Gijs de Jong
Consultant – Microsoft Services

ASP.NET Security

EEN RIJK EN FLEXIBEL SECURITY-MODEL VOOR AUTHENTICATIE EN AUTORISATIE

ASP.NET biedt qua veiligheid veel meer flexibiliteit met de introductie van drie security modes, respectievelijk, Windows, Forms en Passport. In dit artikel zal alleen op de eerste twee modes worden ingaan. Om ASP.NET security goed te kunnen begrijpen is inzicht in de architectuur een absoluut vereiste. Het gebruik van het internet is de afgelopen jaren sterk toegenomen. Er heeft een ware evolutie plaats gevonden, een verandering van de 'wereldwijde encyclopedie' naar een echt commercieel platform. Aangezien elektronische handel een veel hogere graad van data-integriteit vereist, is beveiliging een steeds belangrijker aspect geworden. Active Server Pages (ASP) en Internet Information Services (IIS) boden een veilig platform voor het ontwikkelen van webapplicaties maar er was een aantal beperkingen. Het security-model van ASP / IIS was volledig gericht op het kunnen authentiseren en autoriseren van gebruikers met een equivalent Windows account. Dit was voor vele toepassingen een moeilijk aanvaardbare keuze. Om toch aan de behoefte van de markt te kunnen voorzien levert bijvoorbeeld Microsoft Commerce Server wel de funtionaliteit om eigen gedefinieerde gebruikers te kunnen authentiseren en autoriseren. ASP.NET biedt hierin echter veel meer functionaliteit.

ASP.NET security is een extra laag bovenop die van het OS. We spreken vaak van een managed security context (ASP.NET) versus een unmanaged security context (OS). De term managed komt voort uit het feit dat de executie van .NET-code gemanaged wordt door de Common Language Runtime (CLR). Uit welke onderdelen bestaat ASP.NET en hoe werken deze samen met IIS?

ASP.NET is net zoals zijn voorganger (ASP) geïmplementeerd als een isapi-extensie. De voornaamste taak van deze isapi-extensie (aspnet_isapi.dll) is het doorsturen van het request naar een volgend proces, het zogenaamde ASP.NET Workerprocess (aspnet_wp.exe). Binnen dit proces wordt de werkelijke .NET-code uitgevoerd. Alle ASP.NET applicaties draaien in dit ene proces. Er is echter een uitzondering: ASP.NET introduceert een nieuw concept dat 'Web Garden' heet. Hierbij wordt op een multi CPU server voor elke CPU een instantie van het Workerprocess opgestart. De communicatie tussen aspnet_isapi en het Workerprocess gebeurt op basis van completion ports in de vorm van named pipes. In IIS6 (Windows.NET Server) zal deze communicatie vervangen worden door een kernel-mode http listener (http.sys) voor een nog betere performance.

Nu zullen velen wel denken: alle ASP.NET -applicaties binnen hetzelfde win32 proces? Vindt er dan geen isolatie meer plaats? In IIS was het mogelijk om per virtual directory de 'Application Protection' te configureren. Hiermee was te bepalen of een webapplicatie in het proces van IIS (inetinfo.exe) draait of in een surrogaatproces in de vorm van een COM+ serverapplicatie (dllhost.exe). Op deze manier was het mogelijk om applicaties van elkaar te scheiden maar ook om buiten inetinfo.exe te draaien.

In ASP.NET wordt isolatie geboden door een concept dat een 'Application Domain' wordt genoemd. Een AppDomain kan gezien worden als een managed sub-proces binnen het fysieke win32 proces. ASP.NET creëert voor elke virtual directory in IIS een nieuw AppDomain. Het configureren van de 'Application Protection' in IIS heeft dus geen enkel effect meer op een ASP.NET -webapplicatie omdat deze altijd buiten inetinfo.exe in het eigen Workerprocess draait.

Windows Security

De eerste security mode waar we naar

zullen kijken en tevens ook de default, is Windows Security. In deze mode komt het er op neer dat IIS het authenticatie-proces uitvoert, net zoals 'vroeger'. Aangezien ASP.NET security een aparte laag op het OS is moeten we dit ook apart configureren zodat ze elkaar op een correcte manier aanvullen. Alle configuratie-settings van een ASP.NET-applicatie worden opgeslagen in het web.config bestand. Web.config bevindt zich in de root van de webapplicatie en in eventuele subdirectories tenminste als daar andere configuratie-settings vereist zijn (zie afbeelding 1).

```
<authentication mode="Windows"/>
```

Afbeelding 1.

