WCF claims-based authorization

Auteur
Edwin van Wijk
Info Support
edwinw@infosupport.nl

Publicatie
05-03-2008
SDN Magazine

Samenvatting
Met .Net 3.0 hebben we WCF beschikbaar gekregen voor het bouwen van gedistribueerde applicaties. Daarnaast is ook het zogenaamde IdentityModel beschikbaar gekomen (System.IdentityModel). In dit artikel beschrijf ik hoe we met behulp van dit IdentityModel gebruikers van WCF services kunnen autoriseren op basis van zogenaamde claims.

In elk project waar ik tot nu toe aan heb meegewerkt is beveiliging altijd een belangrijk onderdeel gebleken. De vraag is vaak wat de meest geschikte manier van authenticeren en autoriseren van de gebruikers is voor de specifieke situatie bij de opdrachtgever. Hierbij speelt ook de infrastructuur waarop een oplossing uiteindelijk moet worden uitgerold een grote rol.
Nu biedt WCF ons voor het authenticeren en autoriseren van gebruikers een aantal mogelijkheden. Wat ik in dit artikel specifiek wil behandelen is het autoriseren van gebruikers van WCF services door gebruik te maken van claims die onderdeel zijn van het zogenaamde IdentityModel (System.IdentityModel) dat binnen .Net 3.0 beschikbaar is. Ik ga er van uit dat de lezer bekend is met de basisprincipes van WCF.

ServiceAuthorizationManager

Gebruikers worden standaard binnen WCF alleen geauthenticeerd (afhankelijk van de binding) maar niet geautoriseerd bij het uitvoeren van serviceoperaties. WCF biedt ons echter wel een elegant mechanisme om autorisatiegedrag toe te voegen aan een service, zonder dat dit impact heeft op de implementatie van de service zelf. Hiervoor kunnen we zogenaamd servicebehavior inzetten. Een servicebehavior is een klasse die een bepaald gedrag vertegenwoordigt en kan worden geregistreerd bij een servicehost. Zodra de host wordt gestart zal een WCF Runtime worden opgebouwd (op basis van de configuratie) en het gedrag zal een onderdeel worden van deze runtime. Afhankelijk van het soort behavior dat is geregistreerd zal de WCF runtime er voor zorgen dat het behavior op de juiste momenten gedurende de aanroep van een serviceoperatie wordt aangeroepen zodat het zijn werk kan doen. WCF biedt zelf een aantal standaard behaviors, maar we kunnen ook zelf behaviors maken. Om autorisatiegedrag toe te voegen aan een service is binnen WCF het ServiceAuthorizationBehavior beschikbaar. Bij dit behavior moet een authorizationmanager gespecificeerd worden. Dit is een klasse die we zelf implementeren en moet afleiden van ServiceAuthorizationManager (zie listing 1). Als het ServiceAuthorizationBehavior is geregistreerd voor een service zal het onderdeel worden van de WCF runtime die wordt opgebouwd voor het hosten van de service (zie figuur 1). Het registreren van dit behavior kan m.b.v. een attribuut op de service, in code bij het starten van de host (zie listing 2) of in de configuratie van de host (zie figuur 2). Wanneer we een ServiceAuthorizationBehavior hebben geregistreerd bij een service, zal de WCF runtime op de authorizationmanager die is geconfigureerd, de methode CheckAccessCore aanroepen voor elke SOAP action die wordt uitgevoerd op de service. Het resultaat van deze methode is een boolean die aangeeft of de methode behorende bij de SOAP action mag worden uitgevoerd (true) of niet (false). In het laatste geval zal op de client een SecurityAccessDeniedException optreden. Door op onze authorizationmanager een override te maken van de methode CheckAccessCore kunnen we zelf bepalen hoe we gebruikers gaan autoriseren. Denk bijvoorbeeld aan een AzMan store waarin voor elke gebruiker is vastgelegd welke operaties zijn toegestaan. Onze authorizationmanager kan dan op basis van de Windows identity van de gebruiker (uitgaande van het feit dat de gebruiker op basis van zijn of haar Windows account is geauthenticeerd) achterhalen of de operatie mag worden uitgevoerd. Het is echter wel zo dat wanneer later een andere manier van autoriseren moet worden ingezet (waarbij bijvoorbeeld een database wordt gebruikt voor het vastleggen van de autorisatie), we onze authorizationmanager moeten aanpassen. Daarnaast kan het wellicht voorkomen dat meerdere bronnen moeten worden geraadpleegd om de autorisatie van de gebruiker te kunnen bepalen. Mede om dergelijke scenario's mogelijk te maken is binnen WCF de mogelijkheid gecreëerd om het bepalen van de kenmerken die een gebruiker heeft en het uiteindelijk autoriseren van de gebruiker op basis van deze kenmerken los te koppelen. Het is

namelijk mogelijk om bij een authorizationmanager één of meerdere zogenaamde AuthorizationPolicies te registreren.

