Marcel de Vries is werkzaam bij Info Support.

Applicatieautorisatie met Microsoft Authorization Manager

HET SCHRIJVEN VAN EEN AUTORISATIE-FRAMEWORK

Vraag een tiental ontwikkelaars een lijst met problemen waarvoor ze al verscheidene malen een oplossing hebben geschreven en hierop zal zeker de term applicatieautorisatie voorkomen. Het Windows besturingsysteem levert al sinds jaar en dag een mechanisme om besturingsysteem resources te autoriseren. Dit gebeurt op basis van het controleren van Access control lists (ACL) tegen een access token dat de gebruiker krijgt toegewezen tijdens het inloggen op de machine.

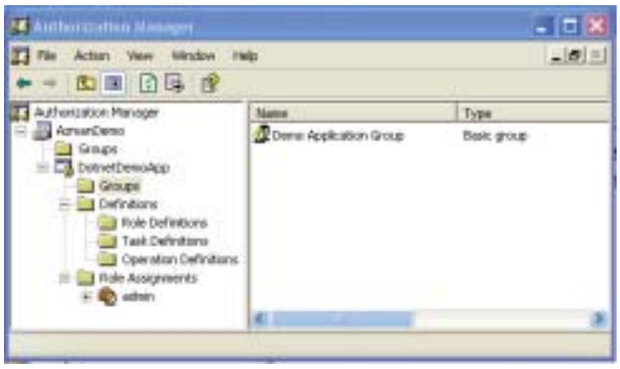
ACL's worden op systeemresources geplaatst zoals een file, share of registry key. Na controle van het access token tegen een ACL levert dat dan soms de wel bekende melding "Access denied" op. Dit gebeurt indien uit het access token blijkt dat de gebruiker niet voldoende rechten heeft om de resource te gebruiken. Bij het autoriseren van gebruikers in een applicatie kun je ook gebruik maken van deze Windows-autorisatie als het gaat om door het besturingsysteem beheerde resources. Binnen een applicatie heb je echter meestal te maken met resources die virtueel zijn. Resources die moeten worden afgeschermd zijn bijvoorbeeld schermen, menuopties, knoppen of zelfs (web)services die worden aangeboden door de applicatie. Dit soort resources worden door het besturingsysteem niet beschermd en hiervoor zal een applicatieprogrammeur dus zelf een oplossing moeten bedenken.

Beheerproblematiek

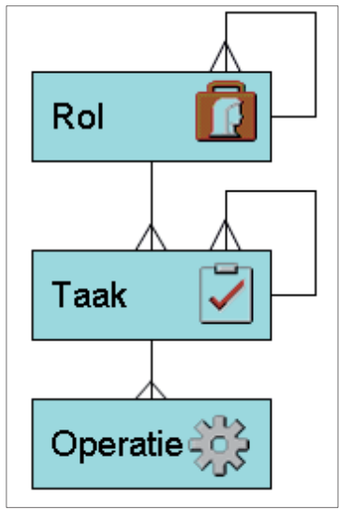
Doordat er tot voor kort geen standaard oplossing beschikbaar was voor applicatieautorisatie is er in de loop der jaren in organisaties een grote hoeveelheid losse applicatiebeheertools ontstaan. Er is dus niet alleen een probleem aangaande applicatieonderhoud voor de programmeur, maar ook een probleem voor de beheerder die voor iedere applicatie een eigen beheertool heeft gekregen. Om aan deze problematiek een einde te maken heeft Microsoft in Windows Server 2003 een generieke oplossing meegeleverd onder de naam Authorization Manager, kortweg AzMan. AzMan biedt de applicatieprogrammeur een generieke API voor het uitvragen van autorisaties op rol-, taak- en operatieniveau. De beheerder heeft daarbij een generieke beheerapplicatie gekregen in de vorm van een managementconsole snap-in. De snap-in kan in twee modi worden gebruikt, te weten: developer mode en adminstration mode. In afbeelding 1 is een screenshot te zien van de managementconsole snap-in die in administration mode draait. In deze mode kan de beheerder alleen taken en rollen aanmaken en hieraan Windows-gebruikers of groepen toekennen. Tevens heeft de beheerder de optie om zelf application-groups te definiëren. Application-groups zijn virtuele groepen die gebruikers en groepen uit de Active Directory samenvoegt. Op die manier kan een beheerder eenvoudig de groep van gebruikers samenstellen voor een specifieke applicatie zonder dat hij hiervoor zijn accounts-database moet gaan aanpassen in de Active Directory.

