Quellcode auf CD

Vollstes Vertrauen

Auswirkungen der Code Access Security von .NET – Teil 2

von Andreas Kosch

Nachdem sich der erste Teil um die Begrifflichkeiten und die neuen Regeln am exemplarischen Fall der Direct Execution gekümmert hat, steht im zweiten Teil mehr der praktische Aspekt an der ganzen Sache im Vordergrund. Das fängt beim Einrichten verschiedener Sandkästen mit unterschiedlichen Sicherheitskontexten an und hört beim Zumauern des Zugriffswegs für Dritte auf eigene Assemblies im dritten Teil dieser Artikelserie auf.

Eine Assembly wird in einem Sicherheitskontext ausgeführt, für den vorher bestimmte Berechtigungen definiert wurden. Die einmal geladene Assembly kann diese Berechtigungen nur verringern, aber niemals erhöhen. Dies führt dazu, dass immer dann eine vorherige Konfiguration notwendig wird, wenn die Assembly aufgrund des Sicherheitskontexts nur eine eingeschränkte Default-Berechtigungsmenge erhält, die für die beabsichtigte Nutzfunktion aber nicht ausreichen würde. Das .NET Framework stellt vier prinzipiell unterschiedliche Wege für den Zugriff auf die Sicherheitskonfiguration zur Verfügung:

- Visuelle Benutzeroberfläche für die interaktive Konfiguration in Form des Moduls .NET Framework Configuration *(mscorcfg.msc)* für die Microsoft Management Console
- Kommandozeilen-Anwendung *caspol.exe* für den Einsatz in Batch-Dateien
- Klassen aus dem .NET Framework (*Installer*, *SecurityManager*, *AppDomain* usw.) für die Konfiguration über ein eigenes Tool beziehungsweise durch das Installationsprogramm der Anwendung
- Direktes Manipulieren der XML-Konfigurationsdateien (auch wenn das eigentlich nicht im Sinne des Erfinders ist)

In meinem Artikel ist nur Platz für den ersten, bequemen und sicheren Weg: Über die Dialoge der visuellen Benutzeroberfläche der .NET Framework Configuration können Sie sich niemals ungewarnt ins Knie schießen. Die anderen Wege sind schon gefährlicher, denn die CLR erlaubt es Ihnen, sich mit einer unüberlegten Aktion selbst auszusperren. Für diesen Worst Case sollten Sie sich merken, dass ein beherztes Löschen der XML-Konfigurationsdateien ausreicht, um den Originalzustand der Sicherheitskonfiguration eines frisch installierten .NET Frameworks wiederherzustellen. In diesem Fall legt die CLR die XML-Dateien in der Default-Fassung neu an. Wenn Sie sich nun fragen, wo diese XML-Dateien zu finden sind, gibt die CLR auf Nachfrage selbst Auskunft, wie das folgende Beispiel zeigt (die VB.NET-Fassung finden Sie im Beispielprojekt auf der CD-ROM zum Heft).

Die Methode *PolicyHierarchy* der *SecurityManager*-Klasse liefert über die *PolicyLevel*-Instanzen die Informationen getrennt nach den Sicherheits-Ebenen zurück. Abbildung 1 zeigt unter anderem das Ergebnis dieses Aufrufs.

```
IEnumerator i = SecurityManager.PolicyHierarchy();
while (i.MoveNext())
{
 PolicyLevel level = (PolicyLevel)i.Current;
 listBox1.Items.Add(string.Format("{0,10}: {1}",
  level.Label, level.StoreLocation));
}
```

Abb. 1: Die CLR unterscheidet verschiedene Sicherheits-Ebenen

Policy Level

Wenn Sie sich nun fragen, warum gleich drei verschiedene XML-Dateien im Spiel sind, hängt das mit dem Nachbilden des realen Lebens zusammen. Unter .NET gibt es vier verschiedene Policy Level, die unabhängig von allen anderen sind und deren Zweck darin besteht, die in der realen Wirklichkeit eines Firmen-Netzwerks zu berücksichtigenden unterschiedlichen Zuständigkeiten nachzubilden:

