

Stopping the Barbarians at the Gate: .NET Code Access Security

Michael Stiefel

co-author “Application Development Using C# and .NET”



Reality For .NET Components

- Code will Eventually be Exposed to the Outside World.
 - Can Never Assume Technology resident on one Machine Only.
- Any .NET Assembly is Potentially a Component
 - No COM-like or Windows DLL infrastructure
- Easy to expose through Web Services or .NET Remoting.
- Developers Need to Understand Security

Key Security Ideas

- Security is a requirement just as functionality or performance.
 - Must be considered in the initial analysis and design.
- Threat Assessment
 - Security is based on Risk Management.
 - Most secure system is the disconnected system.
- Principle of Least Privilege
 - Understand what privileges your application needs.
 - Don't run as user with administrator privileges.

Preventing Unauthorized Access

- Authentication = Who are you?
 - Spoof Identity
- Authorization = Can you do what you want?
 - Access Data
 - Alter Data
 - Elevation of Privilege or “Luring attack”

Traditional Approach = Identity Management



Monolithic application whose behavior can be controlled based on user identity.



A clear entry point into the application where identity can be checked.

What is Wrong With This Model?

- Identity is Often Unknown or Not Well Defined
 - Public Access to Web Applications or Web Services
 - Run as Generic Identities
- Applications are Built from Components
 - Components/Web Services Come From Different Vendors
 - Can be Dynamically Linked
 - Different Vendors Have Different Trust Levels
 - Different Vendors Have Varied Code Quality

Code Access Security (CAS)

- Uses Identity of Code instead of User.
- Authentication: What Code Is Running?
- Authorization: What Rights does the Code Have?
- Works with Operating System Security.
 - Since the CLR is a virtual execution environment it does not replace the underlying operating system.

Sidebar: Application Domains

- Application Domains allow more scalable application isolation than process isolation.
 - Processes are inefficient for really scalable solutions (tens to hundreds of thousands).
 - Process switches are slow.
 - Interprocess communication is much slower than intraprocess communication.
- ASP.NET uses application domains.
- AppDomain class
- Each .NET application has at least one App domain.

Assemblies

- Assemblies are the unit of code deployment.
 - Dependencies and Type Definitions defined in its Manifest.
 - Can have a unique name based on version and culture.
- Code Rights are assigned to Assemblies and Application Domains.
- Assemblies are loaded into application domains.

Strong Names

- A Strong Name uniquely names an assembly.
- An assembly name has four parts: friendly name, version, culture, and publisher.
 - Friendly name usually non-extension part of filename.
 - Code assemblies must have neutral culture.
- Public/Private Key technology used to build strong name.
 - Private key uses friendly name, version, and culture to create a digital signature by signing assembly.
 - Public key in manifest used to check signature.

Security Evidence

- Code rights can be assigned based on assembly characteristics called Evidence.
- Examples:
 - URL the code was downloaded from.
 - Strong Name of the Assembly
 - Publisher of the Assembly (X.509 certificate)
 - Zone of execution (local computer, Internet)
- Can define your own Evidence.

Security Evidence Example

```
Evidence ev = AppDomain.CurrentDomain.Evidence;
IEnumerator iEnum = ev.GetEnumerator();
bool bNext = iEnum.MoveNext();
while (bNext == true)
{
    object x = iEnum.Current;
    Type t = x.GetType();
    if (t == typeof(System.Security.Policy.Zone))
        ...
    else if (t == typeof(System.Security.Policy.Url))
        ...
    else if (t == typeof(System.Security.Policy.Hash))
        ...
    else if (t == typeof(System.Security.Policy.StrongName))
        ...
    bNext = iEnum.MoveNext();
}
```

CLR Hosts

- A CLR Host is an unmanaged executable that starts an instance of the Common Language Runtime.
 - ASP.NET is an IIS ISAPI DLL that is a CLR Host
- A Host can provide additional evidence when it starts up an application domain.
- We will discuss host supplied evidence later.

CLR Host Startup Code

```
CComBSTR bstrVer(L"v0.0.0000");
CComBSTR bstrCLRType(L"wks");
CComPtr<ICorRuntimeHost> pHost;

HRESULT hr = CorBindToRuntimeEx(NULL, bstrCLRType,
    STARTUP_LOADER_OPTIMIZATION_MULTI_DOMAIN |
    STARTUP_CONCURRENT_GC, CLSID_CorRuntimeHost,
    IID_ICorRuntimeHost, (void **)&pHost);

...
pHost->Start();
CComPtr<IUnknown> punkDefaultDomain;
hr = pHost->GetDefaultDomain(&punkDefaultDomain);

...
CComPtr<_AppDomain> pDefaultDomain;
hr = punkDefaultDomain->QueryInterface(__uuidof(_AppDomain), (void **)&pDefaultDomain);

...
```

Security Permissions

- Permissions are the rights that are managed by the security system.
- Permissions can be granted or denied.
- A Permission Set is the list of rights that are granted or denied to a particular assembly.
- You can define your own permissions beyond those that ship with the CLR.

