Alex Thissen is senior-docent software development bij Twice IT Training.

Identiteiten en principals

ONTWIKKELEN VAN VEILIGE APPLICATIES MET SYSTEM.SECURITY.PRINCIPAL NAMESPACE

Voor applicaties zijn identiteiten een belangrijk begrip, aangezien zij de gebruikers vertegenwoordigen die met de applicatie interacteren. Een identiteit representeert daarom impliciet de rechten die aan de gebruiker verleend zijn. Een ander belangrijk concept in security-scenarios is de principal. De principal representeert een identiteit met de daarbij behorende rollen. Principals worden vooral gebruikt in situaties waar op basis van rollen rechten in een applicatie worden gegeven. In dit artikel kijken we hoe het .NET Framework met identiteiten en principals werkt en wat dit voor invloed heeft op de security en het functioneren van applicaties.

Een identiteit wordt vastgesteld door authenticeren van de gebruiker. Hierbij presenteert de gebruiker credentials die een autoriteit valideert. De combinatie van gebruikersnaam en wachtwoord is de bekendste set van credentials, maar authenticatie door middel van tokens, zoals smart-cards of certificaten behoren ook tot de mogelijkheden.

In het geval van Windows-authenticatie valideert het Windows operating-systeem de gebruikersnaam en wachtwoord combinatie van een gebruiker. Het systeem valideert de gegevens door ze te vergelijken met de gegevens die opgeslagen zijn in de Security Account Manager (SAM) database of Active Directory. Wanneer er een Windows account wordt gevonden die overeenkomt met de credentials, dan krijgt de gebruiker de corresponderende identiteit. De bijbehorende rollen van de identiteit zijn in dit geval de Windowsgroepen waar de gebruiker deel van uitmaakt.

Bij custom-authenticatie zal een applicatie of service de gepresenteerde credentials toetsen bij een zelf gemaakte opslagplaats, zoals bijvoorbeeld een database. De eventuele rollen van de identiteit zijn naar eigen inzicht gedefiniëerd. In dit geval staan dus zowel de identiteiten als de rollen volledig los van de Windows accounts en -groepen. Na authenticatie vindt meestal authorisatie plaats. Hierbij wordt gecontroleerd wat de rechten van een principal zijn op basis van diens identiteit en rollen. Deze authorisatie vindt op verschillende momenten plaats binnen het Windows Operating-systeem, bijvoorbeeld bij het benaderen van resources zoals het NTFS-bestandssysteem, database-connecties of de registry. Verder kan zowel programmatisch als declaratief geauthoriseerd worden op de basis van de rollen van een principal. Binnen .NET heet dit Role Based Security.

.NET Framework principal classes

Het .NET Framework kent een aparte namespace System.Security .Principal met daarin een aantal classes en interfaces die het werken met identiteiten en principals mogelijk maken. De classes zijn afgestemd op het specifiek soort identiteit en diens rollen. Voor Windows-authenticatie zijn dit de WindowsIdentity en WindowsPrincipal. Daarnaast is er voor situaties waar custom-authenticatie wordt gedaan eenzelfde tweetal, genaamd GenericIdentity en GenericPrincipal. Alle classes die een identiteit representeren implementeren de interface IIdentity. Voor principal classes is dit de IPrincipal interface. In listing 1 staan de prototypes van deze interfaces.

Met deze interfaces is het mogelijk om runtime allerlei gegevens over de identiteit van een gebruiker en diens rollen te achterhalen. De IPrincipal interface stelt ons in staat te toetsen op rollen en geeft ons toegang tot het identiteit-object via de Identity property. Met het identiteit-object, dat IIdentity dient te implementeren, kunnen we zien of er authenticatie heeft plaatsgevonden en zo ja, met welk mechanisme dit gebeurde en wat de naam van de geauthenticeerde gebruiker is. Merk op dat het niet mogelijk is om een lijst met rollen van de principal te achterhalen. Er kan alleen getoetst worden op een specifieke rol.

Praktisch gezien betekent dit dat als we een principal object hebben, we de identiteit van de gebruiker weten en kunnen testen of de principal een bepaalde rol heeft. Op grond daarvan kunnen we rechten verlenen. Een interessante vraag is op welke manier we een principal object te pakken kunnen krijgen of kunnen maken. Het antwoord is verschillend voor de Windows en generic principals.