- *Enterprise*: Höchster Level für das vollständige Netzwerk. Der Administrator des Active Directory erhält die Macht, die Einhaltung seiner Regeln firmenweit zu erzwingen.
- *Machine*: Erlaubt dem lokalen Administrator dieses Rechners, die Regeln für den verwalteten Programmcode festzulegen, wobei dieser Level primär auf den Sicherheitszonen des Internet Explorers basiert.
- *User*: Erlaubt jedem Benutzer, eigene Regeln für den verwalteten Programmcode festzulegen, um auf diesem Rechner für jeden Benutzer eigene Einstellungen zu aktivieren.
- *AppDomain*: Legt die Regeln für den verwalteten Programmcode in einer Application Domain fest, wobei diese Regeln nur innerhalb der Anwendung beziehungsweise vom aufrufenden Host definiert werden können. Aus diesem Grund gibt es für diese Ebene auch keine XML-Datei und auch keine visuelle Konfigurationsmöglichkeit.

Die Kombination dieser vier Policy Levels wird als Policy-Hierarchie des Systems bezeichnet. Jeder Level in der Hierarchie vergibt eine Menge von Berechtigungen (Permissions) aufgrund des vorgefundenen „Beweismittels“ (Evidence). Das Ergebnis der Berechtigungen ist die Schnittmenge der einzelnen Levels, sodass das Prinzip des kleinsten gemeinsamen Nenners der Berechtigungen der verschiedenen Policy Levels wirkt. Um die Angelegenheit noch etwas komplizierter zu machen, hat jeder Policy Level drei Bestandteile, die in dieser Form auch in der TreeView-Darstellung (siehe Abb. 1) angezeigt werden:

- *Code Groups*-Hierarchie als Zuordnungssystem von Assemblies zu bestimmten Sicherheitseinstellungen. Über die so genannte Membership Condition wird zur Laufzeit je nach vorgefundenem Evidence eine Assembly einer oder mehreren Code Groups zugeordnet, wobei die Gruppe *All_Code* die Wurzel bildet. Das Eingruppierungsmerkmal können zum Beispiel das Installationsverzeichnis, ein bestimmter Aufruf-URL oder ein Strong Name sein. Der Administrator kann eine neue Code Group anlegen, wenn eine Assembly eine Konfiguration erfordert, die von einer bereits vorhandenen Code Group nicht abgedeckt wird.
- *Permission Set*-Aufstellung mit dem Verzeichnis der Berechtigungsmengen, die einer bestimmten Code Group zugeordnet wurden. Der Administrator kann entweder eines der vordefinierten Permission Sets zuordnen oder er legt eine eigene Berechtigungsmenge an, um dort gezielt nur die Rechte zu vergeben, die von der auszuführenden Anwendung benötigt werden.
- *Policy Assemblies* beziehungsweise die Liste von *FullTrust*-Assemblies.

Ein Beispiel schafft Klarheit

Bevor Sie nun entnervt zu einem anderen Beitrag weiterblättern, soll ein Beispiel zeigen, dass es in der Praxis dann doch nicht allzu kompliziert ist. Angenommen, eine Assembly wird über den URL *http://bad.guy.com* heruntergeladen und ausgeführt. In diesem Fall gehört die Assembly aufgrund ihrer Download-Herkunft automatisch gleich zwei verschiedenen Gruppen an (Tabelle 1 stellt Ihnen weitere Beispiele vor):

- *All_Code* (dem Wurzeleintrag für die anderen Code Groups)
- *Internet_Zone* (alle numerischen IP-Adressen bzw. URLs mit einem Punkt im Namen)