Sample Permissions

- FileIO Permission governs access to the File system.
- User Interface Permission controls the rights to put up Forms and access the clipboard.
- File Dialog Permission controls the rights to put up open and save dialogs.

Permission Demands

- You “demand” a permission to see if you have it.
 - Demand is a misleading term since you are really inquiring, not demanding.

```
FileIOPermission fileIOPerm =  
    new FileIOPermission(FileIOPermissionAccess.AllAccess,  
                          fileWithFullPath);  
fileIOPerm.Demand();
```

- Often the FCL will demand a permission for you.
- You also demand a set of permissions.

Security Policy

- The CLR uses Security Policy to determine what permissions an assembly is allowed or denied.
- Trusted code is required to inquire of the CLR if the executing code has the permissions to perform an operation.
- Normally, all the assemblies that have methods on the call stack must have the required permissions.
 - Otherwise a “luring attack” would be possible.

Verifiable vs. Managed Code

- For Security Policy to be effective, the behavior of code must be predictable. Only *verifiable* code can be the subject of security policy.
 - The JIT compiler verifies code as it compiles.
 - You can also PEVerify to verify IL code.
- Managed Code uses CLR facilities such as garbage collection.
- C++ can produce managed code, but not verifiable code.
 - By default, the permission to call unmanaged code is allowed only for code loaded from the local file system.

Code Groups

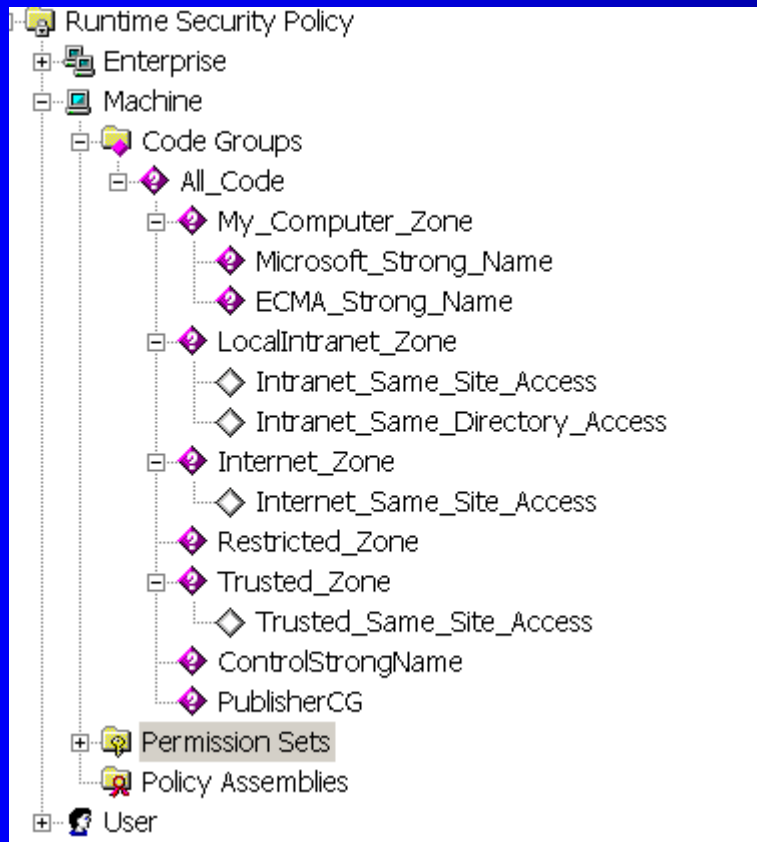
- A *code group* defines a set of assemblies that satisfy a membership condition. Membership conditions are defined using evidence.
 - The My_Computer_Zone code group consists of all assemblies loaded from the local computer.
- Associated with each code group is a *named permission set* that lists the permissions granted to members of that code group.
 - The Execution Permission Set only contains the Execution permission.

Policy Levels

- Security Policy is defined on four levels: Enterprise, Machine, User, and AppDomain.
 - Enterprise applies to all machines in an Active Directory installation.
- Enterprise, Machine and User policy levels are loaded from an Xml config file.
 - Use mscorcfg.msc or caspol.exe to edit.
- AppDomain policy is set via the AppDomain.SetAppDomainPolicy method.

