Nikander en Margriet Bruggeman zijn beiden freelance softwareontwikkelaars

# Drie manieren van code-impersonatie

## TREED BUITEN DE SCOPE VAN JE EIGEN SECURITY-IDENTITY

In dit artikel bespreken we drie manieren van user-impersonatie, waarbij we laten zien hoe je vanuit C#-code de security-identity kunt bepalen. Allereerst nemen we impersonatie vanuit eigen code onder de loep, waarbij de impersonatie uitgevoerd kan worden door de LogonUser Win32 API-functie aan te roepen. Als tweede bespreken we de RevertToSelf Win32 API-call. Ten slotte is het ook mogelijk code-impersonatie uit te voeren door een nieuw AppDomain te creëren.

Code wordt altijd uitgevoerd in een proces. .NET-code in eerste instantie in een CLR-proces, maar uiteindelijk altijd in een OS-proces. Code is ook altijd gekoppeld aan een security-identity, waarmee bepaald wordt wat code mag doen. Denk bijvoorbeeld maar aan het schrijven van een bestand, dat is niet iets wat iedereen mag doen. Wat doe je als je iets doet dat buiten de scope van je huidige security-identity valt? Je kunt altijd een webservice of windows-service aanroepen, of een component via remoting of een enterprise service-component. Deze code wordt in een eigen security-identity uitgevoerd. Daarnaast kun je ook vanuit je eigen code programmatisch iets aan de security-identity doen waarbinnen de door jou beoogde taak wordt uitgevoerd.

### Voorbeeld

Als voorbeeld nemen we code die het SharePoint-administration objectmodel gebruikt. Deze code vereist dat de uitvoerder ervan SharePoint-administrator-rechten heeft. Overigens hoef je voor het begrijpen van het artikel niets te weten van SharePoint. Als je de beschikking hebt over SharePoint Portal Server 2003 en je wilt de code zelf testen, maak dan een account met SharePoint-reader-rechten. Het codevoorbeeld 1 gebruikt het SharePoint-objectmodel om de url en server-id van de huidige virtual server te tonen. Hiervoor zijn reader-rechten niet voldoende en moet je administrator zijn. Maak een webpart en override de RenderWebPart-methode; zie codevoorbeeld 1. Als je deze webpart uitvoert door hem op een webpartpagina te plaatsen en hem via de browser bekijkt met een SharePoint-readeraccount, krijg je een 'access denied'-fout te zien.

```
protected override void RenderWebPart(HtmlTextWriter output)
{
  string strValue = String.Empty;
  try
  {
    SPSite objSite = SPControl.GetContextSite(Context);
    SPGlobalAdmin objAdmin = new SPGlobalAdmin();
    SPVirtualServer objServer = objAdmin.OpenVirtualServer
      (new Uri(objSite.Url));
    objServer.CatchAccessDeniedException = false;
    strValue += "url: " + objServer.Url + " virtual server id: " +
      objServer.VirtualServerId;
  }
  catch (Exception err)
  {
  strValue = "Error in wp: " + err.Message;
  }
  output.Write(strValue);
}
```

Codevoorbeeld 1.

Let op: SharePoint is secure by default, dus standaard mag je niet veel doen. De security-identity at runtime wijzigen is één van de dingen die niet is toegestaan. Zo moet je onder meer Execution-, UnmanagedCode-, ControlPrincipal-, ControlAppDomain- en ControlEvidence-security-permissions hebben. Om het gemakkelijk te maken kan het trust-level in SharePoint op 'Full' gezet worden, zodat alle codevoorbeelden in dit artikel uitgevoerd kunnen worden. Dit doe je op de volgende manier in het bestand web.config:

```
<trust level="Full" originUrl="" />
```