Operaties, taken en rollen

Als je autorisatie dient af te dwingen in een applicatie moet je een keuze maken op welk niveau dit dient te gebeuren. Als je kijkt naar applicaties dan kun je deze bijvoorbeeld opsplitsen in operaties en taken. Een operatie is dan gelijk aan het uitvoeren van een methode of het versturen van een bericht. Een taak is dan een groep van operaties die dienen te worden uitgevoerd voor het



Afbeelding 1. MMC snap-in AzMan



Afbeelding 2. Relatie tussen rollen, taken en operaties

vervullen van een Use Case. Denk daarbij aan bijvoorbeeld het scherm voor het opvragen van klantgegevens of het opvoeren van een prospect. Bij het uitvoeren van de taken kan de gebruiker een bepaalde rol vervullen op basis waarvan hij of zij meer of minder rechten heeft. Een rol kan bijvoorbeeld 'beheerder' zijn of 'medewerker personeelszaken'. Afhankelijk van de rol is men geautoriseerd voor het uitvoeren van bepaalde taken. Als je nadenkt over de relatie tussen taken en rollen dan kun je in eenvoudige applicaties volstaan met een één-op-veel relatie tussen rol en taak. AzMan biedt alle niveaus van autorisatie. Hierbij gaat AzMan uit van het complexe scenario waarbij rollen bestaan uit rollen en taken, taken bestaan uit taken en operaties. Op deze manier kan een applicatiebeheerder de gewenste autorisatiestructuur aanmaken met alle mogelijke combinaties van rollen en taken. Operaties worden door de ontwikkelaar gedefinieerd en zijn vast. Afbeelding 2 geeft de relatie weer tussen rollen, taken en operaties binnen AzMan.

Authorization store

AzMan kan de autorisatie-informatie opslaan in twee typen authorization stores te weten: een XML-file of een Active Directory container. Het gebruik van Active Directory als authorization store heeft een aantal extra voordelen ten opzichte van het gebruik van een XML-file, te weten:

- Beter beveiligde opslag van autorisatiegegevens.
- Mogelijkheid om access checks te auditen via standaard windows security auditing.
- Gedelegeerd beheer.

Een belangrijke beperking in het gebruik van Active Directory als authorization store is dat Active Directory in Windows Server 2003 in functional mode moet draaien. Voor de ontwikkelaar en bedrijven die alleen een Windows 2000 Active Directory beschikbaar hebben, is er sinds kort wel een alternatief. Dit alternatief heet Active Directory Application Mode of kortweg ADAM. Je kunt van ADAM verscheidene instanties op dezelfde machine draaien en daarbij hoeft de machine ook geen DNS server te zijn. Hierdoor is ADAM ook een prima oplossing om in de ontwikkelomgeving een Active Directory te hebben voor autorisatie. Het verdient dan ook de aanbeveling om gebruik te maken van ADAM indien je applicatieautorisatie gaat toepassen op basis van AzMan.

.NET rolebased security

Het .Net Framework bied een role-based security-model en onderkent daarin twee typen objecten: de identity en de principal. De identity wordt bepaald op basis van de geauthenticeerde gebruiker. Het identity-object bevat onder ander gegevens over de user-naam en de wijze waarop hij is geauthenticeerd. De principal bevat informatie of de identity geautoriseerd is om bepaalde operaties uit te voeren op basis van de rol die hij heeft. Op het principal-object vind je een methode IsInRole() om uit te vragen of de huidige identity een bepaalde rol heeft. Het principal-object zal altijd terug te vinden zijn als contextinformatie bij het uitvoeren van een method call. In het geval van een ASP.NET-applicatie zal de principal terug te vinden zijn in de HTTPContext. In een windows forms-applicatie is de principal terug te vinden op de executerende thread. Het aanmaken en doorgeven van de principal wordt door het .NET Framework zelf geregeld. Het is ook mogelijk om zelf een eigen principal-object aan te maken en toe te kennen. Indien na authenticatie een principal-object wordt aangemaakt en wordt toegewezen aan de thread of de HTTPContext dan zal deze automatisch met de calls worden meegegeven. Het is op deze wijze heel goed mogelijk om zelf een principal-object aan te maken met daarin een eigen implementatie voor onder andere de IsInRole() methode.