Das Sicherheitssystem der CLR geht davon aus, dass ein Recht speziell eingeräumt werden muss, bevor es gültig wird. In der Voreinstellung räumen die Policy Levels *Enterprise*, *User* und *AppDomain* einer Assembly über die Gruppe *All_Code* generell alle Rechte (*FullTrust*) ein, sodass nur die vorinstallierten Beschränkungen im Policy Level *Machine* die zur Laufzeit wirksamen Regeln bestimmen. Und der Level *Machine* ordnet in der Voreinstellung der Gruppe *All_Code* nur die Berechtigungsmenge *Nothing* zu. Dies hat die Konsequenz, dass immer dann, wenn eine Assembly nicht einer anderen Gruppe zugeordnet werden kann, diese Assembly keine Rechte hat und somit nur eine unnütze Byteanordnung darstellt. Würde diese Assembly vom lokalen Dateisystem des Rechners gestartet, würde zusätzlich die Gruppe *My_Computer_Zone greifen*, sodass die Assembly *FullTrust*-Berechtigungen (d.h. eine uneingeschränkte Berechtigungsmenge) erhielte. Da die Assembly in meinem Beispiel über einen Internet-URL aufgerufen wurde, gehört diese Assembly aber zusätzlich zur Gruppe *Internet_Zone*, sodass eine stark eingeschränkte Berechtigungsmenge wirksam wird. Die vom .NET Framework vorinstallierte *Machine*-Policy für die Internet-Zone wird dieser Assembly einen Schreibzugriff auf ein beliebiges Festplattenver-

.NET und das Internet

Beim Thema Policy Level und Code Group spielt die verwendete Version des .NET Frameworks eine große Rolle, wobei im Fall der Version 1.0 auch die installierten Service Packs das Verhalten bestimmen. Generell kann man sagen, dass Microsoft mit jeder neuen Framework-Version die Zügel etwas straffer anzieht, was Sicherheits-Belange angeht. Die Aussagen im Artikel beziehen sich auf das .NET Framework 1.1.

Pfad	Beispiel	Zone
Lokale Datei	*C:\NET\Test.exe*	*My_Computer_Zone*
UNC bzw. URL ohne Punkt	*http://Localhost/test/test.exe* oder *\\Server\C\NET\Test.exe*	*LocalIntranet_Zone*
Alle numerischen IP-Adressen bzw. URLs mit einem Punkt	*http://127.0.0.1/test/test.exe*	*Internet_Zone*

Tabelle 1: Einige Zuordnungsbeispiele

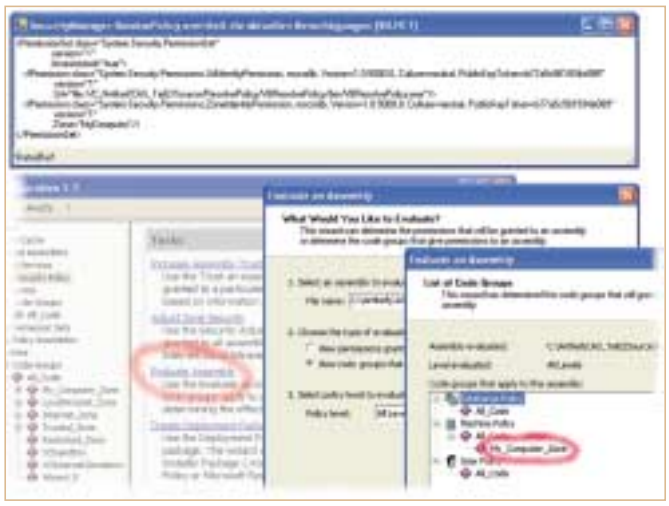

Abb. 2: Informationen über die eigene Code-Gruppe abrufen

zeichnis und vieles andere mehr verbieten. Wenn die Assembly so etwas trotzdem versucht, löst .NET eine Exception aus, wenn die Berechtigungsmenge für diese Assembly nicht vorher über die Sicherheits-Konfiguration erhöht wurde.