### LogonUser

Een manier om een ander user-account te impersoneren is via de LogonUser Win32 API- call. Als je deze functie wilt aanroepen, moeten de advapi.dll en kernel32.dll geïmporteerd worden. Dit kun je doen door de code zoals in codevoorbeeld 2 aan je class toe te voegen (in ons voorbeeld aan de webpart-class):
Advapi32.dll bevat de LogonUser-functie die een gebruiker probeert aan te loggen. Deze functie krijgt als argumenten onder meer user-naam, domein en wachtwoord mee. De functie geeft als returnwaarde een Boolean terug die vertelt of de login succesvol was. Er wordt by reference een handle doorgegeven. Deze handle is erg belangrijk, omdat deze gebruikt kan worden om een nieuwe windows-identity te maken. Nu de benodigde Win32-functies zijn gedeclareerd, is het verstandig bij het aanroepen ervan een aantal enumerations te declareren die als argumenten voor de LogonUser-functie gebruikt kunnen worden. Voeg codevoorbeeld 3 toe aan je class.
De kernel32 dll bevat de CloseHandle-functie waarmee de handle, die eerder werd geopend tijdens de LogonUser-call, weer netjes gesloten kan worden. Codevoorbeeld 4 laat zien hoe een gebruiker geïmpersonaliseerd kan worden via de LogonUser API-call:

```
[DllImport("advapi32.dll", SetLastError=true)]
static extern bool LogonUser(
  string principal,
  string authority,
  string password,
  LogonTypes logonType,
  LogonProviders logonProvider,
  out IntPtr token);

[DllImport("kernel32.dll", SetLastError=true)]
static extern bool CloseHandle(IntPtr handle);
```

Codevoorbeeld 2.

```
enum LogonTypes : uint
{
   Interactive = 2,
   Network,
   Batch,
   Service,
   NetworkCleartext = 8,
   NewCredentials
}
enum LogonProviders : uint
{
   Default = 0,
   WinNT35,
   WinNT40,
   WinNT50
}
```

Codevoorbeeld 3.

In codevoorbeeld 4 werd de huidige user-context vervangen door een nieuwe context. Een groot nadeel van deze aanpak is dat de user-credentials in plain text in de code opgeslagen worden. Dat is onveilig. Je kunt beter het wachtwoord in een config-file bewaren en encrypten met DPAPI. Omdat we in dit voorbeeld met SharePoint Portal Server werken, zou het ook een mooie oplossing zijn om SharePoint Single Sign On (SSO) te gebruiken en de user-credentials op te slaan in de credential mapping-database.

## Credential-less impersonation

Er is een andere oplossing die ook wel 'credential-less impersonation' genoemd wordt. In deze oplossing wordt de Win32-API RevertToSelf-functie aangeroepen. In een ASP.NET-webapplicatie wordt een user-account geïmpersonaliseerd. Standaard wordt hiervoor het ASPNET-user-account gebruikt. Als windows-authentication aanstaat wordt het account gebruikt waarmee de bezoeker van de website is ingelogd. Het aanroepen van de RevertToSelf-functie zorgt ervoor dat deze impersonatie wordt beëindigd.
SharePoint Portal Server 2003 maakt gebruik van Internet Information Server 6 (IIS). In IIS 6 worden application pool identities gebruikt als de context waarin een worker-process wordt uitgevoerd. Het is mogelijk van de huidige user-context (het windows-useraccount van de gebruiker, dat zelf al een geïmpersoneerde identity is) terug te switchen naar de originele identity van het worker-process, de application pool identity. De credentials van deze application pool identity worden veilig opgeslagen in de IIS- metabase. In ons SharePoint-voorbeeld wordt het daarmee erg gemakkelijk, SharePoint vereist dat de application pool identity local administrator-rechten heeft en bovendien administrator is van de SharePoint-contentdatabase. Het gevolg is dat het dumpen van de huidige user-context en het terugvallen naar de application pool security-context leidt tot een context waarin code uitgebreide privileges geniet. Met deze aanpak wordt vermeden dat credentials in code bewaard moeten worden. Het volgende voorbeeld laat zien hoe dit in zijn werk gaat. Voeg de volgende code toe aan een class-definitie.

```
[DllImport("advapi32.dll")]
static extern bool RevertToSelf();
```

Zoals te zien valt is de RevertToSelf-function gemakkelijk te gebruiken. Codevoorbeeld 5 demonstreert het gebruik ervan.

