000000240000525341310004000001000100FF854B189D3BC3DB8225158793BA854A0BE71149CDEE74EF8A5ADCE8FBD1539599297CC6306A7A16077B6FAF8941CC207124C561A8E4C2184CFA9812CA07D6FA4D059DF615644117CF371316D5B9E0DE043C6EA37868674C71F3DE76BF56C6EA423F8D3B3696BF356E90F99037EFC9C6C24B9EABD7BAF3AE23F13DC371814FC6"
    Name="WriteToFile"
    Version="1.0.0.1"/>

<System.Security.Policy.Hash version="1">
<RawData>4D5A90000300000004000000FFFF0000B800000000000000400000000000000000000000000000000000000000000000000000000000000000800000000E1FBA0E00B409CD21B8014CCD21546869732070726F6772616D2063616E6E6F742062652072756E20696E20444F53206D6F
…
</RawData>
</System.Security.Policy.Hash>
```

Afbeelding 11.

afbeelding 11. Er is te zien dat de assembly vanaf het locale systeem (MyComputer) is geladen (Zone policy). Daarnaast valt op dat deze assembly een public key bevat en dus is voorzien van een strongname.

Een assembly kan tot meerdere code groepen behoren. Op basis van het evidence wordt bepaald tot welke codegroepen een assembly behoort. Codegroepen zijn hiërarchisch geordend. Een assembly behoort tot een bepaalde codegroep als hij voldoet aan de bijbehorende membership-conditie. Een voorbeeld van een membership-conditie is dat de assembly tot een bepaalde zone behoort, of dat hij een bepaalde public key heeft.

Om te bepalen tot welke codegroepen een assembly behoort, checkt men het verzamelde evidence bij de codegroepen. Dit gebeurt in dfs-order (depth-first-search). Eerst wordt gecontroleerd of de assembly evidence heeft die voldoet aan de bovenste codegroep met de naam all_code. Als de membership-conditie voldoet worden de bijbehorende permission sets (en dus indirect de bijbehorende permissions) toegewezen aan de assembly, en wordt de zoektocht voortgezet bij de eerste kind-code groep. Opnieuw kent men hier de permission sets toe (vereniging) als de membership-conditie voldoet.

Als de membership conditie niet voldoet worden er geen permission sets toegekend en wordt er niet dieper in de boom gezocht maar wordt zijn sibling verder gecontroleerd. Door codegroepen onder elkaar te plaatsen (vader-kind) relatie kan er dus een boolean AND worden gemodelleerd terwijl je siblings (broer/zus relaties) kan gebruiken om een OR te modelleren.

De derde dimensie van het model in afbeelding 9 is het security-niveau. Er zijn drie niveaus: enterprise, machine en user. De geschetste code-groepen en permission sets, en de relaties daartussen, kun je dus op drie niveaus definiëren. Op al deze niveaus wordt bovengenoemd algoritme toegepast om de permissies van een assembly te bepalen per niveau. De uiteindelijke set van toegekende permissies is dan de doorsnede hiervan.

Om tijdens design-time te bepalen welke code groepen er worden toegekend aan een bepaalde assembly op basis van het verzamelde evidence kun je de command-line tool CASPOL[3] gebruiken. In afbeelding 12 zie je per niveau tot welke codegroepen bovengenoemde assembly behoort via het statement caspol –rsg (resolve groups). Op zowel enterprise als user level zien we dat er geen enkele restricties gelden (full-trust). In een standaardconfiguratie wordt alles namelijk afgedwongen op machine niveau.

Nu we weten tot welke codegroepen een assembly behoort, kunnen we vervolgens de totale set van toegekende permissies van de assembly bepalen. Om dit met CASPOL te doen gebruik je het statement caspol –rsp (resolve permissions).

Stel dat je wilt definiëren dat geen enkel programma op een machine een verbinding met het web (WebPermission) mag maken behalve de lokaal geïnstalleerde

3 CASPOL staat voor Code Access Security POLicy editor.

```
Microsoft (R) .NET Framework CasPol 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.