Windows principals en identiteiten

Iedere Windows-applicatie draait in een proces, waarbinnen één of meer threads de feitelijke code uitvoeren. De threads doen dit onder de Windows-identiteit die is vastgesteld voor het proces. De identiteit van een Windows thread, ook wel Windows-identiteit genaamd, is altijd te achterhalen met de statische methode WindowsIdentity.GetCurrent(). De bijbehorende WindowsPrincipal moet geconstrueerd worden via een constructor die het WindowsIdentity object als argument heeft. Het WindowsPrincipal object houdt daarna intern een lijst met rollen bij die overeenkomen met de Windowsgroepen waarvan de gebruiker deel uitmaakt.

```
public interface IPrincipal
{
        IIdentity Identity { get; }
        bool IsInRole(string role);
}
public interface IIdentity
{
        bool IsAuthenticated { get; }
        string AuthenticationType { get; }
        string Name { get; }
}
```

Listing 1.
Prototypes van IPrincipal en IIdentity interfaces

```
WindowsIdentity i = WindowsIdentity.GetCurrent();
WindowsPrincipal p = new WindowsPrincipal(i);
Debug.WriteLine("Type principal: " + ((object)p).GetType().FullName);
Debug.WriteLine("Type identiteit: " + ((object)i).GetType().FullName);
Debug.WriteLine("Gebruiker is " + (i.IsAuthenticated ? "wel" : "niet")
  + " geauthenticeerd.");
Debug.WriteLine("Naam gebruiker: " + i.Name);
Debug.WriteLine("Authenticatievorm: " + i.AuthenticationType);
Debug.WriteLine("Lid van Administrators groep? " +
  p.IsInRole(WindowsBuiltInRole.Administrator));
```

Listing 2.

Werken met de Windows principal

De code uit listing 2 haalt de Windows-identiteit op, toont de typen van het principal en identiteit-object en alle eigenschappen die hiermee te achterhalen zijn. Tenslotte toetst het op de deelname aan de Administrators-groep via IsInRole. Deze methode is een WindowsPrincipal specifieke overload die één van de ingebouwde Windowsgroepen als enumeratie accepteert. Hiermee is te voorkomen dat domain- of computernamen opgenomen moeten worden in de namen van ingebouwde groepen en dat hierin eventueel typefouten worden gemaakt. De output van dit fragment zou kunnen zijn:

```
Type principal: System.Security.Principal.WindowsPrincipal
Type identiteit: System.Security.Principal.WindowsIdentity
Gebruiker is wel geauthenticeerd.
Naam gebruiker: KILLER-CODE\Alex Thissen
Authenticatievorm: NTLM
Lid van Administrators groep? True
```

Er is ook een manier om rechtstreeks de principal van de uitvoerende thread te achterhalen. De statische property CurrentPrincipal van de System.Threading.Thread class geeft een IPrincipal-referentie terug.

```
IPrincipal p = Thread.CurrentPrincipal;
IIdentity i = p.Identity;
```

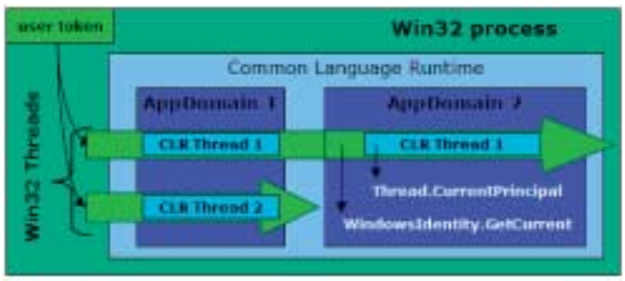
Listing 3:

Principal van een CLR thread

Als de eerste twee regels uit listing 2 vervangen zouden worden door die in listing 3, dan levert dat het volgende, onverwachte resultaat op:

```
Type principal: System.Security.Principal.GenericPrincipal
Type identiteit: System.Security.Principal.GenericIdentity
Gebruiker is niet geauthenticeerd.
Naam gebruiker:
Authenticatievorm:
Lid van Administrators groep? False
```

