

Introduction to Evidence-based security in .NET Framework

Brad Merrill
Program Manager
.NET Frameworks Integration

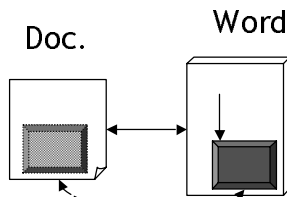
Agenda

- **The Problem: Customer Scenarios**
- **The Solution: .NET Security**
- **Role-based Security**
- **Evidence-based Security**
- **Demos**

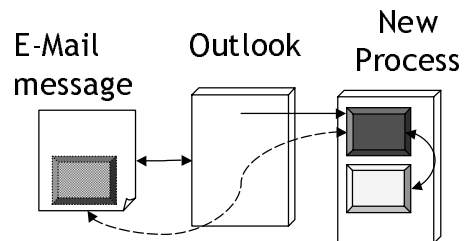
Defining the Problem: Customer Scenarios

Scenario #1 – Active Content

*Word Macro
(In-process)*

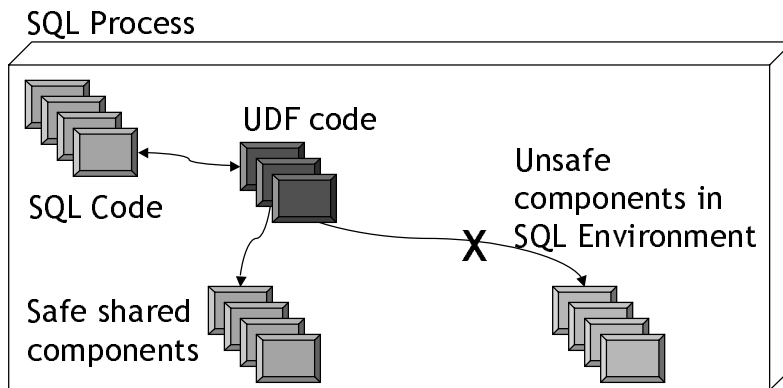


*E-Mail attachment
(Out-of-process)*

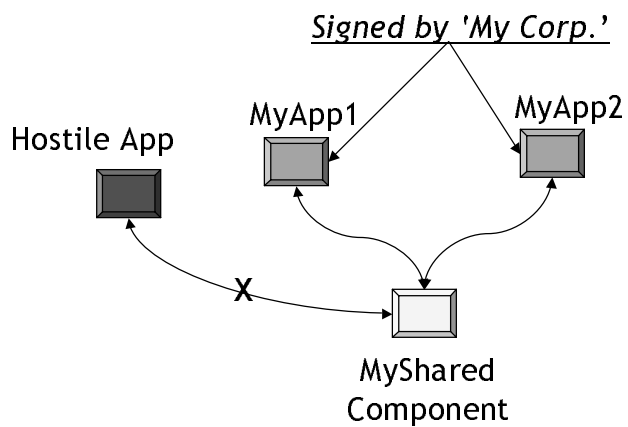


Scenario #2 – App Extensibility

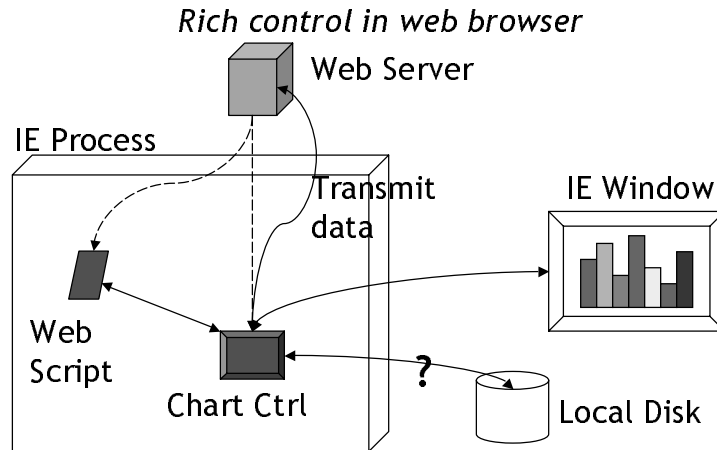
SQL server: UDFs (user defined functions)