AuthorizationPolicies

AuthorizationPolicies hebben als doel het bepalen van de kenmerken die een gebruiker van een service binnen een systeem heeft en deze vervolgens toe te voegen aan een zogenaamde evaluationcontext. Deze kenmerken zijn vervolgens gedurende de gehele serviceaanroep beschikbaar (hier kom ik later op terug). Een authorizationpolicy moet een implementatie leveren van de IAuthorizationPolicy interface (zie listing 3). De interface bevindt zich in de namespace System.IdentityModel. De Id-property moet een unieke string retourneren. De Issuer-property kan gebruikt worden om de uitgever van bepaalde kenmerken te identificeren. In de Evaluate-method wordt bepaald welke kenmerken een gebruiker heeft en deze worden vervolgens toegevoegd aan de evaluationcontext die als argument aan de methode wordt meegegeven. Ook kan een authorizationpolicy bepaalde reeds aanwezige kenmerken mappen op andere, die door het systeem kunnen worden gebruikt. Denk bijvoorbeeld aan het mappen van de identiteit van de gebruiker naar een set met operaties die deze gebruiker mag uitvoeren (het recht om iets uit te mogen voeren is uiteraard ook een kenmerk). Als we even terugdenken aan de authorizationmanager uit de vorige paragraaf die we zelf moeten implementeren: deze hoeft dan niet meer m.b.v. de identiteit van de gebruiker te bepalen of een operatie mag worden uitgevoerd, maar hoeft alleen te controleren of een bepaald kenmerk in de autorisatiecontext aanwezig is. Een extra voordeel is hierbij tevens dat de authorizationmanager onafhankelijk is geworden van de manier waarop de gebruiker is geauthenticeerd (Windows, username-password, certificaat, enz.). Dit wordt namelijk door een authorizationpolicy afgehandeld. De boolean return waarde van de Evaluate methode geeft aan of een policy genoeg informatie had om de juiste kenmerken uit te delen (true) of niet (false). Dit is handig wanneer een policy bijvoorbeeld bepaalde kenmerken nodig heeft die door een andere policy moeten worden uitgereikt. Als meerdere policies zijn geregistreerd en een bepaalde policy retourneert false, zal nadat de andere policies zijn aangeroepen deze policy opnieuw worden aangeroepen. Dit stopt zodra de desbetreffende policy uiteindelijk true retourneert of er geen overige policies meer zijn die nog moeten worden uitgevoerd. Er wordt aan de Evaluate methode ook een state argument (van het type object) meegegeven (by reference). Hierin kan een policy gegevens stoppen die dan bij een volgende aanroep weer kunnen worden gebruikt. Het registreren van authorizationpolicies kan in code bij het starten van de servicehost (zie listing 4) of in de configuratie van de servicehost (zie figuur 3).

Claims

We hebben gezien dat authorizationpolicies bepaalde kenmerken toevoegen aan de evaluationcontext, die door de authorizationmanager kunnen worden gebruikt om een gebruiker te autoriseren. Er moet dus een uniforme manier zijn om deze kenmerken aan te kunnen leveren. Binnen het IdentityModel zijn voor dit doel de zogenaamd Claims bedacht (zie listing 5). Claims kunnen worden gebruikt om een bepaalde kenmerken van een eindgebruiker te beschrijven. Omdat een bepaald recht om iets binnen een systeem te mogen doen ook een kenmerk van een gebruiker is, kan een claim gebruikt worden om de gebruiker te autoriseren. Een claim bevat altijd de volgende informatie:

- Het type van de claim.
- Een resource, de waarde waarover de claim iets zegt.
- Het recht dat de claim vertegenwoordigd m.b.t. de resource.