Je zou verwachten dat dit dezelfde Windows principal en -identiteit op zou leveren. Echter, de gevonden principal is van het type GenericPrincipal en heeft geen enkele rol. Bovendien representeert de identiteit een ongeauthenticeerde gebruiker, die daarom geen naam heeft. Op dit moment blijkt dat een .NET thread niet hetzelfde is als een Win32 Operating System thread. De Common Language Runtime (CLR) biedt threads aan die overeenkomen met een onderliggende Win32 thread (afbeelding 1). Echter, ze zijn niet hetzelfde en worden om het onderscheid duidelijk te maken ook wel eens soft (CLR) respectievelijk hard (Win32) threads genoemd. Role Based Security (zie het kader) werkt met de principals die via de CLR thread achterhaald zijn. Dat betekent dat het noodzakelijk is dat de principal van de .NET thread overeenkomt met de onderliggende Windows-identiteit. Standaard vindt deze match, zoals we hebben kunnen constateren, niet plaats. De CLR evalueert bij het aanmaken van een CLR thread hoe de principal wordt bepaald aan de hand van de principal policy. Een dergelijke policy geldt per AppDomain en is in te stellen zoals listing 4 laat zien.



Afbeelding 1. CLR biedt threads aan op basis van onderliggende Win32 threads

```
In.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
```

Listing 4.

Instellen van de principal policy voor het AppDomain

Het argument voor SetPrincipalPolicy is een waarde uit de enumeratie PrincipalPolicy. Deze heeft drie waarden:

- NoPrincipal: Thread.CurrentPrincipal geeft null terug.
- UnauthenticatedPrincipal: een GenericPrincipal met ongeauthenticeerde identiteit wordt gebruikt.
- WindowsPrincipal: de principal van de .NET thread komt overeen met die van de Win32 thread.

De standaardwaarde is PrincipalPolicy.UnauthenticatedPrincipal. Door de policy van het AppDomain op PrincipalPolicy.WindowsPrincipal te zetten is de principal van iedere daarna aangemaakte CLR thread dezelfde als die van de Win32 threads en het proces. Het moment waarop dit gebeurt is niet zo belangrijk, zolang het maar voor de eerste evaluatie van Thread.CurrentPrincipal is. De current principal wordt namelijk opgehaald via lazy instantiation, oftewel pas gemaakt bij de eerste keer opvragen. Een gebruikelijke plaats om de principal policy te zetten is in het statische entrypoint Main van de applicatie.

Nu we weten hoe de Windows principal van een CLR en Win32 thread correct te achterhalen zijn, is de volgende stap het beïnvloeden van de principal en diens identiteit.

Role Based Security

.NET kent twee vormen van security: Code Access Security (CAS) en Role Based Security (RBS). Bij CAS wordt aan code in een assembly rechten verleend op basis van het bewijs dat de assembly aandraagt. Vormen van bewijs zijn bijvoorbeeld de locatie of herkomst van de assembly en de public key token van een strong named assembly. Bij Role Based Security wordt gecontroleerd of een principal een bepaalde rol heeft. Dit kan zowel programmatisch als declaratief. De programmatische manier maakt gebruik van de IPrincipal interface en de IsInRole-methode. Hiermee heeft de programmeur volledige vrijheid in het bepalen van de rol waarop getoetst wordt. Bij declaratieve RBS wordt door middel van attributen aangegeven welke rol een principal dient te hebben. Het PrincipalPermission attribuut wordt daartoe op een klasse of methode niveau gemarkeerd. De listing in dit kader toont een voorbeeld hiervan.

```
[PrincipalPermission(SecurityAction.Demand, Role=@"BUILTIN\Administrators")]
public void RestrictedMethod()
{
  // Alleen toegankelijk voor principals in Administrators rol
  MessageBox.Show("Principal is minstens lid van Administrators groep.");
}
```

Listing:

Declaratieve Role Based Security

Een belangrijke constatering is dat de Common Language Runtime voor het achterhalen van de principal de Thread.CurrentPrincipal property evalueert.