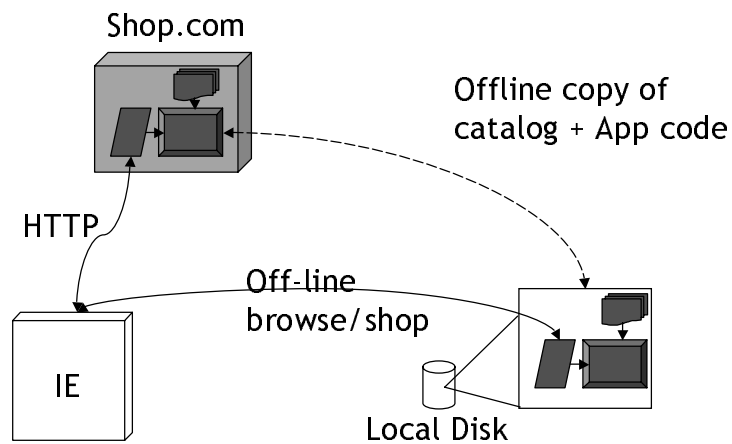
Scenario #3 – Controlled Sharing



Scenario #4 – Rich Web Apps

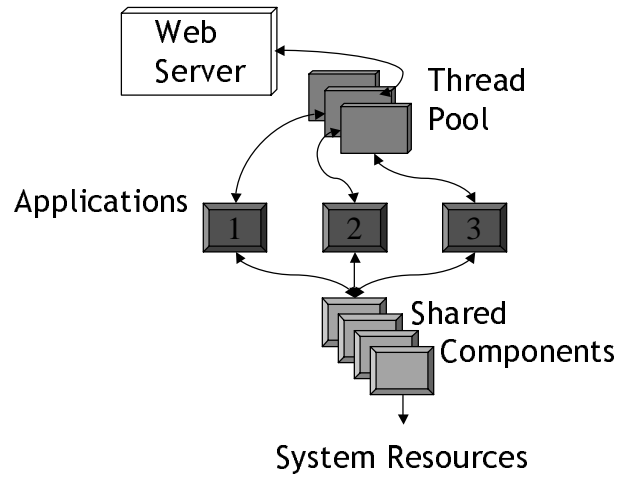


Scenario #5 – Off-Line Application



Scenario #6 – Web Hosting

Service provider hosting 3rd party apps



**The Solution:
.NET Security**

Background

- **.NET Security model is a combination of 2 elements**
 - **Role-based security**
 - **Evidence-based security**

Role-based Security

Common Terms

- **Role-based Security**
 - Making Authorization decisions based on the identity and/or role(s) of the entity on whose behalf an application is executing.
- **Identity**
 - Distinguishing characteristic of the entity on whose behalf an application is executing. Commonly a user or account name.
- **Principal**
 - The encapsulation of Identity and role information—everything you need to know about an entity in order to make Authorization decisions.


Role-based Security Infrastructure

- **The CLR provides an infrastructure for managing identity and role information.**
 - A Host authenticates the user and provides the identity and role information to the CLR
 - The CLR makes that information available to code via APIs and permission demands (both imperative and declarative)
 - Example: ASP.NET

Using the Role-based APIs

- **Example: checking a user's NT group membership**
 - Must specify Windows authentication (requires SecurityPermission)
 - Principal is accessed from the Thread object


```
public bool IsAdministrator() {  
    //Default principal is unauthenticated  
    //We must tell the system we want to use Windows auth  
    AppDomain.CurrentDomain.SetPrincipalPolicy  
        (PrincipalPolicy.WindowsPrincipal);  
  
    WindowsPrincipal user =  
        Thread.CurrentPrincipal as WindowsPrincipal;  
    return user.IsInRole("Administrators");  
}
```



Using the Principal permission

- **Example: a declarative demand to ensure the user is in the Administrator group**
 - Must specify Windows authentication (requires SecurityPermission)
 - Like demands for other permissions, a principal permission demand will throw an exception if it fails
 - Name, Role, or both can be supplied as named parameters

```
[PrincipalPermission  
    (SecurityAction.Demand, Role="Administrators")]  
public void ProtectedMethod() {  
    //does some operation reserved for Administrators  
}
```



Considerations

- **Lazy authentication: we don't create a principal object until you ask for it.**
- **The role-based security functionality in the CLR does not replace COM+ 1.0 Services security.**
 - **If your application contains both managed and unmanaged (COM) components, consider using COM+ 1.0 Services role-based security via the managed wrappers.**
 - **If your application is entirely managed, CLR role-based security may be just what you need.**

Common Terms

- **Authentication**
 - **Determining the identity of the party/entity making a request.**
 - **User authentication is generally by means of name/password verification.**
 - **Code authentication can be done by collecting evidence about the code: location of origin, digital signature, hash, etc.**
- **Authorization**
 - **Determining whether to honor a request made by an identified party/entity.**
 - **User authorization is generally done by business logic or by the system (NTFS access control lists, IIS security settings, etc.)**
 - **Code authorization is achieved via policy—analyzing evidence in order to grant appropriate permissions.**