Standaard biedt het IdentityModel een aantal typen claims om verschillende zaken te kunnen beschrijven. Sommige van die claims worden bij het uitvoeren van een WCF serviceoperatie automatisch gevuld. Een voorbeeld van een dergelijke standaard claim is de zogenaamde ‘Name’ claim, die de naam beschrijft van het account waarmee een service is aangeroepen:

ClaimType : http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name
Resource : edwinw

Right **:** http://schemas.xmlsoap.org/ws/2005/05/identity/right/Identity

Deze claim wordt - afhankelijk van de gebruikte binding - standaard toegevoegd aan de evaluationcontext. Voor claimtype en right zijn binnen WCF constanten gedefinieerd. Zo zijn voor het claimtype en right in het bovenstaande voorbeeld respectievelijk de constanten ClaimTypes.Name en Rights.Identity gedefinieerd. Deze constanten kunnen worden gebruikt bij het creëren van een instantie van een standaard claim. Daarnaast bevat de claim klasse ook een lijst met static methoden die kunnen worden gebruikt om instanties van standaard claims te creëren. Deze lijst met methoden geeft een beeld van de verschillende typen claims die standaard binnen het IdentityModel zijn gedefinieerd: CreateDenyOnlyWindowsSidClaim, CreateDnsClaim, CreateHashClaim, CreateMailAddressClaim, CreateNameClaim, CreateRsaClaim, CreateSpnClaim, CreateThumbprintClaim, CreateUpnClaim, CreateUriClaim, CreateWindowsSidClaim en CreateX500DistinguishedNameClaim. De MSDN documentatie voor deze methoden bevat meer informatie over de verschillende standaard claimtypen. In listing 6 is te zien hoe de eerder beschreven Name claim kan worden gecreëerd door gebruik te maken van een static methode op de claim klasse. De resource-property van een claim is van het type object en kan van alles bevatten. Het is mogelijk om zelf een nieuw type claim te bedenken om bijvoorbeeld te beschrijven dat een persoon een bepaald bestand mag lezen:

ClaimType : urn:edwinw:claimtypes:fileaccess
Resource : c:\Documents\sales.xls
Right : urn:edwinw:claimrights:read

Zoals gezegd zijn de uitgereikte claims gedurende een complete serviceaanroep beschikbaar. Ze kunnen worden verkregen via de operationcontext (zie listing 7). Dat betekent dat de authorizationmanager dus niet de enige is die de claims kan gebruiken. In de implementatie van de service kunnen de claims dus ook gebruikt worden om bijvoorbeeld bepaalde bedrijfsregels af te dwingen. Denk bijvoorbeeld aan een internet applicatie waar klanten van een verzekeringsmaatschappij online verzekeringen kunnen aanschaffen. Een bedrijfsregel die hierbij van toepassing zou kunnen zijn is: "voor verzekeringsproduct X mag de klant niet ouder zijn dan 50 jaar". Wanneer de gebruiker wordt geautoriseerd door de applicatie zou een claim kunnen worden uitgereikt die de geboortedatum van de gebruiker beschrijft:

ClaimType **:** http://insuricorp.com/claimtypes/geboortedatum
Resource **:** 10-03-1975
Right **:** http://schemas.xmlsoap.org/ws/2005/05/identity/right/PossessProperty

Het voordeel is hierbij dat de code die de bedrijfsregel controleert niet op de hoogte hoeft te zijn van de manier waarop de gegevens van de gebruiker zijn opgeslagen. Dit wordt namelijk bij het autoriseren van de gebruiker door een authorizationpolicy bepaald en deze kent de bovenstaande claim toe. Belangrijk om hierbij – mede vanuit security oogpunt - expliciet te vermelden is dat alleen authorizationpolicies die zijn geregistreerd bij de authorizationmanager van een service de set met claims kunnen wijzigen. Daarbuiten kan alleen een read-only kopie worden verkregen.

Autorisatie op basis van claims

Zoals gezegd kunnen naast de claims die standaard worden meegeleverd met het IdentityModel ook eigen claims worden bedacht. Dit artikel behandelt het autoriseren van gebruikers op basis van claims. Voor dit doel bedenken we zelf een claim die het recht beschrijft om een bepaalde serviceoperatie (geïdentificeerd door de bijbehorende SOAP action) te mogen uitvoeren. Hier volgt een voorbeeld:

ClaimType **:** urn:edwinw:claims:authorization
Resource **:** urn:edwinw:customerservice:getcustomer
Right **:** urn:edwinw:claims:executeoperation