Über die Methode *ResolvePolicy* der *SecurityManager*-Klasse kann eine Assembly – vorausgesetzt, sie hat die Berechtigung dafür – bei der CLR nachfragen, in welchen Schubkasten sie eingeordnet wurde (das C#-Beispiel hierzu finden Sie ebenfalls auf der CD-ROM vor). Beim Aufruf von einem lokalen Laufwerk aus sind die folgenden Programmzeilen aber in jedem Fall erfolgreich:

```
Dim ev As Evidence = [Assembly].
                        GetExecutingAssembly().Evidence
Dim ps As PermissionSet = SecurityManager.
                                    ResolvePolicy(ev)
TextBox1.Text = ps.ToString()
```

Sie müssen auch nicht extra diesen Weg gehen – Sie können dies über die .NET Framework Configuration auch visuell über den Dialog *Evaluate an Assembly* erledigen. Abbildung 2 zeigt, dass beide Wege zum selben Ziel führen.

Sie sehen, dass es für uns als Entwickler nun nicht mehr ausreicht, wenn sich unser mühsam eingetippter Quelltext vom Compiler übersetzen lässt und sogar auch noch das Gewünschte tut, wenn das Programm direkt aus der Entwicklungsumgebung heraus gestartet wird. Stattdessen müssen wir immer dann zusätzliche Vorkehrungen treffen, wenn das Programm nicht nur von den lokalen Laufwerken gestartet werden soll. Zum einen müssen wir das Verhalten unserer Anwendung unter eingeschränkten Berechtigungsmengen testen, um die gewonnenen Erkenntnisse im zweiten Schritt bei der Weitergabe der Anwendung zu berücksichtigen.

Jede Menge Sandkästen

Der Test unter eingeschränkten Berechtigungsmengen bedeutet nun nicht, in diesem Fall ständig online sein zu müssen, um

die Assembly von einem echten Internet-Host starten zu können. (Es lohnt sich also nicht, allein deshalb den Aktienanteil von Telekommunikationsfirmen im Portfolio zu erhöhen.) Sie können alle Gruppen auch dann erfolgreich testen, wenn Sie mit einem Notebook arbeiten, das noch nicht einmal mit einem anderen Rechner verbunden ist. Am Beispiel der Gruppe *LocalIntranet_Zone* spiele ich diesen Fall einmal durch, wobei das System so konfiguriert werden soll, dass automatisch alle Assemblies dieser Gruppe zugeordnet werden, die von dem lokalen Festplattenverzeichnis *C:\Artikel\Sandbox_Intranet* geladen werden.

Da sich alles auf dem Policy Level *Machine* abspielt, lege ich dort zuerst unterhalb des Wurzeleintrags *All_Code* über die rechte Maustaste eine neue Code Group mit der Bezeichnung *SandboxIntranet* an (siehe Abb. 3). Der dafür zuständige Dialog führt mich schrittweise durch die Einstellungen, wobei ich auf der zweiten Dialogseite festlegen muss, über welches Merkmal die CLR eine Assembly dieser neuen Gruppe zuordnen soll. Aus der Liste wähle ich dazu den Eintrag *URL* aus und trage im daraufhin eingeblendeten Eingabefeld die Zeichenkette *file:///C:/Artikel/Sandbox_Intranet/** ein. Die CLR erhält über diese Syntax den Auftrag, automatisch alle die Assemblies, die aus diesem Verzeichnis heraus geladen werden, der Code Group *SandboxIntranet* zuzuordnen. Ich hätte aber auch das Merkmal *Strong Name* auswählen können, um die Assemblies nicht über das Verzeichnis, sondern über den dort vorgefundenen öffentlichen Schlüsselteil zuzuordnen.