Granulariteit van autorisatie

Bij het implementeren van autorisatie in een applicatie is het altijd een vraag op welk niveau je autorisatie wilt afdwingen. Mag een gebruiker een heel scherm niet zien, of maar enkele onderdelen?

Type autorisatie	Mogelijkheid in AzMan
Autorisatie of een gebruiker een scherm mag opstarten	Taakniveau
Autorisatie of een gebruiker een onderdeel van een scherm mag gebruiken, bijv een delete knop	Taak- en operatieniveau
Autorisatie of een gebruiker een deel van de geretourneerde gegevens mag inzien, bijvoorbeeld privacygevoelige informatie	Taak en rolniveau
Autorisatie of een businessservice mag worden aangeroepen	Operatieniveau

Tabel . Mogelijke autorisatiewensen

Mag een gebruiker bepaalde services wel of niet raadplegen? Indien je een generiek framework wilt hebben voor autorisatie, dien je ook op deze verschillende niveaus controles te kunnen uitvoeren. Alleen het kunnen controleren van een rol is in veel gevallen niet voldoende. Zoals eerder aangegeven heeft AzMan de notie van rollen, taken en operaties. Voor een generiek framework zou de combinatie van alle niveaus van autorisatie de meest flexibele oplossing bieden. In tabel 1 is een aantal mogelijke autorisatiewensen afgezet tegen de mogelijkheden binnen de authorization manager.

Om het verhaal niet te complex te maken zal in dit artikel alleen een implementatie te zien zijn die op basis van AzMan controles uitvoert op operatieniveau. Het implementeren van autorisatie op taak- en rolniveau is vervolgens zelf relatief eenvoudig te realiseren.

AzMan API's

AzMan wordt geleverd als een COM-component op Windows Server 2003 en is ook te gebruiken op Windows XP door het Windows Server 2003 administration pack te installeren. Na installatie zal de AzMan MMC snap-in beschikbaar zijn vanuit de administrative tools-folder en zijn de COM-componenten geregistreerd op het systeem. We beginnen met het maken van een eenvoudige implementatie, die alleen autorisatie controleert op operatieniveau, gedefinieerd in de AzMan. Om gebruik te maken van de AzMan API's dien je een referentie op te nemen naar de COM typelibrary: *"azroles 1.0 Type Library"*. Vervolgens kun je in de code de namespace AZRolesLib opnemen voor alle AzMan API-calls.
De voornaamste API's die we nodig hebben, zijn die voor het initialiseren van de authorization store en die voor het uitvragen van permissies op operatieniveau. Het doel van het te implementeren framework is het de applicatieprogrammeur zo eenvoudig mogelijk te maken om autorisatiecontroles te laten uitvoeren. De uiteindelijke aanroep vanuit de applicatie is als volgt:

```
Authorization.IsPermitted("Operatie naam").
```

Om dit te kunnen realiseren op een zo veilig mogelijke manier zullen we de volgende classes aanmaken:

- Authorization Class die de IsPermitted-methode implementeert. Deze class kan door de applicatie worden gebruikt voor het uitvoeren van de autorisatiecontroles.
- AzManAgent Class die een wrapper biedt om de AzMan API. Deze class wordt sealed internal, zodat deze alleen kan worden gebruikt vanuit de Authorization-class.
- OperationAuthorization Class die de implementatie biedt van de operatieautorisatie. Dit wordt een internal nested sealed class van AzManAgent. Ook deze class is alleen vanuit de Authorization-class te benaderen. Het enige entrypoint van de assembly wordt de Authorization-class.

We beginnen met het maken van een Authorization-class die de static methode IsPermitted() bevat. De implementatie zal deze call

```
public class Authorization
{
  // zorg er voor dat de constructor niet is aan te roepen. Het betreft
  // hier een class met alleen static methods
  private Authorization()
  {
  }
  public static bool IsPermitted(string operation)
  {
    // Haal de huidige windows identity op
    WindowsIdentity identity = WindowsIdentity.GetCurrent();
    Return AzManAgent.OperationAuthorization.IsPermitted(identity,operation);
  }
}
```

Codevoorbeeld 1

direct delegeren aan de class die de wrapper vormt naar AzMan genaamd AzManAgent. De implementatie van de autorisatie-class is weergegeven in codevoorbeeld 1.