Evidence-based Security

Evidence-Based Security

- **Permissions**
 - Objects that represent specific authorizations
- **Policy**
 - Determines what code is permitted to do:
set of permissions to grant to an assembly
- **Evidence**
 - Inputs to policy about code, from multiple sources
- **All three are fully extensible**

Permissions

- A permission object **represents a specific authorization, such as access to a resource**
 - *“permission to do something”*
- A permission grant **is an authorization given to an assembly (code)**
 - *“this code is authorized to do something”*
- A permission demand **is a security check for corresponding grants**
 - *“is something allowed?” (else, raise exception)*

Standard .NET permissions

- **Permissions for Framework resources**
 - These permissions represent access to protected resources.

<i>Data</i>	<i>Directory Services</i>	DNS
Environment Variables	<i>Event Log</i>	File Dialog
File IO	Web	Isolated Storage
<i>Message Queue</i>	<i>Performance Counters</i>	<i>Printing</i>
Reflection	Registry	Security System
Socket	UI	...

(Italicized permission are Beta 2)

Standard .NET permissions

- **Identity permissions**

- These permissions represent code identity. They are granted to code based on its corresponding evidence.

Publisher	URL
Site	Zone
Strong Name	

- **Other permissions**

- A user identity permission is also supported. This is the only non-code access permission in the Framework.

Principal (User Identity/Role)

Code Access Security

- **Most permissions are code access permissions**

- Demanding a permission performs a stack walk checking for related grants of all callers
- Support dynamic stack modifiers
- Two ways to make checks:
 - Imperatively (method implementation)
 - Declaratively (method metadata)

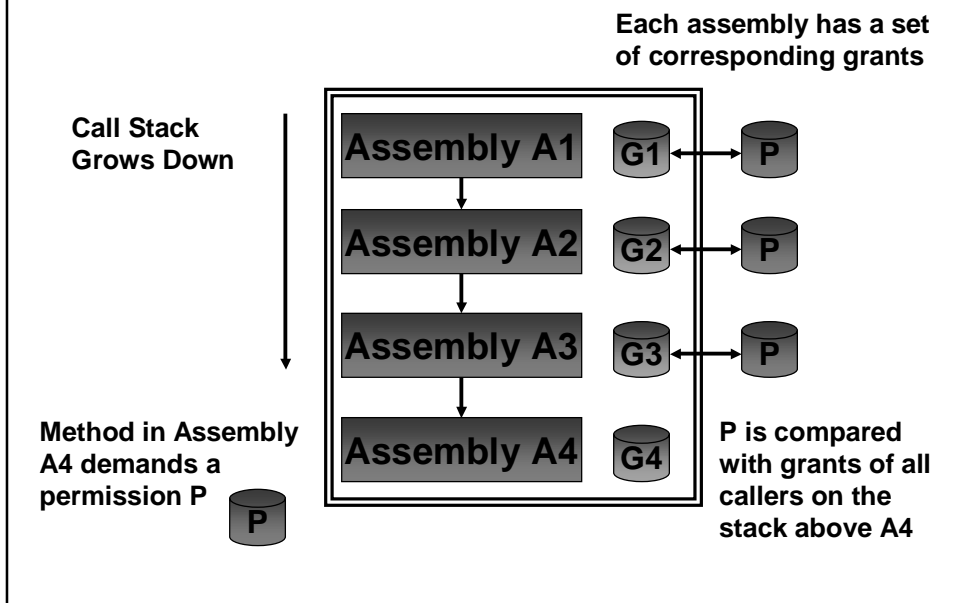
Normal application code view of security enforcement

- Most applications get security enforcement simply by calling the class libraries

```
try {  
    Environment.GetEnvironmentVariable ("USERNAME") ;  
}  
catch ( SecurityException se ) {  
    Console.WriteLine ("SECURITY EXCEPTION:" + se.ToString()) ;  
}
```

C#

Stack Walk Behavior



Stack Walk Modifiers

- **Modifiers provide fine-grained, dynamic control over state of grants on the stack**
- **Assertion**
 - “I vouch for my callers; checks for *perm* can stop at this frame”
- **Example: “Gatekeeper” classes**
 - **Managed wrappers for unmanaged resources**
 - Demand appropriate permission from caller
 - Assert permission to call unmanaged code
 - Make the unmanaged call