Als een gebruiker deze claim uitgereikt krijgt van een authorizationpolicy, mag deze de operatie 'GetCustomer' van de 'CustomerService' uitvoeren (voor deze methode is in het

operationcontract als SOAP action de URN gespecificeerd die als resource in de claim aanwezig is (zie listing 8)). Hoe de bovenstaande claim kan worden toegekend aan de gebruiker is te zien in listing 9. Wanneer op de service de operatie 'GetCustomer' wordt aangeroepen, zal de authorizationmanager - voordat de implementatie van de methode wordt aangeroepen – controleren of de bovenstaande claim is toegevoegd aan de authorizationcontext. Is dit het geval, dan mag de operatie uitgevoerd worden, anders niet. In listing 10 is te zien hoe een dergelijke controle uitgevoerd kan worden.

Federated security

Dit artikel beschrijft het gebruik van claims om de autorisatie te regelen voor .Net WCF services. Omdat claims zijn gebaseerd op open standaarden worden ze echter ook ingezet om federated security scenario's mogelijk te maken. Dit wil zeggen dat partijen uit verschillende beveilingsdomeinen, gebruikers van elkaar kunnen authenticeren en autoriseren op basis van claims. Mede vanwege de omvang van dit onderwerp op zich behandel ik het niet in dit artikel. Toch wil ik het wel even expliciet onder de aandacht brengen, omdat we mijns inziens in de toekomst steeds vaker zullen zien dat organisaties hun IT infrastructuur willen integreren zonder direct een trust relatie te moeten (of kunnen) leggen tussen elkaars beveiligingsdomeinen. Federated security kan hier een oplossing voor bieden.

Conclusie

Claims kunnen worden ingezet om op een uniforme manier bepaalde kenmerken en rechten van gebruikers te beschrijven. Dit stelt ons onder andere in staat om op een elegante manier het autoriseren van gebruikers van WCF services op basis van bepaalde rechten los te koppelen van het bepalen van deze rechten. Dit heeft als voordeel dat de authorizationmanager - die verantwoordelijk is voor het daadwerkelijk autoriseren van een gebruiker – onafhankelijk is van de manier van authenticeren van de gebruiker en de manier waarop de autorisatiegegevens zijn vastgelegd.

```
public class ServiceAuthorizationManager
{
  protected virtual bool CheckAccessCore(
    OperationContext operationContext);
    ...
}
```
Listing 1: System.ServiceModel.ServiceAuthorizationManager base-class.

```
ServiceHost myServiceHost =
  new ServiceHost(typeof(CustomerService));

myServiceHost.Authorization.ServiceAuthorizationManager =
  new ClaimsBasedSecurity.AuthorizationManager();
```
Listing 2: Registreren van een authorizationmanager bij de een servicehost in code

```
public interface IAuthorizationPolicy
  : IAuthorizationComponent
{
  string Id { get; }
  ClaimSet Issuer { get; }
  bool Evaluate(EvaluationContext evaluationContext,
    ref object state);
}
```
Listing 3: System.IdentityModel.IAuthorizationPolicy interface

```
List<IAuthorizationPolicy> policies =
  new List<IAuthorizationPolicy>();

policies.Add(
  new ClaimsBasedSecurity.XmlAuthorizationPolicy());

myServiceHost.Authorization.ExternalAuthorizationPolicies =
  policies.AsReadOnly();
```
Listing 4: Registreren van een custom AuthorizationPolicy in code

```
public class Claim
{
  public string ClaimType { get; }
  public object Resource { get; }
  public string Right { get; }
  …
}
```
Listing 5: De System.IdentityModel.Claim klasse

```
Claim nameClaim = Claim.CreateNameClaim("edwinw");
```
Listing 6: Creatie van een Name claim

```
OperationContext opCtx =
  OperationContext.Current;

AuthorizationContext azCtx =
  opCtx.ServiceSecurityContext.AuthorizationContext;

ClaimSet claims = azCtx.ClaimSets;
```
Listing 7: Opvragen van de claims binnen een operationcontext

```
[ServiceContract(Namespace = "urn:edwinw:customerservice")]
public interface ICustomerService
{
  [OperationContract(Action = "urn:edwinw:customerservice:getcustomer")]
  Customer GetCustomer(long customerNumber);

  [OperationContract(Action = "urn:edwinw:customerservice:addcustomer")]
  int AddCustomer(Customer customer);

  [OperationContract(Action = "urn:edwinw:customerservice:deletecustomer")]
  void DeleteCustomer(long customerNumber);
}
```