Policy Level Hierarchy

- Each policy level is a hierarchy of code groups.



Policy Resolution

- At each level the set of permissions is the union of all the code groups the assembly is a member.
 - If membership condition fails, no lower levels evaluated.
 - Can prevent evaluation of lower policy levels
 - Can mark one code group as exclusive among siblings.
- The Security Policy is the intersection of all permissions of all the levels.
 - A specific permission must be allowed on all levels. By default Enterprise, User and AppDomain levels allow full trust. On the Machine level only My_Computer_Zone has full trust.
 - No way to deny a specific permission.

Built-In Permission Sets

- Nothing
- FullTrust
 - No list, all permissions that implement IUnrestrictedPermission
- Everything
 - Explicit list of permissions granted, by default allows all built-in permission types except SkipVerification.
- SkipVerification
- Execution
 - Can Execute
- Internet
- LocalIntranet

Modifying Security Policy

- Demo of MMC-Snapin
- Simple program puts up a UI Window
- Create a Named Permission Set with UI Permission
- Associate PermissionSet with an exclusive Code Group based on Assembly Strong Name
- Remove UI Permission from Permission Set
- Rerun Assembly
- Restore Permission, rerun Assembly

Declarative vs. Imperative Requests

- Permission requests can be made declaratively or imperatively.
 - Attributes do not allow for dynamic permissions or exact control over Exceptions thrown.

```
[UIPermission(SecurityAction.Demand, Window = UIPermissionWindow.AllWindows)]  
public static void Main()  
{
```

```
    UIPermission perm;  
    perm = new UIPermission(UIPermissionWindow.AllWindows);  
    perm.Demand();
```

Stack Walk

- A Demand causes a stack walk.
- This stack walk checks the permissions for every assembly that has code on the stack.
 - A SecurityException is thrown if one assembly does not have the requested permission.
- This stack walk starts at the frame above the current one.
- An Assert can be used to stop a stack walk.
 - Every use of Assert requires a code review because you are trusting every caller of your code!

Partially Trusted Callers

- Assemblies that are strongly named have to call assemblies that are strongly named.
 - Otherwise version policy would be indeterminate.
 - APTCA sample demonstrates this.
- Now let us change the permissions on the main assembly from FullTrust.
- Code fails to run!
- Since the main assembly is not fully trusted, it cannot call a strongly named assembly.

Allow Partially Trusted Callers

- Only fully trusted assemblies can call strongly named assemblies such as the standard .NET assemblies!
- Every assembly must be fully trusted!? All should have the managed code permission which can be used to circumvent CAS security!?
- You can avoid this by marking your assembly with the AllowPartiallyTrustedCallers attribute (APTCA).

Implications

- With the APTCA attribute the caller does not have to be fully trusted.
- By marking your assembly with the APTCA attribute you are attesting that there are no security flaws in your code.
 - This is particularly true for code that you put in the GAC.

System Assemblies with APTCA

- Currently not all System assemblies are marked with APTCA.
- In Version 1.0 (among others)
 - Mscorlib, System.dll, System.Data.dll
 - System.Windows.Forms.dll, System.Web.Services.dll
- In Version 1.1 (among others)
 - System.Web.dll got APTCA
- If you make a call to a strongly named assembly and get a SecurityException, suspect an APTCA issue.

Scenario #1: Third Party Components

- Third-party components have less trust than your own software.
- Define an interface used by these components.

```
public interface IUserCode
{
    int PotentialRogueCode();
}
```

- You invoke the interface:

```
public void OurCode(IUserCode code)
{
    ...
    int v = code.PotentialRogueCode();
    ...
}
```

- Remove Assert permission from untrusted software.

Denying Permissions

```
UIPermission uiPerm = new UIPermission(PermissionState.Unrestricted);  
FileIOPermission fileIOPerm = new  
    FileIOPermission(PermissionState.Unrestricted);
```

```
PermissionSet ps = new PermissionSet(PermissionState.None);  
ps.AddPermission(uiPerm);  
ps.AddPermission(fileIOPerm);
```

```
ps.Deny();
```

```
int v = code.PotentialRogueCode();
```

```
CodeAccessPermission.RevertDeny();
```

Scenario #2: Administrative Metadata

- Administrators can control what software runs in an enterprise or a machine.
- How can you communicate to the administrator what permissions your code requires?
- Use `RequestMinimum`, `RequestOptional`, and `RequestRefuse` attributes.