Level = Enterprise

Code Groups:
1.  All code: FullTrust
Level = Machine


Code Groups:
1.  All code: Nothing

   1.1.  Zone - MyComputer: FullTrust
Level = User


Code Groups:
1.  All code: FullTrust
Success
```

Afbeelding 12.

software die afkomstig is van een bepaalde fictieve leverancier MyCompany.NET[4]:

1. Maak een kopie van de standaard permission set Everything en noem deze EverythingButTheWeb. (We maken eerst een kopie zodat we het origineel behouden.);
2. verwijder de permissie web access (System.Net.WebPermission) uit de set EverythingButTheWeb;
3. wijzig op machineniveau de permissionset van code groep 1.1 Zone My Computer van fulltrust in EverythingButTheWeb. (Hierdoor kan geen enkele locale applicatie meer toegang tot het web krijgen.);
4. creëer een permission set ConnectToWeb, met daarin alleen een web access permission voor de juiste URL's;
5. voeg een codegroep MyCompanySoftware toe direct onder de code groep uit stap 3 met een publisher membership op basis van de public key van MyCompanySoftware;
6. koppel de permission set ConnectToWeb aan de code groep MyCompanySoftware. (Hierdoor krijgt de locaal geïnstalleerde software van MyCompany toegang tot het web.)

Permissies gebruiken in een assembly

Hiervoor heb ik geschreven over hoe je permissies toekent aan je code. Ook heb ik in een voorbeeld de software van MyCompany.NET het recht gegeven om een verbinding met het web te maken. Stel dat MyCompany.NET een zelfontwikkelde assembly ConnectToWeb.dll heeft gemaakt om de toegang tot het web te regelen. Wanneer we geen code access security zouden hebben, zou iedereen nu deze dll kunnen gebruiken om toegang tot het web te krijgen. Met code access security is dat echter niet mogelijk omdat bij het controleren van een permissie niet alleen naar huidige assembly wordt gekeken maar ook naar alle voorliggende aanroepers. Kortom, er vindt een stack walk plaats waarin wordt gekeken of alle aanroepers ook in bezit zijn van de gevraagde permissie. In dit geval, waarin we een webservice aanroepen, wordt door het framework een Demand gedaan op de System.NET.WebPermission. Doordat andere applicaties niet beschikken over de juiste private key van MyCompany.NET is de ConnectToWeb.dll onbruikbaar voor derden.
Naast de bestaande permissions is het ook mogelijk om je eigen permissies te ontwikkelen. Zie hiervoor de IPermission interface in de System.Security namespace. Als laatste is nog op te merken dat het een goede manier is om vooraf in je code aan te geven wat de minimale permissies zijn die je assembly nodig heeft om te kunnen draaien. Dit kan je weergeven in code die staat in afbeelding 13.
Het voordeel van deze benadering is dat de loader dit controleert met de gekregen permissies en automatisch het laden stopt als de minimale permissies niet via policies worden toegewezen. Hierdoor blijft veel exception handling code bespaard. Een bijkomend voordeel is dat van buitenaf ook te zien is welke permissies je code nodig heeft om goed te kunnen draaien. Dit kan een administrator zien door via de command line te vragen: permview ConnecToWeb.dll

```
[assembly: WebPermission(SecurityAction.RequestMinimum, Connect =
  @"http://MyCompany01/datastorage/storedata.asmx")]
```

Afbeelding 13.

Complete en open set van security-mechanismen

Door de nieuwe trends op het gebied van software-ontwikkeling waarin systemen steeds vaker en hechter geïntegreerd worden met internet is het noodzakelijk om opnieuw na te denken over beveiliging van systemen. Het .NET framework biedt een zeer complete en open set van security-mechanismen om op een veilige manier gebruik te maken van deze nieuwe mogelijkheden.

Nuttige internetadressen

- http://msdn.microsoft.com/msdnmag/issues/02/09/SecurityinNET/default.aspx
- http://msdn.microsoft.com/library/en-us/dnnetsec/ html/netframesecover.asp
- http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcodeaccesssecurity.asp
- http://msdn.microsoft.com/library/en-us/cpguide/html/cpconrole-basedsecurity.asp

4 We gaan er in dit geval vanuit dat de code van MyCompany Software gesigneerd is met een certificaat via de .NET framework utility (`makecert` en) `signcode`.