## Impersonatie via nieuw AppDomain

Naast deze twee methodes is er nog een derde, complexere aanpak. Deze aanpak is in SharePoint 2003-development geen overbodige luxe. In sommige (ongedocumenteerde) gevallen is duidelijk geworden dat bij gebruik van het SharePoint-objectmodel acties gevalideerd worden tegen de originele context waarin een request gedaan werd. Dit zijn de credentials waarmee de user is ingelogd.

```
protected override void RenderWebPart(HtmlTextWriter output)
{
   string strValue = String.Empty;
   try
   {
      WindowsImpersonationContext objUserContext;
      IntPtr objToken;
      WindowsIdentity objOrgIdentity;
      WindowsIdentity objIdentity;
      bool blnReturn = LogonUser(@"myadministrator", "mydomain",
         "myadminpassword",
      LogonTypes.Interactive,
      LogonProviders.Default,
      out objToken);
      if ( blnReturn )
      {
         objOrgIdentity = WindowsIdentity.GetCurrent();
         objIdentity = new WindowsIdentity(objToken);
         objUserContext = objIdentity.Impersonate();
         SPSite objSite = SPControl.GetContextSite(Context);
         SPGlobalAdmin objAdmin = new SPGlobalAdmin();
         SPVirtualServer objServer = objAdmin.OpenVirtualServer(new
         Uri(objSite.Url));
         objServer.CatchAccessDeniedException = false;
         strValue += "url: " + objServer.Url + " virtual server id: " +
         objServer.VirtualServerId + "<br/><br/>";
         strValue += "Identity name after impersonation: " + " " +
         objIdentity.Name + "<br/>";
         objUserContext.Undo();
         strValue += "Indentity name when impersonation is undone: " +
         objOrgIdentity.Name;
         CloseHandle(objToken);
      }
      else
      {
         strValue = "Logon failed!";
      }
   }
   catch (Exception err)
   {
      strValue = "Error in wp: " + err.Message;
   }
   output.Write(strValue);
}
```

Codevoorbeeld 4.

De twee vorige manieren om impersonatie te bewerkstelligen werken dan niet meer. Deze derde aanpak ziet er als volgt uit.

1. Maak gebruik van de LogonUser- of RevertToSelf Win32 API-calls om een user-account met veel privileges te impersoneren.
2. Maak een nieuw AppDomain.
3. Voer de taken die aanvullende privileges nodig hebben in het nieuwe child AppDomain uit.
4. Marshall de resultaten terug naar het parent AppDomain.
5. Unload het child AppDomain.
6. Switch terug naar de originele identity-context.

De eerste stap en de laatste stap zijn al in dit artikel uitgelegd. Om de andere stappen te begrijpen moet je het een en ander weten over AppDomains. AppDomains zijn vergelijkbaar met operating system (OS)-processen. Zij vormen het fundament waarin code wordt uitgevoerd. Het grote verschil tussen OS-processen en AppDomains is dat OS-processen abstracties zijn die door het operating systeem zelf gemaakt worden, AppDomains zijn abstracties die gecreëerd worden door de Common Language Runtime (CLR). Elk AppDomain bevindt zich in één OS-proces, elk OS-proces kan meer AppDomains hosten. Een object of type wordt altijd binnen een AppDomain ingeladen. Hetzelfde type kan uiteraard ook nog steeds door andere AppDomains ingeladen worden. Een

object-reference moet altijd verwijzen naar een object in hetzelfde AppDomain. Met de volgende code kun je programmatisch een AppDomain maken en vernietigen.

```
AppDomain objAppChild = AppDomain.CreateDomain("MyChild");
AppDomain.UnLoad(objAppChild);
```