Bij het aanroepen van de static methode op de internal class OperationAuthorization binnen AzManAgent-class wordt de operatie gevalideerd bij de AzMan store. Voordat deze store echter kan worden benaderd, dient deze eerst te worden geïnitialiseerd. Dit is mogelijk door het aanmaken van een instantie van de AuthorizationStoreClass. Op het object kun je vervolgens de methode aanroepen Initialize() en hierbij geef je onder andere de parameters op naar de authorization store (dit kan een XML-file of een Active Directory-container zijn) en de naam van de applicatie waar we de autorisaties voor nodig hebben. Alle controlefuncties van AzMan zijn beschikbaar binnen de context van een applicatie. Hiervoor wordt de class-variabele “application” van het type IAzApplication geïnitialiseerd, zodat deze in vervolgaanroepen beschikbaar is. De initialisatiecode is terug te vinden in codevoorbeeld 2.

De initialisatiecode uit codevoorbeeld 2 zal worden geplaatst in de constructor van de AzManAgent-class. Deze zal eenmalig worden uitgevoerd bij de allereerste aanroep op een van de static methodes.

Voor de implementatie van de IsPermitted-methode maken we gebruik van een nested internal sealed class met de naam operationAuthorization. Dit doen we omdat op deze manier de diverse niveaus van authorisatie netjes kunnen worden opgedeeld in de classes OperationAthorization, TaskAuthorization en eventueel RoleAuthorization. We implementeren nu echter alleen OperationAutorization. De OperationAuthorization-class is een internal sealed class zodat deze niet zomaar door overerving kan worden aangepast en niet van buiten de assembly kan worden aangeroepen. Het is vanuit een beveiligingsoogpunt een gewenste aanpak om de classes zo restrictief mogelijk te programmeren en inheritance te beperken. Zie voor security-richtlijnen ook het boek Writing Secure Code 2nd edition, ISBN 0-7356-1722-8. De implementatie van de OperationAuthorization-class is terug te vinden in codevoorbeeld 3.

```
const int defaultOpenFlags = 0;
      // Haal configuratie informatie uit de config file op
string authorizationStore =
       ConfigurationSettings.AppSettings.Get(“authorizationStore”);
string authorizationApplication =
       ConfigurationSettings.AppSettings.Get(“applicationName”);

      // Open de AzMan store.
      AzManAgent.store = new AzAuthorizationStoreClass();
      AzManAgent.store.Initialize(
                                  defaultOpenFlags,
                                  authorizationStore,
                                  null);
      // Open the AzMan application
      AzManAgent.application =
                             AzManAgent.store.OpenApplication(
      authorizationApplication,
      null);
```

Codevoorbeeld 2

Operatienaam en ID

Voor het opvragen of de gebruiker geautoriseerd is voor een operatie wil je gebruik kunnen maken van de naam van de operatie. In AzMan kun je echter alleen uitvragen of een gebruiker een operationID mag benaderen. Om op naam te kunnen valideren dienen we een lookup-table te implementeren waarbij de operatienaam naar OperationID kan worden vertaald. Om de lookup zo snel mogelijk te maken gebruiken we een hashtable die we initialiseren op de allereerste call naar de nested AzManAgent-class OperationAuthorization. In deze hashtable plaatsen we alle beschikbare operaties die in de authorization store aanwezig zijn, zodat deze op naam kunnen worden teruggevonden. De key in de hashtable is de naam van de operatie en de value is het operationID. Ook de code voor de initialisatie van de hashtable is terug te vinden in codevoorbeeld 3.