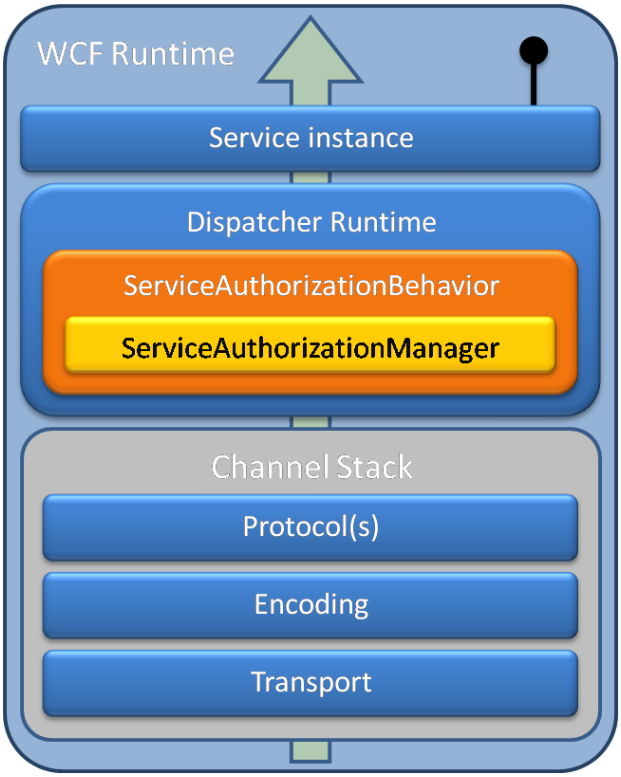
Listing 8: Servicecontract van de CustomerService

```
List<Claim> claims = new List<Claim>();

Claim claim = new Claim(
  // claimtype
  "urn:edwinw:claims:authorization",
  // resource
  "urn:edwinw:customerservice:getcustomer",
  // right
  "urn:edwinw:claims:executeoperation");

claims.Add(claim);

ClaimSet claimSet =
  new DefaultClaimSet(this.Issuer, claims);

evaluationContext.AddClaimSet(this, claimSet);
```

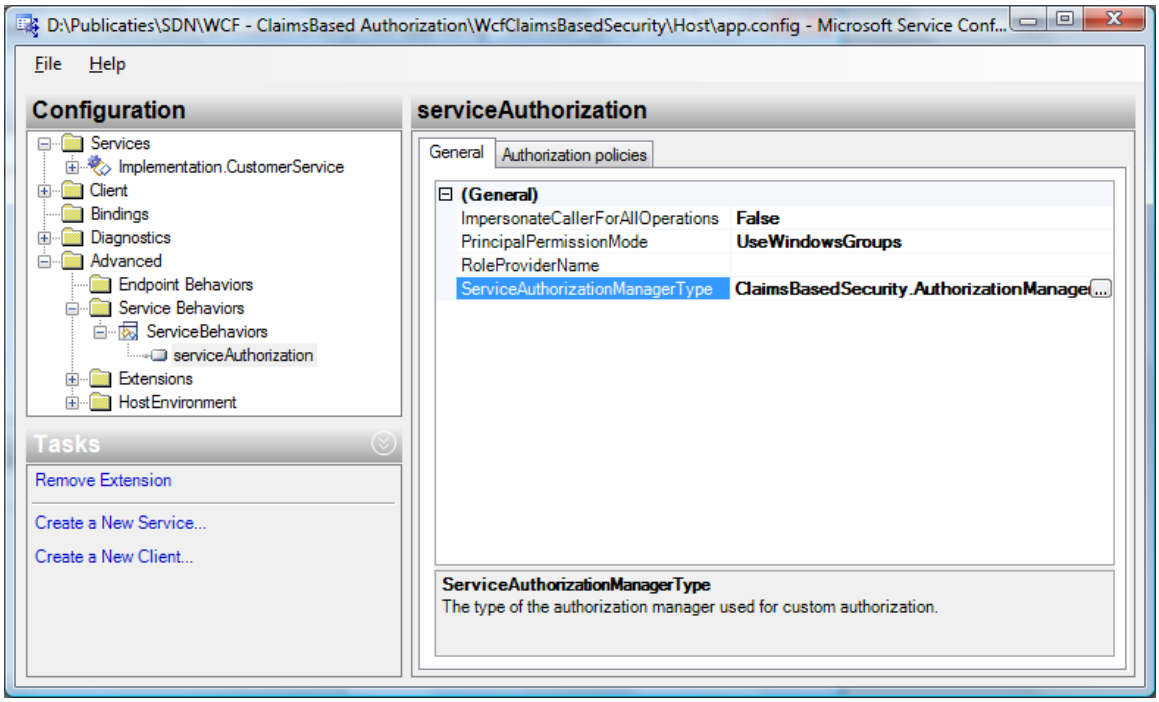
Listing 9: Het toekennen van een claim

```
string operationAction =
  operationContext.IncomingMessageHeaders.Action;

Claim requiredClaim = new Claim(
  // claimtype
  "urn:edwinw:claims:authorization",
  // resource
  operationAction,
  // right
  "urn:edwinw:claims:executeoperation");

OperationContext opCtx =
  OperationContext.Current;

AuthorizationContext azCtx =
  opCtx.ServiceSecurityContext.AuthorizationContext;

foreach (ClaimSet claimSet in azCtx.ClaimSets)
{
  if (claimSet.ContainsClaim(requiredClaim))
  {
    // gebruiker is geautoriseerd
    return true;
  }
}

// gebruiker is NIET geautoriseerd
return false;
```