Het configureren van IIS wordt gedaan via de IIS MMC Snap-in. De verschillende authenticatie-methodes in IIS zijn Basic, Digest, Windows Integrated en Certificates. Wanneer het authenticatie proces succesvol is verlopen wordt er voor de gebruiker een logon session op de web server gecreëerd. Het request zal vervolgens draaien in de security-context van de aangelogde gebruiker. Dit wordt mogelijk gemaakt door een concept dat impersonation heet. Aan de thread waar het request op afgehandeld wordt, wordt een impersonation token gekoppeld die verwijst naar de eerder gecreëerde logon session. Impersonation gebeurt zelfs bij anonymous-toegang van een website. Een impersonation token wordt dan aan de thread gekoppeld die verwijst naar de logon session van de anonymous-gebruiker. Dit is standaard IUSR_Machine.

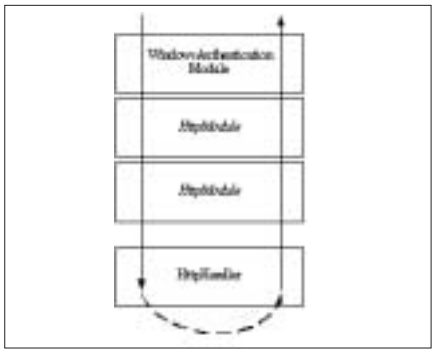
Aangezien bij het doorsturen van het request naar het ASP.NET Workerprocess een proces switch optreedt, belanden we in een andere security-context, die van het Workerprocess. Het Workerprocess draait standaard onder het aspnet account. Dit is een account met weinig rechten. Mocht je dit om wat voor reden dan ook willen aanpassen dan kan dat in het machine.config bestand. In afbeelding 2 staan de mogelijkheden.

Alle requests die door ASP.NET afgehandeld worden, gaan door een pipeline

```
<processModel userName="Machine"  password="autogenerate" />
<processModel userName="System"   password="autogenerate" />
<processModel userName="username" password="password" />
```

Afbeelding 2.

heen. Deze pipeline bestaat uit HttpModules, te vergelijken met een isapi filter, waar pre- en post processing gedaan kan worden van het request. De pipeline eindigt bij een HttpHandler, te vergelijken met een isapi extension (zie afbeelding 3).



Afbeelding 3.

WindowsAuthenticationModule is zo'n HttpModule die ervoor zorgt dat er een WindowsPrincipal object aan de managed thread wordt gekoppeld. Een principal is de representatie van een gebruiker (Identity object) en zijn bijbehorende rollen. Bij een WindowsPrincipal komen deze rollen standaard overeen met Windowsgroepen waar de gebruiker inzit. Het opvragen van de identiteit van de gebruiker kunnen we doen volgens de manier in afbeelding 4.

```
string userName = Thread.
CurrentPrincipal.Identity.Name;
```

Afbeelding 4.

Nu de gebruiker bekend is, kunnen we bepalen wat deze gebruiker allemaal mag doen binnen de applicatie. Dit principe wordt autorisatie genoemd. Een veel gebruikte vorm is Role based-autorisatie, waar op basis van de rol waar de gebruiker in zit rechten worden verleend. In .NET kan Role based-autorisatie op een declaratieve manier heel eenvoudig worden toegepast door gebruik van het PrincipalPermission attribuut (zie afbeelding 5).

Zoals je ziet is het PrincipalPermission attribuut eenvoudig te gebruiken en

```
[PrincipalPermission(SecurityAction.Demand,Role="Domain\managers")]
public void TransferMoney(double amount)
{
// Transfer the money here
}
```

Afbeelding 5.

```
public static void TransferMoney(double amount)
{
if (amount > 1000)
  {
   if (!Thread.CurrentPrincipal.IsInRole(@"domain\managers"))
   {
throw new SecurityException("You're not allowed to transfer more
than $ 1000");
   }
  }

// Transfer the money here
}
```

Afbeelding 6.

```
private void WindowsAuthentication_Authenticate(object s,
WindowsAuthenticationEventArgs e)
{
  string[] roles = GetRoles(e.Identity.Name);
GenericPrincipal principal = new GenericPrincipal(e.Identity,roles);

  Context.User = principal;
}

private string[] GetRoles(string username)
{
// Do a Database lookup here
  string[] roles = {"managers","supervisors"};
  return roles;
}
```

Afbeelding 7.

heeft het totaal geen invloed op de code binnen de functie.

In sommige gevallen wil je op een fijnere manier bepalen of iemand rechten heeft om bepaalde functionaliteit uit te voeren. Dat is te zien in afbeelding 6. Iedereen mag de TransferMoney-methode aanroepen maar alleen gebruikers in de managers groep mogen bedragen boven $ 1000 overmaken. Dit is een imperatieve vorm van Role based-autorisatie.