Imperative Security Checks

- **Example: the File object constructor**
 - Requires read access to the corresponding file

```
public File(String fileName) {  
    // Must fully qualify the path for the security check  
    String fullPath = Directory.GetFullPathInternal(fileName);  
    new FileIOPermission(FileIOPermissionAccess.Read, fullPath)  
        .Demand();  
    // [... read the specified file at behest of caller(s) ...]  
}
```

C#

Declarative Security Checks

- **Declarative security is**
 - Part of a method's metadata
 - Implemented with custom attributes
 - Processed by JIT

```
[FileIOPermission(SecurityAction.Demand, Read = "c:\\temp")]  
public void foo() {  
    // class does something with c:\temp  
}
```

C#


Controlling access to code

- **Identity permissions allow the same security checks on identity of code**
 - Digital signature, location (URL, site), etc.
- **Declarative security checks by JIT instead of (most costly) runtime checks**
 - LinkDemand: code reference by a caller
 - InheritanceDemand: subclass/overriding
- **Combination provides a tool for developers to control who uses code**

Controlling access to code (cont.)

- **Example: controlling access with a Strong Name identity link demand.**
 - Ensures that the immediate caller is signed with the given key and has the correct name and version.


```
[StrongNameIdentityPermissionAttribute
  (SecurityAction.LinkDemand,
   PublicKey="00240000048000009400000006020000...",
   Name="MyApp", Version="0.0.0.0")]
// Only MyApp can use this class
public class MyClass {
    ...
}
```



Controlling access to code (cont.)

- **Example: calling code that is restricted by a Strong Name check.**
 - Calling code must be signed with the private key corresponding to the public key used in the previous example.

```
[assembly: AssemblyKeyFileAttribute ("keypair.dat")]
[assembly: AssemblyVersionAttribute ("0.0.0.0")]
public class MyApp {
    ...
}
```

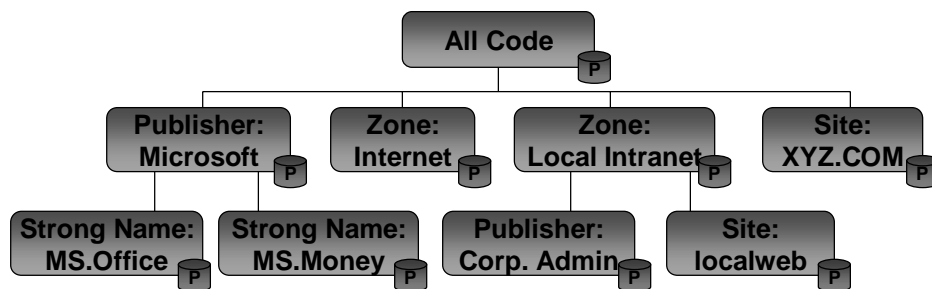


Policy

- **Policy is the process of determining the permissions to grant to code**
 - Permissions granted to code, not user
 - Grants are on a per-assembly basis
- **Multiple levels of policy**
 - Machine-wide, User-specific
 - Enterprise support: Group Policy (*Beta 2*)
 - Further policy restrictions allowed on a per-application domain basis

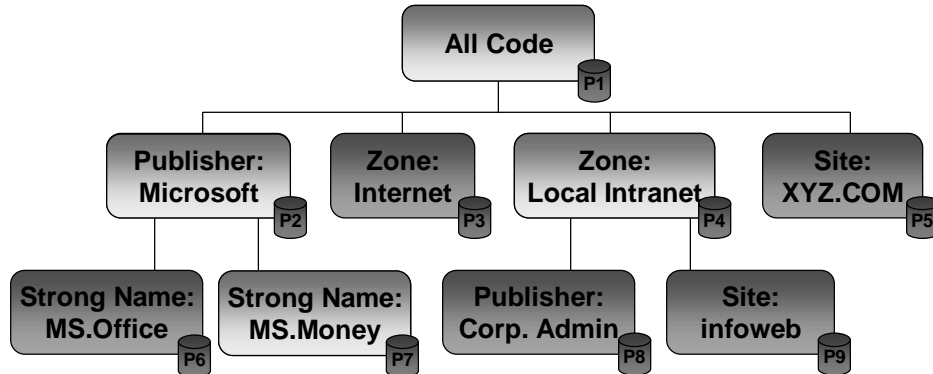
Inside policy

- **A policy level is a collection of code groups**
 - Code has identity in the runtime, just like users have identity in Windows NT®
 - Permissions are associated with each code group
- **Evidence determines group membership**
 - In the group, get granted the related permissions



Sample Policy Level

- **Example: MS.Money on Local Intranet**
 - Member of four groups (highlighted)
 - Granted permissions = $P1 \cup P2 \cup P7 \cup P4$