Controle over de Windows identiteit

Voor een Windows Forms-applicatie wordt de identiteit standaard bepaald door die van de aangelogde gebruiker, die de betreffende applicatie start. Oftewel, als de gebruiker HOGWARTS\HarryP de applicatie runt, dan zal de authenticatie token van diens interactieve logon session overgenomen worden als user token in het nieuwe proces. Alle nieuwe threads binnen dat proces krijgen daardoor dezelfde identiteit, namelijk HOGWARTS\HarryP. Een applicatie kan ook onder de identiteit van een andere gebruiker worden gestart. Rechts-klik daarvoor op een snelkoppeling naar de applicatie of de executable zelf en selecteer Run As. De dialoog die volgt biedt de mogelijkheid om een andere account en credentials te geven (zie afbeelding 2).

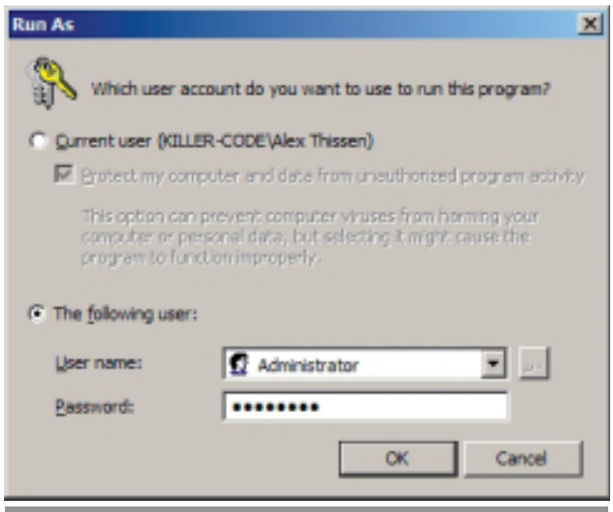
Voor een Windows Service (voorheen Windows NT Service) wordt de Windows-identiteit opgegeven in de Services MMC Snap-in uit de Administrative Tools van het Control Panel. Daarin kan een specifieke account opgegeven worden aan de hand van gebruikersnaam en wachtwoord, maar ook een speciale Local System account die beperkte rechten heeft. Als de service start, dan gebruikt deze de credentials voor de identiteit van het service proces (zie afbeelding 3).

ASP.NET-applicaties draaien standaard onder een dedicated identiteit genaamd ASPNET. Deze account is voor alle applicaties tegelijk te veranderen in de machine.config. Het <processModel> element heeft twee attributen, userName en password, waarmee de nieuwe Windows-identiteit aangegeven kan worden.

Het is ook mogelijk om per applicatie een Windows-identiteit op te geven. In de web.config bepaalt het <identity> element deze identiteit. Net als in de machine.config gaat dat via de attributen userName en password, zoals in onderstaand fragment.
<identity impersonate="true" userName="HOGWARTS\HermioneG" password="Accio" />
Op het impersonate attribuut komen we zodadelijk nog terug.

Het is ten zeerste af te raden om in de machine.config of web.config de credentials open en bloot op te nemen. Iedereen die toegang heeft tot deze configuratiebestanden kan deze gegevens dan lezen. ASP.NET kan de gegevens ook uit de registry lezen. De registry is bovendien selectief te beveiligen, zodat de gegevens daarin veiliger zijn. De web.config bevat dan alleen nog verwijzingen naar de te lezen registrykeys (zie listing 5).

Om de gegevens in de registry op te slaan wordt de tool aspnet_setreg.exe gebruikt. De volledige instructies om de configuratiebestanden op deze manier te beveiligen staan in referentie 1 (zie einde van dit artikel).



Afbeelding 2.

```
<identity impersonate="true"
  userName="registry:HKLM\SOFTWARE\MyWebApp\identity\ASPNET_SETREG,userName"
  password="registry:HKLM\SOFTWARE\MyWebApp\identity\ASPNET_SETREG,password" />
```

Listing 5.