RequestMinimum

- RequestMinimum specifies permissions that are necessary for the assembly. If the security policy denies these permissions, the assembly is not loaded, and a PolicyException is thrown before your code is ever run.
- By default, no permissions are required.

```
[assembly:UIPermission(SecurityAction.RequestMinimum, Window =  
    UIPermissionWindow.AllWindows)]
```

RequestRefuse

- RequestRefuse actions specifies permissions that the assembly does not want.
- It might be easier to specify what is not wanted than what is needed.
- Permissions can be explicitly refused to avoid an assembly being used in unforeseen ways because of a security flaw.

[assembly:SecurityPermission(SecurityAction.RequestRefuse, UnmanagedCode = true)]

RequestOptional

- RequestOptional specifies permissions that are not essential for the assembly.
 - Functionality might be missing, but the assembly can provide its basic services.

[assembly:FileIOPermission(SecurityAction.RequestOptional, Unrestricted = true)]

- RequestOptional implicitly refuses permissions not in a RequestMinimum or a RequestOptional.
 - Implicit RequestRefuse for all other permissions except Execution permission.

Permview

- The Permview tool allows an administrator to find the request attributes.
 - `permview.exe assemblyname`
- The */decl* attribute allows you to find all declarative security attributes.

Secnario #3: Managed Control Hosting

- Host a managed control in Internet Explorer
 - Review major parts of Code Access Security
 - Compare with hosting a Windows Form Control
- When an application domain is created, the creator can override the assembly evidence.

```
AppDomain appDomain = AppDomain.CreateDomain("x", evidence);
```

The Control

- The control is a rich-edit control embedded in a form.
 - The control is colored blue, and you can type text into it.
 - The control has three methods: clear, save text, and load text.
- It is easy to put this control on a Windows form and associate three buttons with the functions.

Attempt #1: Embedding in IE

- Embed the control on an HTML page with three buttons.

```
<html>
  <body MS_POSITIONING="FlowLayout">
    <object id = "test" classid= "Control.dll#TestControl" height= 200 width=200
VIEWASTEXT></object>
    <p></p>
    <input type = "button" value = "Clear" onclick="test.Clear()"/>
    <input type = "button" value = "Save" onclick="test.Save()"/>
    <input type = "button" value = "Load" onclick="test.Load()"/>
  </body>
</html>
```

- Attempting to save data generates an exception because the control does not have the FileIO permission.

Local Intranet Zone

- Since the control is running in IE, and accessed via a WINS style URL, it is running in the LocalIntranet_Zone.
 - DNS names or IP addresses run in the Internet Zone.
 - Code running in this Zone does not have the FileIO permission.
 - When run in the Windows Form, the code ran in the My_Computer_Zone. Code running here has FullTrust.

What to do?

- Adding the FileIO Permission to the LocalIntranet_Zone.
 - malicious or buggy code could more easily attack your system.
- Elevating the permission associated with the URL .
 - An attacker could place a control with the same name at that URL for a “spoofing” attack.

Attempt #2: Embedding in IE

- The safest approach is to give the control a strong name and assign the FileIO Permission to a control with that strong name.
- We create a code group based on that control's name and assign it the FileIO Permission.
- Now when we run...

The Mysterious Security Exception

- The control fails to render, and the buttons do not seem to function, but no exception is generated.
 - Clicking on the buttons does generate an Error on page message on the bottom of the page.
- Have we made things worse?
- IE writes its error messages to a an html page error log in the Temporary Internet files folder.
 - There you will find that a SecurityException was thrown.
- When there is a mysterious SecurityException...

Attempt #3: Embedding in IE

- When IE creates the application domain, it uses the control's URL as evidence. Hence this application domain is not fully trusted.
- When this new application domain calls the strongly-named control, this will fail because a not fully trusted assembly cannot call a strongly named assembly.
- So...we now mark our control with the `AllowPartiallyTrustedCallers` attribute.
- After clearing the temp files, we try again...

Permissions and the Stack Walk

- The control renders, and the Clear button works.
- Once again we seem not to have the FileIO permission!
- Remember all callers on the stack frame must have the FileIO permission, and the application domain created by Internet Explorer does not.
- We must use the dreaded Assert for the FileIO permission. When we are done, we RevertAssert to prevent any possibility of a “luring attack.”
- Crossing our fingers...

Attempt #4: Embedding in IE

- It works!!
- Hosting example shows details of CLR Hosting and controlling evidence for a new application domain.

Summary

- Code Access Security allows you to control code, irrespective of the user identity that it runs as.
- CAS is based on evidence, permissions, and security policy.
 - Understanding how application domains and assemblies have their permissions granted is important in getting complicated scenarios to work.
 - In side-by-side scenarios remember to use the right version of the configuration files and utilities.
- Remember the Principle of Least Privilege.
- Friends Don't Let Friends Run as Administrator.