Als je code in een ander AppDomain wilt inladen en uitvoeren en de resultaten wilt ophalen, dan kun je doen zoals in codevoorbeeld 6 is aangegeven.
Vergeet het Unwrap-gedeelte eventjes, daar komen we later op terug. We concentreren ons nu op de obj2.DoSomething()-call. Omdat we zojuist hebben vastgesteld dat het onmogelijk is om een object-reference te hebben naar een object in een ander AppDomain, is de vraag gerechtvaardigd hoe de DoSomething-methode aangeroepen kan worden. Om de illusie van een object-reference naar een object in een ander AppDomain te wekken, moet een type gemarshalled kunnen worden. Niet elk type kan gemarshalled worden. Als je een eigen class maakt, zal deze by default remote-unaware zijn. Dit betekent dat het type niet naar een ander AppDomain gemarshalled kan worden. Als je een type als [Serializable] markeert, dan wordt dat een Unbound-type genoemd. Unbound-types worden gemarshalled by value. De CLR zal een kloon maken van een object dat op geen enkele manier nog verbonden is met het originele object. Deze kloon wordt doorgegeven aan een tweede AppDomain. Naast een remote-unaware-type en een Unbound-type kennen we ook de AppDomain-bound-type. AppDomain-bound-types worden gemarshalled by reference.
De CLR geeft een proxy-object aan een tweede AppDomain. Dit proxy-object 'forward' alle calls die vanuit dat AppDomain naar de proxy gedaan worden naar het originele object dat zich in het originele AppDomain bevindt. Alle AppDomain-bound-types worden direct of indirect afgeleid van System.MarshalByRefObject. Stel dat we een AppDomain-bound-class maken met de naam MyClass, en stel dat deze class enkele operaties uitvoert die gebruikmaken van het SharePoint-objectmodel, dan zou de code er uit kunnen zien als in codevoorbeeld 7.
Er is nog één belangrijk onderwerp dat besproken moet worden over de CreateInstance-call. Bekijk de volgende instantiëring in codevoorbeeld 8.
De CreateInstance-methode maakt een instantie van een AppDomain-bound-type. Maar de CreateInstance-methode geeft een object-handle terug, geen echte object-reference. Dat betekent dat het type MyType niet in het AppDomain objMyAppDomain ingeladen wordt, totdat de Unwrap-methode wordt aangeroepen. Dit is handig en efficiënt in scenario's waarbij meer AppDomains gemaakt worden. De parent AppDomain zou bijvoorbeeld een object-handle naar een object in AppDomain-child1 door kunnen geven aan AppDomain-child2 zonder dat het type van het object in het parent AppDomain ingeladen hoeft te worden. Dat betekent wel dat de Unwrap-methode altijd eerst aangeroepen moet worden, voordat een proxy-object daadwerkelijk gebruikt kan worden.
Een andere interessante vraag is waar de CLR kijkt als het een assembly probeert in te laden tijdens de aanroep van CreateInstance. Het is mogelijk een assembly in de Global Assembly Cache (GAC) te deployen, maar het is ook mogelijk andere locaties te specificeren. Elk AppDomain heeft een Setup Information-property waarmee toegang verkregen wordt tot data die het gedrag van de assembly resolver beïnvloeden. In ons voorbeeld zullen we de informatie domweg kopiëren van het huidige AppDomain.

```
AppDomainSetup objAppDomainSetup = AppDomain.CurrentDomain.SetupInformation;
```

Omdat dat in ons voorbeeld over het AppDomain gaat waarin de webpart-assembly is ingeladen, is het mogelijk om de assembly die

```
protected override void RenderWebPart(HtmlTextWriter output)
{
  string strValue = String.Empty;
  try
  {
    WindowsIdentity objOriginalUser = WindowsIdentity.GetCurrent();
    RevertToSelf();
    SPSite objSite = SPControl.GetContextSite(Context);
    SPGlobalAdmin objAdmin = new SPGlobalAdmin();
    SPVirtualServer objServer = objAdmin.OpenVirtualServer(new
    Uri(objSite.Url));
    objServer.CatchAccessDeniedException = false;
    strValue += "url: " + objServer.Url + " virtual server id: " +
    objServer.VirtualServerId + "<br/><br/>";
    strValue += "application pool identity name: " +
    WindowsIdentity.GetCurrent().Name + "<br/>";
    WindowsImpersonationContext objContext =
    objOriginalUser.Impersonate();
    strValue += "original user name: " +
    WindowsIdentity.GetCurrent().Name;
  }
  catch (Exception err)
  {
    strValue = "Error in wp: " + err.Message;
  }
  output.Write(strValue);
}
```