Impersonation: veranderen van de Windows-identiteit

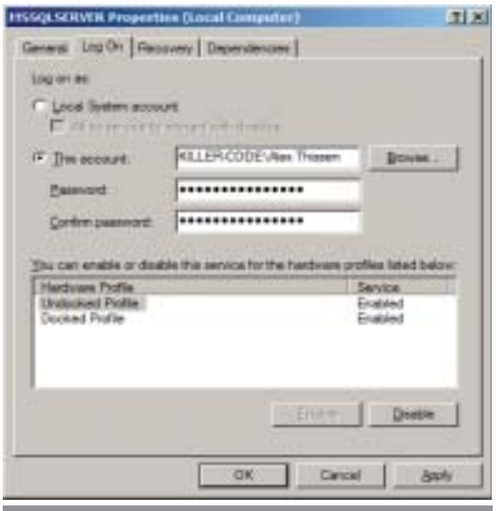
Ofschoon een Win32-applicatie vanaf de start een bepaalde identiteit heeft, is het mogelijk om bepaalde stukken code onder een andere identiteit uit te voeren. Dit proces wordt "impersonation" genoemd. Een WindowsIdentity-object is in staat om de identiteit die het vertegenwoordigt over te dragen aan de Win32 thread door middel van de methode Impersonate(). De daarop volgende code wordt onder de nieuwe identiteit uitgevoerd.

Uit een succesvolle aanroep van Impersonate komt een WindowsImpersonationContext-object terug. Dit object heeft een Undo-methode, waarmee de oorspronkelijke identiteit van de thread weer hersteld kan worden na de code met afwijkende identiteit. Het fragment in listing 6 toont het gebruik.

In listing 6 is tevens te zien dat er voorbereidend werk moet worden gedaan voordat Impersonate kan worden aangeroepen. De eerste stap in impersonation is het aanmaken van de gewenste identiteit. Dit is het lastigste deel van impersonation, omdat hiervoor een handle naar een user token nodig is. Dit wordt niet rechtstreeks door het .NET Framework ondersteund. Er is een aantal unmanaged Win32 API calls nodig om deze ontbrekende functionaliteit aan te vullen. In de custom Impersonation class bevindt zich een aantal externe, statische methoden dat verwijst naar de Win32 API-methoden LogonUser, DuplicateToken en CloseHandle. Hiermee zijn handles naar user tokens te achterhalen, te dupliceren en te sluiten. De GetToken-methode van de Impersonation class gebruikt deze methoden om het user token te achterhalen. De precieze invulling van de Impersonation klasse is terug te vinden in referentie 2 (zie einde van dit artikel).

Impersonation in ASP.NET

De hierboven beschreven wijze van impersonation is voor alle typen applicaties bruikbaar. Echter, ASP.NET is een runtime voor multi-user applicaties, waarbij het voor de hand ligt dat er regelmatig code onder de identiteit van de browsende gebruiker uitgevoerd moet worden. Daarom biedt ASP.NET de mogelijkheid om impersonation op een eenvoudigere manier te doen. Het impersonation-proces wordt verzorgd door Internet Information Server (IIS) en ASP.NET en is verder transparant voor de programmeur. De ASP.NET impersonation is configureerbaar via de web.config van de webapplicatie. Het eerder genoemde <identity> element



Afbeelding 3.

```
Debug.WriteLine("Voor impersonation: " + WindowsIdentity.GetCurrent().Name);
IntPtr userToken = Impersonation.GetToken("HOGWARTS", "RonW",
                   "5tr0nGPa55w0rD");
WindowsIdentity otherIdentity = new WindowsIdentity(userToken);
WindowsImpersonationContext context = otherIdentity.Impersonate();
// Vanaf hier is de identiteit gelijk aan HOGWARTS\RonW
Debug.WriteLine("Tijdens impersonation: " + WindowsIdentity.GetCurrent().Name);
context.Undo();
// Oorspronkelijk identiteit is hersteld
if (userToken != IntPtr.Zero)
      Impersonation.CloseHandle(userToken);
Debug.WriteLine("Na impersonation: " + WindowsIdentity.GetCurrent().Name);
```

Listing 6.