Het is niet altijd wenselijk of zelfs mogelijk om Windowsgroepen als rollen te gebruiken. Wanneer dit het geval is zal er een eigen security principal gecreëerd moeten worden, gebaseerd op de Windowsidentiteit en eigen gedefinieerde rollen. Deze rollen zouden bijvoorbeeld uit een SQL Server database kunnen komen.

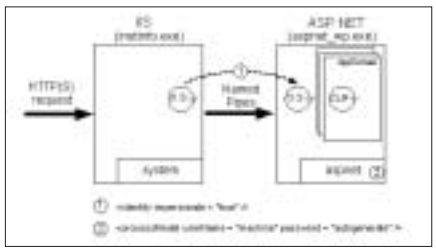
In het global.asax bestand van de webapplicatie kunnen we code implementeren die aangeroepen wordt op moment van authenticatie. Op dit moment wordt het security principal object gecreëerd en koppelen we deze aan de Context. ASP.NET zorgt er daarna voor dat deze principal ook aan de (managed) thread wordt gekoppeld (zie afbeelding 7).

Ten behoeve van het voorbeeld zijn in de code van afbeelding 7 de rollen hard gecodeerd. In een echte bedrijfsapplicatie komen deze rollen typische uit een database. Houd er ook rekening mee dat bij het laden van elke pagina deze methode aangeroepen wordt. Het is dus wenselijk om de rollen in een cache te bewaren om ze niet elke keer uit de database te hoeven ophalen.

Wanneer je externe (unmanaged) resources benadert vanuit een applicatie wil je dit in sommige gevallen kunnen doen als zijnde de aangelogde gebruiker. Om dit mogelijk te maken zal er voor de onderliggende (unmanaged) thread impersonation moeten plaats vinden. Om dit te bewerkstelligen moet dit in het Web.Config bestand aangegeven worden zoals wordt getoond in afbeelding 8.

```
<identity impersonate="true" />
```

Afbeelding 8.



Afbeelding 9.

Om alle requests in een specifieke security-context af te handelen ongeacht de ingelogde gebruiker moet de username en password ook gespecificeerd worden (zie afbeelding 10).

Een nadelige bijkomstigheid van impersonation op Windows 2000 is dat het aspnet account het SE_TCB_NAME (“Act as part of the operating system”) privilege moet hebben.

Het opvragen van de security-context van de (unmanaged) thread kan op de manier zoals in afbeelding 11 staat.

Een (unmanaged) thread heeft default geen token en erft daarom de security-context van het proces waarbinnen het leeft. Dus wanneer impersonate op false staat zal WindowsIdentity.GetCurrent(). Name de accountnaam retourneren waar aspnet_wp.exe onder draait, dit is standaard aspnet.

Forms Authenticatie

Tot nu toe zijn alleen nog maar gebruikers geauthentiseerd waarvoor een equivalent Windows account aanwezig is. Dit is in een intranetomgeving vaak de ideale oplossing maar voor het authentiseren van gebruikers op het internet worden er vaak hele andere eisen gesteld.

Forms authenticatie biedt een framework voor het authentiseren van gebruikers tegen een eigen gedefinieerde autoriteit. Een voor de handliggende mogelijkheid is dan bijvoorbeeld een database. Aangezien Forms authenticatie geheel geboden wordt door ASP.NET, moet als eerste IIS zo worden geconfigureerd dat alle http(s) requests tot aan ASP.NET komen. Dit betekent concreet dat in IIS anonymous toegang gegeven moet worden op de desbetreffende website. Vervolgens zullen we ASP.NET zo moeten configureren dat er Forms authenticatie gebruikt moet worden. Zoals inmiddels bekend is, wordt de authenticatie mode van ASP.NET geconfigureerd in web.config (zie afbeelding 12).

Met het loginUrl attribuut wordt aangegeven dat wanneer er authenticatie vereist is, de gebruiker automatisch door gestuurd moeten worden naar de gespecificeerde pagina, in dit geval dus login.aspx. Default hebben alle anonymous-gebruikers toegang. Dit is in machine.config gespecificeerd. In eerste instantie zullen we explicitiet moeten

aangeven dat anonymous toegang niet is toegestaan. Vervolgens moet de desbetreffende users en/of roles toegang worden verleend. Tot slot specificeren we dat alle overige gebruikers geen toegang hebben (zie afbeelding 13).

Een gebruiker is geauthentiseerd wanneer deze een geldig AuthenticationTicket cookie bezit. Dit wordt gedetecteerd door de FormsAuthenticationModule. Standaard is het ticket geëncrypt en wordt het gevalideerd. Mocht het noodzakelijk zijn om dit aan te passen dan kan dit door het protection attribuut op te geven (zie afbeelding 14). Het password van de gebruiker wordt niet opgeslagen in de cookie!