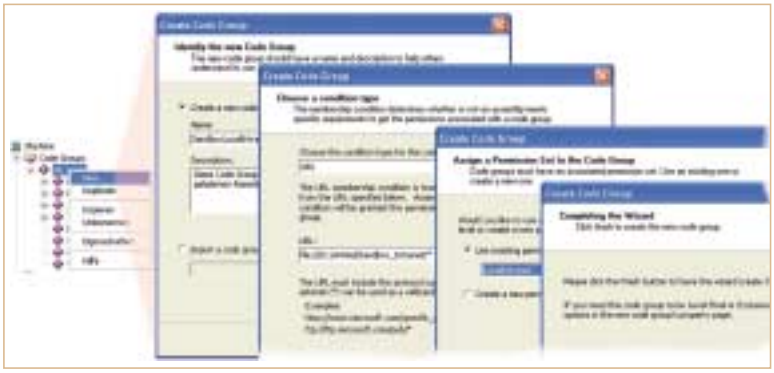

Abb. 3: Der erste Schritt beim Einrichten des Sandkastens

Wenn feststeht, wie die CLR eine Assembly einer Gruppe zuordnen soll, muss ich auf der dritten Dialogseite festlegen, welche Berechtigungsmenge diese Gruppe erhalten soll. Hier kann ich eine der vordefinierten Berechtigungsmengen (Permission Sets) auswählen oder ich stelle selbst eine neue zusammen. In meinem Beispiel wähle ich den vorhandenen Eintrag *LocalIntranet* aus. Damit ist der erste Teil abgeschlossen – allerdings funktioniert es noch nicht. Denn zur Zeit würde eine Assembly, die über das Verzeichnis *C:\Artikel\Sandbox_Intranet* ge-

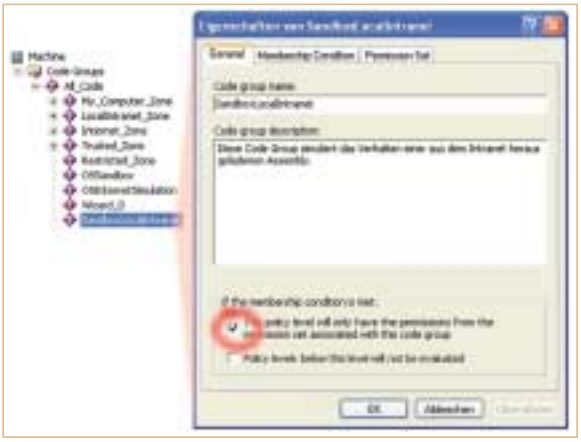

Abb. 4: Der zweite Schritt beim Einrichten des Sandkastens

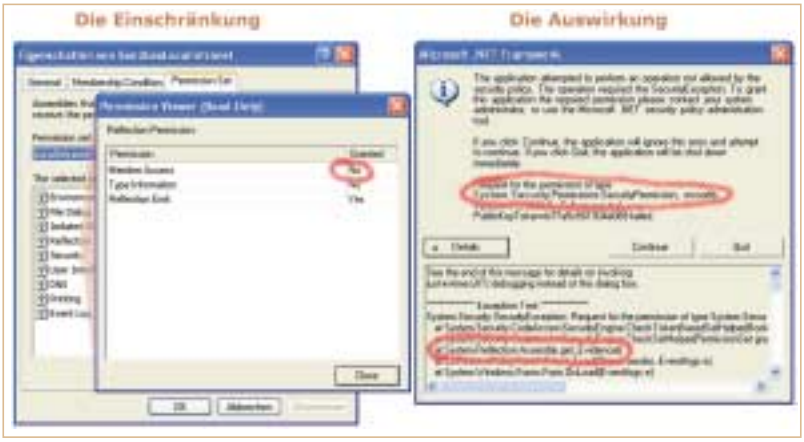

Abb. 5: Ursache und Wirkung

laden wird, den folgenden Code Groups angehören:

- *All_Code* mit der Berechtigungsmenge *Nothing*
- *My_Computer_Zone* mit der Berechtigungsmenge *FullTrust*
- *SandboxIntranet* mit der Berechtigungsmenge *LocalIntranet*

Ich muss also in einem zweiten Schritt dafür sorgen, dass nur die Regeln der Code Group *SandboxIntranet* gelten. Dazu rufe ich den Eigenschaftsdialog dieser Code Group auf, um dort die obere Checkbox (siehe Abb. 4) anzukreuzen und somit den *PolicyStatementAttribute*-Wert *Exclusive* zu aktivieren. Dieses Attribut legt fest, dass die betroffene Assembly keine Berechtigungen aus anderen Code Groups erhält. Die so gekennzeichnete Code Group überschreibt somit die Berechtigungsmengen aller anderen Code Groups für die betroffene Assembly und ist somit ein sehr effektiver Weg, die Rechte eindeutig zu verringern. Die CLR löst beim Laden einer Assembly immer dann eine Exception aus, wenn eine Assembly in mehrere Code Groups einsortiert werden kann, die alle das *Exclusive*-Attribut setzen.