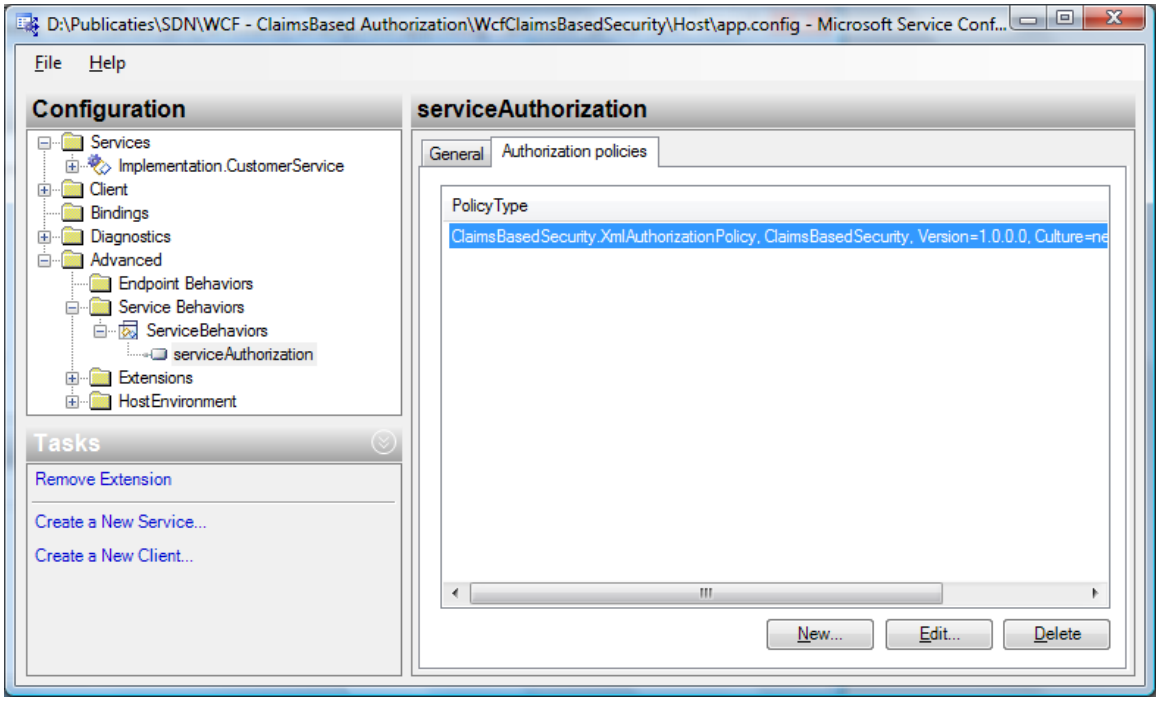
Listing 10: Het controleren of een claim voor de SOAP action is toegekend



Figuur 1: ServiceAuthorizationBehavior wordt een onderdeel van de WCF runtime



Figuur 2: Registreren van een authorizationmanager bij een servicehost m.b.v. het serviceAuthorization behavior in configuratie



Figuur 3: Registreren van een authorizationpolicy in configuratie