Wanneer het AuthenticatieTicket niet aanwezig is wordt de gebruiker doorgestuurd naar de loginpagina. Een loginpagina in de meest elementaire vorm staat in afbeelding 15.

Afbeelding 15.

Wanneer er op de loginknop geklikt wordt zal de code uitgevoerd worden die staat in afbeelding 16. Allereerste zullen we moeten bepalen of de gebruiker een geldig username en password heeft opgegeven. Dit wordt gedaan in de ValidateUser-methode. Als ValidateUser true retourneert, is er dus sprake van een geldige gebruiker. Vervolgens roepen we de FormsAuthentication.RedirectFromLoginPage-methode aan. RedirectFromLoginPage zorgt ervoor dat de AuthenticatieTicket cookie gecreëerd wordt en dat de gebruiker teruggestuurd wordt naar de initieel opgevraagde pagina. De eerste parameter van deze methode is de username, op basis hiervan wordt er de Identity gecreëerd. De tweede parameter geeft aan of het AuthenticationTicket opgeslagen moet worden als een persistent cookie.

```
<identity impersonation=”true” username=”domain|machine\user”
 password=”password” />
```

Afbeelding 10.

```
string userName = WindowsIdentity.GetCurrent().Name;
```

Afbeelding 11.

```
<authentication mode="Forms">
  <forms loginUrl="/login.aspx" />
</authentication>
```

Afbeelding 12.

```
<authorization>
<deny users="?" />
  <allow users="george" />
<allow roles="managers" />
<deny users="*" />
</authorization>
```

Afbeelding 13.

```
<forms
loginUrl="/login.aspx" protection="All|None|Encryption|Validation
/>
```

Afbeelding 14.

```
private void Login_Click(object sender, System.EventArgs e)
{
string username = Username.Text;
  string password = Password.Text;

  if (ValidateUser(username,password) == true)
FormsAuthentication.RedirectFromLoginPage(username, false);
  else
   status.Text = "Access denied !";
}

private bool ValidateUser(string username, string password)
{
// Do a Database lookup here
if (username == "george" && password == "password")
   return true;
  else
   return false;
}
```

Afbeelding 16.

Nu de gebruiker geauthentiseerd is, kunnen we vervolgens z’n bijbehorende rollen toevoegen. Om dit te doen moet de

```
protected void Application_AuthenticateRequest(Object sender,
EventArgs e)
{
if (Context.User != null)
  {
   if (Context.User.Identity.AuthenticationType == “Forms”)
   {
     string[] roles = GetRoles(Context.User.Identity.Name);
Context.User = new GenericPrincipal(Context.User.Identity,roles);
   }
  }
}

private string[] GetRoles(string username)
{
// Do a Database lookup here
  string[] roles = {"managers","supervisors"};
  return roles;
}
```

Afbeelding 17.

Application_AuthenticateRequest geïmplementeerd worden in de global.asax (zie afbeelding 17).

Na het uitvoeren van Application_AuthenticateRequest zorgt ASP.NET ervoor dat de nieuw gecreëerde principal (Context.User) aan de managed thread wordt gekoppeld. Vanaf dat moment kan Role based-autorisatie afgedwongen worden, net zoals in de vorige voorbeelden. Het veranderen van authenticatie mode heeft dus totaal geen impact op de autorisatie.

Rijk en flexibel security-model

ASP.NET biedt een rijk en flexibel security-model voor het authentiseren en autoriseren van zowel domain-gebruikers als niet domain-gebruikers.

Door de loskoppeling van authenticatie en autorisatie kunnen de rechten van de verschillende soorten gebruikers op een eenduidige manier gevalideerd worden. De ontwikkelaar dient echter wel een goed inzicht te hebben hoe de unmanaged en managed security-context geconfigureerd dienen te worden om beide correct te laten samenwerken.

Nuttige internetadressen

- Algemeen:
  http://www.asp.net
- Security Briefs - ASP.NET Security Issues:
  http://msdn.microsoft.com/msdnmag/issues/01/11/security/security0111.asp
- Security Briefs – Managed Security Context in ASP.NET:
  http://msdn.microsoft.com/msdnmag/issues/02/01/security/security0201.asp
- An Introduction Guide to Building and Deploying More Secure Sites with ASP.NET and IIS:
  http://msdn.microsoft.com/msdnmag/issues/02/04/ASPSec/ASPSec.asp
- An Introduction Guide to Building and Deploying More Secure Sites with ASP.NET and IIS, Part 2:
  http://msdn.microsoft.com/msdnmag/issues/02/05/ASPSec2/ASPSec2.asp