Codevoorbeeld 5.

```
Object obj = objMyAppDomain.CreateInstanceAndUnwrap("MyAssembly", "MyType");
MyClass obj2 = (MyClass) obj;
String strValue = obj2.DoSomething();
```

Codevoorbeeld 6.

```
using System;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Administration;
namespace SeveralLib.Impersonator
{
  /// <summary>
  /// Summary description for MyClass.
  /// </summary>
  public class MyClass : MarshalByRefObject
  {
    public MyClass()
    {
      //
      // TODO: Add constructor logic here
      //
    }
    public string GetSomeValue()
    {
      string strReturn = String.Empty;
      SPSite objSite = new SPSite("http://myportal");
      SPGlobalAdmin objAdmin = new SPGlobalAdmin();
      SPVirtualServer objServer = objAdmin.OpenVirtualServer
        (new Uri(objSite.Url));
      objServer.CatchAccessDeniedException = false;
      strReturn += "url: " + objServer.Url + " server id: " +
        objServer.VirtualServerId;
      return strReturn;
    }
  }
}
```

Codevoorbeeld 7.

```
ObjectHandle objHandle = objMyAppDomain.CreateInstance("MyAssembly", "MyType")
Object objProxy = objHandle.Unwrap();
```

Codevoorbeeld 8

```
// Stap 1. Roep LogonUser of RevertToSelf Win32 API functies aan.
AppDomainSetup objAppDomainSetup = AppDomain.CurrentDomain.
   SetupInformation;
Evidence objEvidence = AppDomain.CurrentDomain.Evidence;
AppDomain objMyAppDomain = AppDomain.CreateDomain("MyAppDomain",
   objEvidence, objAppDomainSetup);
try
{
ObjectHandle objHandle = objMyAppDomain.CreateInstance(Assembly.Get-
   ExecutingAssembly().GetName() FullName, typeof(MyClass).FullName);

   Object objProxy = objHandle.Unwrap();
   MyClass objClass = (MyClass) objProxy;
   String strValue = objClass.GetSomeValue();
}
catch (Exception err)
{
// Voer error handling uit.
}
finally
{
 AppDomain.Unload(objMyAppDomain);
}
// Switch back naar de originele context.
```

**Codevoorbeeld 9.**

we dynamisch willen inladen te deployen in [driveletter]:\inetpub\wwwroot\bin folder. Let er op dat de SetupInformation- property van een nieuw AppDomain gezet moet worden, voordat het AppDomain daadwerkelijk wordt gecreëerd. Daarna is het niet meer mogelijk AppDomain Setup-informatie te wijzigen.

Het nieuwe AppDomain heeft ook Evidence-informatie over security-settings nodig. Om het onszelf makkelijk te maken kopiëren we deze ook van het huidige AppDomain.

```
Evidence objEvidence = AppDomain.CurrentDomain.Evidence;
```

Dit resulteert uiteindelijk in de volgende code die aan de RenderWebPart-methode van een webpart toegevoegd kan worden; zie codevoorbeeld 9.

Als we deze aanpak volgen, werkt impersonatie ook bij het gebruik van het SharePoint- administration-objectmodel in alle gevallen correct.

## Conclusie

In dit artikel zijn drie verschillende manieren van user-impersonatie besproken. Impersonatie vanuit eigen code kan uitgevoerd worden door de LogonUser Win32 API- functie aan te roepen. Als je wilt vermijden dat de user-credentials in een configuratie- file opgeslagen moeten worden, kan je ook gebruikmaken van de RevertToSelf Win32 API-call. Ten slotte is het ook mogelijk code-impersonatie uit te voeren door een nieuw AppDomain te creëren. Deze methode is in sommige gevallen onvermijdelijk als de andere twee manieren falen.

**Nikander** en **Margriet Bruggeman** werken beiden als freelance oftwareontwikkelaar en architect (www.lcbridge.nl). Zij zijn voormalig SharePoint MVP's en medeauteurs van de SharePoint Products and Technologies 2003 Resource Kit. Voor vragen en opmerkingen zijn zij te bereiken via info@lcbridge.nl.