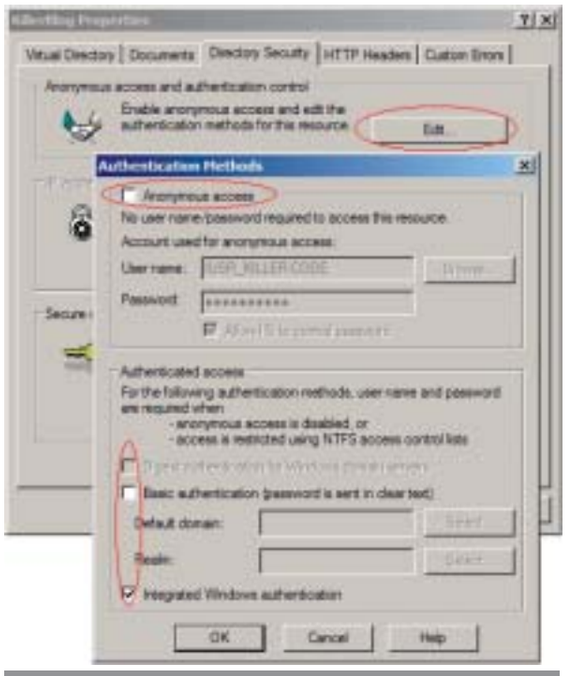
Gebruik van WindowsIdentity.Impersonate

hebben we voorheen gebruikt om een specifieke identiteit toe te wijzen aan een ASP.NET-webapplicatie. Door juist geen identiteit op te geven via het userName en password attribuut, neemt ASP.NET de identiteit aan van de geauthenticeerde gebruiker, die het webrequest maakt. Het is dan wel noodzakelijk dat via de gebruiker IIS authenticeert via Windows-authenticatie.

Als authenticatie niet plaatsvindt, dan zal de identiteit IUSR_*machinenaam* gebruikt worden; de Windows account voor ongeauthenticeerde webgebruikers. De ASP.NET service account zal bij impersonation geen rol meer spelen.
Een gebruiker van een webapplicatie wordt alleen geauthenticeerd door IIS als dat nodig is of verplicht. Het zou nodig zijn als de IUSR_*machinenaam-identiteit* niet voldoende rechten heeft om bepaalde web- of gewone resources te benaderen. Het is verplicht als anonieme toegang tot een webapplicatie niet is toegestaan. Dit laatste is zowel via IIS als ASP.NET te regelen. In de Internet Services Manager kan anonymous access uitgezet worden (zie afbeelding 4).

Als alternatief kan in de web.config voor Windows-authenticatie gekozen zijn en zal de combinatie van toegestane en geweigerde gebruikers en groepen een authenticatie moeten afdwingen (zie listing 7).

Door de impersonation van ASP.NET wordt het webrequest afgehandeld door een Win32 thread onder de identiteit van de



Afbeelding 4.

```
<identity impersonate="true" />
<authentication mode="Windows" />
<authorization>
    <deny users="?" />
</authorization>
```

Listing 7.

browsende gebruiker. In het geval van Windows-authenticatie is de principal van de CLR thread altijd diezelfde. Overigens, bij ASP.NET kan de principal van de CLR thread ook achterhaald worden door middel van de property HttpContext.Current.User. Dit is equivalent aan Thread.CurrentPrincipal.

Generic principals en identiteiten

Windows principals zijn prima in te zetten, zolang de set van accounts in één van de Windows credential stores aanwezig zijn. Wanneer dit niet het geval is of we om andere redenen gebruik willen maken van custom-authenticatie, dan is het nodig om zelf identiteiten en principals te creëren. Hiervoor zijn de GenericIdentity en GenericPrincipal classes bedoeld. Na authenticatie wordt eerst een GenericIdentity-object gemaakt met een identiteit naar keuze en vervolgens een GenericPrincipal met de gewenste rollen voor die identiteit. Door het GenericPrincipal-object toe te wijzen aan de Thread.CurrentPrincipal property kan Role Based Security worden toegepast (zie het kader over Role Based Security in dit artikel).

Custom-authenticatie gebruikt een eigen mechanisme om gebruikers te authenticeren. De validatie van de credentials vindt meestal plaats tegen een database. Het resultaat van de authenticatieslag is een bekende gebruikersnaam en geassocieerde rollen. De vorm en notatie van gebruikersnaam en rollen zijn geheel naar eigen goeddunken in te richten.

De eerste stap is steeds het aanmaken van een GenericIdentity-object. De GenericIdentity class heeft daarvoor een constructor die een string accepteert voor de naam van de identiteit. Er is ook nog een extra constructor met als extra parameter een string voor het authenticatietype. Daarna is het GenericPrincipal-object net zo eenvoudig te maken. Deze wordt geconstrueerd met een referentie naar het GenericIdentity-object en een string array voor de rollen van de principal (zie listing 8).