Der Sandkasten mit den eingeschränkten Rechten eines Intranetzugriffs ist somit fertig. Bevor ich nun eine Assembly in dieses Verzeichnis kopiere und starte, schaue ich mir zur Kontrolle zuerst die übrig gebliebenen Berechtigungsmengen an. Abbildung 5 zeigt das Ergebnis am Beispiel der Reflektions-Rechte, wobei dort auch die Auswirkung dieser Einschränkung zu sehen ist. Wenn ich das vorhin vorgestellte Beispielprojekt *VBResolvePolicy.exe* in das Verzeichnis *C:\Artikel\Sandbox_Intranet* kopiere, kann ich die Anwendung problemlos starten. Jeder Menüaufruf führt jedoch sofort zu einem Sicherheitsveto der CLR in Form einer *SecurityPermission*-Exception, da die Anwendung über Reflektion bestimmte Daten auslesen will, die in diesem Sicherheitskontext nicht abrufbar sind.

Um zu prüfen, ob sich das Sicherheitssystem der CLR tatsächlich in der Praxis so verhält, wie es von Microsoft in der Theorie beschrieben wird, können Sie im Eigenschaftsdialog die Checkbox für das *Exclusive*-Attribut wieder abwählen. Nachdem der Dialog geschlossen wurde, ist auch die Programmfunktion wieder uneingeschränkt verfügbar – da nun die dritte beteiligte Code Group, *My_Computer_Zone*, mit ihrer uneingeschränkten Berechtigungsmenge wieder greift.

Sonstige nützliche Wizards

Das .NET Framework sieht verschiedene Alternativen für die Sicherheitskonfiguration vor. Der Grund dafür liegt primär darin, dass niemandem vorgeschrieben werden soll, wer für die Sicherheit im Netzwerk zuständig ist. Im Worst Case erlaubt der Administrator keine Einflussnahme von Außen, sodass er selbst Hand anlegen muss, damit der Anwender mit unserem Programm arbeiten kann.

Der *Trust an Assembly*-Wizard dient dazu, ohne spezielle Kenntnisse den Level des Vertrauens für eine Assembly (oder für die Assemblies eines Herstellers) zu erhöhen. Der Wizard kann die Rechte nicht verringern, somit ist er aus der Sicht des Entwicklers harmlos. Die Assembly für den Trust an Assembly-Wizard kann auf drei verschiedenen Wegen ausgewählt werden:

- Über den *Datei öffnen*-Dialog (lokale Laufwerke) festlegen.
- Die Aufruf-URL von Hand eintragen.
- Den UNC-Pfad von Hand eintragen.

Alle drei Wege führen zum selben Ziel, da keine Information über den Aufrufweg gespeichert wird, sondern nur der Strong Name (falls vorhanden), das Publisher Certificate (falls vorhanden) oder der Assembly-Hashwert ausgewertet werden. Somit können Sie eine Assembly auf der lokalen Festplatte auswählen und konfigurieren, aber später diese Assembly über das Intranet aufrufen – trotzdem greifen die erhöhten Berechtigungen, die allerdings erst beim nächsten Laden der Assembly wirksam werden.

Immer dann, wenn die Rechte für eine ganze Zone erhöht werden sollen, ist der *Adjust .net Security*-Wizard besser geeignet. Der zweite Wizard ist dafür zuständig, die Berechtigungen für alle Assemblies einer ganzen Zone zu setzen. Somit ist dieser Wizard „gefährlich“, da zum Beispiel das Zuweisen von *FullTrust* an die Internet-Zone zu einem völlig offenen System führt, aber auch der Entzug aller Rechte möglich ist.

Im dritten Teil dieser Artikelreihe stelle ich Ihnen Beispiele vor, um die Hintergründe zu den Begriffen *Link Demand* und *Security Stack Walk* zu verdeutlichen.