Om uit te vragen of de gebruiker geautoriseerd is voor een operatie dient een methode de werkelijke call naar AzMan uit te voeren. We noemen deze methode ook IsPermitted(). Voor de controle binnen AzMan wordt de methode AccessCheck gebruikt. Deze methode is beschikbaar op een zogenaamde ClientContext. Deze context dient eerst te worden geïnitialiseerd op basis van de huidige WindowsIdentity. De initialisatie wordt gedaan met de methode InitializeClientContextFromToken() op de eerder geïnitialiseerde static member van het type IAzApplication. Bij het uitvoeren van de AccessCheck-call is het mogelijk verscheidene operaties tegelijk te controleren. In onze implementatie maken we hier echter geen gebruik van en zullen we altijd maar één operatie per keer controleren. Door deze aanpak zal het return-array ook altijd maar één waarde bevatten. Verder wordt de naam van de methode die we controleren meegegeven voor auditing-doeleinden. Auditing kan worden aangezet indien gebruik gemaakt wordt van de Active Directory als authorization store. De resultaten van iedere AccessCheck zullen vervolgens in de security-log worden weggeschreven. Bij de aanroep van de AccessCheck-methode moeten we het operationID hebben in plaats van een naam van een operatie.

```
internal sealed class OperationAuthorization
{
  private static Hashtable operationIDs = null;

  static OperationAuthorization()
  {
   // maak een hashtable aan met de juiste grootte zodat resizing wordt
   // vermeden
   OperationAuthorization.operationIDs = new
                      Hashtable(AzManAgent.application.Operations.Count);

   foreach (IAzOperation operation in AzManAgent.application.Operations)
   {
     // De operation IDs (hashtable waarden) worden doorzocht op operatienaam
     // dit zijn dan ook de hashtable keys.
     OperationAuthorization.operationIDs.Add(operation.Name,
                       operation.OperationID);
   }
  }
}
```

Codevoorbeeld 3

```
internal static bool IsPermitted(WindowsIdentity identity,
                    string operationName)
{
  // Maak een nieuwe clientcontext aan voor de windows identity.
  IntPtr securityToken = identity.Token;
  IAzClientContext azContext =
       AzManAgent.application.InitializeClientContextFromToken(
            (ulong)securityToken.ToInt64(), null);
  int operationID =
          Convert.ToInt32(OperationAuthorization.operationIDs[operationName]);

  // Maak de parameters aan zoals AzMan deze verwacht.
  // Let op we dienen exact deze datatypen te gebruiken AzMan verwacht
  // een VARIANT met daarin een SafeArray met integer waarden.
  object[] operations = new object[1];
  operations[0] = operationID;
  // De eerste parameter(bstrObjectName) wordt door AzMan gebruikt voor auditing
  // daarom wordt deze ook megegeven als eerste argument
  object[] results = (object[])azContext.AccessCheck(operationName,
          null,operations, null, null, null, null, null);

  // We krijgen maar een resultaat terug omdat we ook maar een operatie
  // meegegeven
  return ((int)results[0] == 0);
}
```

Codevoorbeeld 4

Hiervoor gebruiken we de eerder geïnitialiseerde HashTable om de lookup te doen. De implementatie van de IsPermitted-methode op de OperationAuthorization-class is terug te vinden in codevoorbeeld 4.

De implementatie voor autorisatie op operatieniveau is nu gereed. We moeten nu alleen in de authorization manager de operaties en taken aanmaken die in de applicatie kunnen worden gecontroleerd. Hiervoor maak je gebruik van de authorization manager MMC snap-in.

Autorisatie-framework

In dit artikel hebben we een eerste aanzet gemaakt voor het schrijven van een autorisatie-framework met behulp van AzMan. Zeker niet alle mogelijkheden van AzMan zijn de revue gepasseerd. AzMan is standaard beschikbaar vanaf Windows 2000 servicepack 4 en een vast onderdeel van Windows Server 2003. Verder kun je het beheer van autorisaties ook op een Windows XP werkplek doen met behulp van het zogenaamde administration pack voor Windows Server 2003. In combinatie met bijvoorbeeld het product Active Directory Application Mode (ADAM) is het ontwikkelen van applicaties op basis van AzMan bijzonder goed mogelijk geworden voor de developer. Indien je gebruik wilt maken van een generiek autorisatie-framework dan kun je onder andere ook kijken naar reeds beschikbare implementaties. Er is onder meer een application block beschikbaar dat door Microsoft is gemaakt onder de naam “Authorization Application Block”. Als je daar gebruik van maakt, kies dan wel voor de AzMan provider en niet voor de tevens beschikbare SQL provider. Naast het application block is er ook een public workspace beschikbaar op de site www.gotdotnet.com. Deze workspace heeft de naam Enterprise Development Accelerator For .NET. In deze workspace is ook een autorisatie-frameworkimplementatie beschikbaar rondom AzMan.

Marcel de Vries is werkzaam bij Info Support.