Tenslotte wordt de principal van de huidige thread vervangen via de Thread.CurrentPrincipal property. Alle nieuwe threads die deze thread aanmaakt krijgen automatisch dezelfde, nieuwe principal.

De generic principals en identiteiten zijn in alle typen applicaties te gebruiken, maar komen het meest voor in ASP.NET-applicaties, omdat daar (zonder impersonation) de sterkste ontkoppeling van de principal van de Win32 thread en die van de .NET thread aanwezig is. Immers, voor een webapplicatie is het het meest waarschijnlijk dat de geregistreerde gebruikers niet allemaal opgeslagen zijn als Windows accounts. Denk daarbij aan een e-commerce website met vele duizenden gebruikers, waarvan sterk gepersonaliseerde gegevens bij de accounts worden opgeslagen. Voor deze gebruikers is het ook helemaal niet noodzakelijk om een Windows account te hebben. De Win32 threads binnen ASP.NET zullen zoals altijd een Windows-identiteit hebben. In dit geval zal dit ASP.NET zijn of de identiteit die is opgegeven

```
// Normaliter komen deze gegevens uit bijv. een database
string userName = "Alex";
string[] roles = new string[] { "Programmeur", "Sporter" };
GenericIdentity identity = new GenericIdentity(userName, "CustomAuthenticatie");
GenericPrincipal principal = new GenericPrincipal(identity, roles);
Thread.CurrentPrincipal = principal;
```

Listing 8.

Constructie van een GenericPrincipal

```
private void cmdInloggen_Click(object sender, System.EventArgs e)
{
  bool blnPersistent = chkRememberMe.Checked;
  string userName, encryptedTicket;
  string[] roles;
  FormsAuthenticationTicket ticket;
  HttpCookie cookie;
  userName = txtUserName.Text;
  // AuthenticateUser methode voert authenticatie uit
  if (!AuthenticateUser(userName, txtPassword.Text))
  {
   lblMessage.Text = "Foute gebruikersnaam of wachtwoord.";
   return;
  }
  // GetRolesForUser geeft string-array terug met alle rollen van gebruiker
  roles = GetRolesForUser(userName);
  // Maak zelf een authenticatie ticket
  ticket = new FormsAuthenticationTicket(1, userName, DateTime.Now,
           DateTime.Now.AddMinutes(10), blnPersistent, String.Join(";",
           roles), FormsAuthentication.FormsCookiePath);
  encryptedTicket = FormsAuthentication.Encrypt(ticket);
  cookie = new HttpCookie(FormsAuthentication.FormsCookieName, encryptedTicket);
  Response.Cookies.Add(cookie);
  Response.Redirect(FormsAuthentication.GetRedirectUrl(userName,
           blnPersistent));
}
```

Listing 9.
Opslaan van rollen in een encrypted authenticatie cookie

via de elementen <identity> in web.config of <processModel> in machine.config.

Binnen ASP.NET bestaat er daarom de keuze uit drie authenticatiemechanismen:

- Windows-authenticatie: IIS authenticeert de gebruikers tegen de Windows domain accounts. De authenticatie vindt transparant plaats als de Windows identiteit van de browsende gebruiker overeenkomt met één van de accounts uit het Windows domain van de webserver. Als dit niet het geval is, dan moet de gebruiker inloggen via een netwerklogin dialoog.
- Forms-authenticatie: Via een webform worden credentials gevraagd, die via een ASP.NET-mechanisme of custom-authenticatie worden gevalideerd. Vervolgens wordt een authenticatiecookie uitgedeeld aan de browser die het request maakte. Deze vorm wordt ook wel cookie authentication genoemd. Na authenticatie wordt de gebruiker weer teruggestuurd naar de oorspronkelijk door hem of haar opgevraagde pagina.
- PassPort-authenticatie: De Microsoft PassPort website voert de authenticatie uit. Een ongeauthenticeerde gebruiker wordt doorgestuurd naar deze site en krijgt daar na succesvolle authenticatie een session-ticket uitgedeeld. Dit is net als bij Forms-authenticatie een cookie.

De Forms-authenticatie maakt gebruik van de generic classes. In het .NET Framework bestaat ook een PassPortIdentity voor PassPort-authenticatie. Dit laat ik verder buiten beschouwing.

De Forms-authenticatie vindt plaats als de gebruiker een webresource niet mag benaderen door diens anonieme status of op basis van de rechten van de huidige principal. Op het login-formulier, waarnaar de gebruiker automatisch door ASP.NET wordt verwezen, vult de gebruiker meestal een gebruikersnaam en wachtwoord in. Server-side voeren we de authenticatie uit en zouden dan een GenericPrincipal-object aan kunnen maken, zoals al getoond is in listing 8. Maar vanwege de disconnected-omgeving van Internet en de manier waarop ASP.NET multithreaded werkt, zal iedere daaropvolgende webrequest van dezelfde gebruiker in eerste instantie weer een ongeauthenticeerde principal bevatten. Het is dus zaak om deze "authenticatie-state" op een of andere manier vast te houden. De meest gebruikte

```
public class Global: System.Web.HttpApplication
{
  protected void Application_AuthenticateRequest(Object sender, EventArgs e)
  {
   // Toets op aanwezigheid van authenticatie cookie
   HttpCookie cookie = Request.Cookies[FormsAuthentication.FormsCookieName];
   if (cookie == null)
   return;
   // Decrypt ticket uit cookie
   FormsAuthenticationTicket ticket = FormsAuthentication.Decrypt(cookie.Value);
   string[] roles = ticket.UserData.Split(';');
   // Construeer principal
   GenericIdentity identity = new GenericIdentity(ticket.Name);
   GenericPrincipal principal = new GenericPrincipal(identity, roles);
   HttpContext.Current.User = principal;
   // Of Thread.CurrentPrincipal = principal
  }
  // Rest van code
}
```

Listing 10.
Reconstructie van GenericPrincipal bij ieder webrequest

oplossing is het opslaan van de rollen van de principal als userdata in het Forms authenticatie cookie. Hierbij speelt beveiliging van de gegevens in het cookie uiteraard een voorname rol.

Met de rollen en de Forms-identiteit is de principal steeds opnieuw te construeren bij volgende requests via het Forms-authenticatie cookie. Dit dient te gebeuren in de AuthenticateRequest eventhandler, die te vinden is in de code-behind van Global.asax. Hierna is het op ieder moment mogelijk om de principal te achterhalen en securitybeslissingen te nemen op basis van de principal, verkregen via HttpContext.Current.User of Thread.CurrentPrincipal.

Listing 9 toont hoe de rollen van een principal met behulp van encryptie veilig op de client kan worden vastgehouden. Het maken van het principal object is hier nog niet nodig, omdat er direct een redirect plaatsvindt. Listing 10 laat zien hoe in de AuthenticateRequest eventhandler voor iedere webrequest het GenericPrincipal-object wordt gemaakt en toegekend aan de CLR thread.

Conclusie

De identiteit waarmee een applicatie werkt is van belang voor het benaderen van resources en het maken van runtime autorisatiebeslissingen. Op basis van de rollen van een gebruiker kan Role Based Security ingezet worden. Het centrale concept hierbij is de principal die de combinatie vormt van een identiteit en diens rollen. Het correct inrichten van de principals binnen een applicatie verhoogt de veiligheid daarvan in grote mate. Het .NET Framework biedt met de System.Security.Principal namespace een aantal classes en interfaces aan die het werken met identiteiten en principals mogelijk maken. Ze vormen één van de onderdelen die .NET biedt voor het ontwikkelen van veilige applicaties.

Referenties

1 HOW TO: Use the ASP.NET utility to encrypt credentials and session state connection strings: http://support.microsoft.com/default.aspx?scid=kb;en-us;329290
2 .NET Framework Class Library WindowsIdentity.Impersonate method: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemSecurityPrincipalWindowsIdentityClassImpersonateTopic.asp
3 HOW TO: Create GenericPrincipal objects with Forms Authentication http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetHT04.asp
4 Building Secure ASP.NET Applications http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp

Alex Thissen is senior-docent software development bij Twice IT Training.