Default policy

(Beta 2)

- **Local Computer Zone**
 - Unrestricted
- **Intranet Zone**
 - Read environment variables (limited), UI, IsolatedStorage, Assertion, Web access to same site, File read to same UNC directory
- **Internet Zone**
 - Safe UI, IsolatedStorage, Web access to same site
- **Restricted Zone**
 - No authorizations, can't run
- **MS Strong Name (Frameworks Classes)**
 - Unrestricted

Evidence

- Evidence is the input to policy
- Example: Info about a code assembly
 - Shared names
 - Publisher identity } cryptographically strong
 - Location of origin (URL, zone, site)
- Evidence is completely extensible
 - Any object can be a piece of evidence
 - Only impacts grants if there is a code group membership condition that cares about it!

Host Control of Policy

- Hosts can influence policy
 - Hosts specify “implicitly trusted” evidence
 - Custom membership conditions can interface with other authorization systems
 - Example: ASP.NET/ISP application hosting
 - Semi-trusted hosting cannot provide evidence
- Hosts can limit policy for application domains they create
 - Example: SQL Server™ user-defined assemblies

Assembly Input To Policy

- **Assemblies can request permissions**
 - Three types of request: Minimal, Optional, Refuse
 - If policy does not grant everything in the “Minimal” set, assembly fails to load
 - Assembly is granted:
(MaxAllowed \cap (Minimal \cup Optional)) – Refused
- **Assemblies can carry evidence, too**
 - Assembly evidence is “initially untrusted”
 - Policy evaluates assembly evidence and decides whether to use it
 - Example: third-party certifications

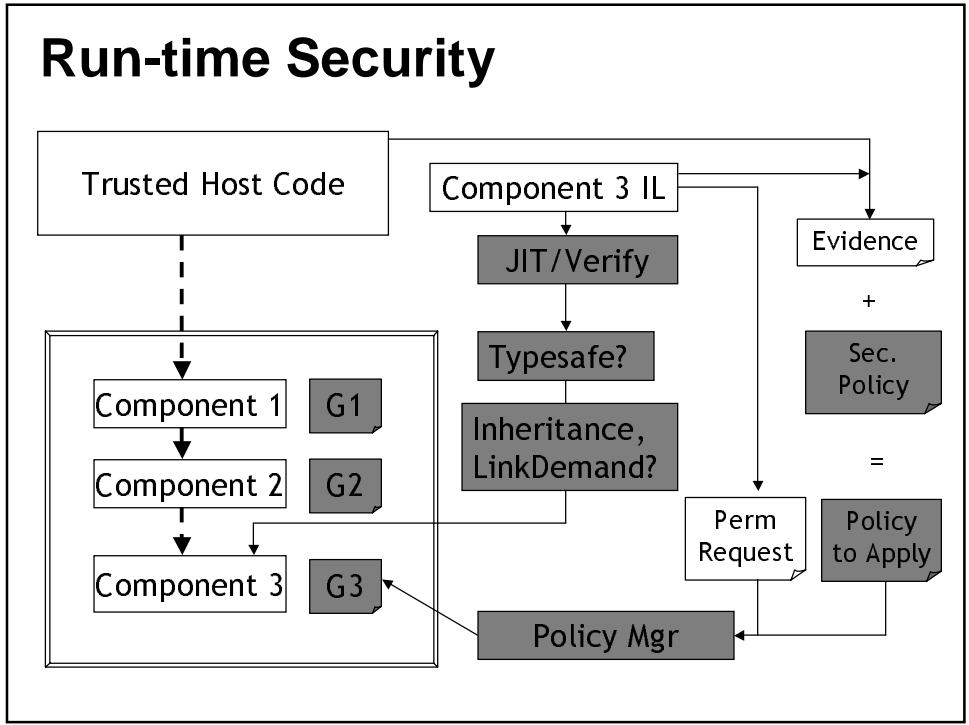
Sample Permission Request

- **Request (minimum,optional,refused)**
- **If none, code gets maximum policy gives**

```
[assembly:UIPermissionAttribute  
(SecurityAction.RequestMinimum,  
Window=UIPermissionWindow.SafeSubWindows)]  
  
[assembly:FileIOPermissionAttribute  
(SecurityAction.RequestOptional,All="C:\\")]  
  
[assembly:SecurityPermissionAttribute  
(SecurityAction.RequestRefused,UnmanagedCode=true)]
```

C#

Run-time Security



Summary

Great Customer Experience

- **End-user**
 - Managed apps just run, consistent experience for scripts, exes, controls
 - Safe defaults, no runtime trust decisions for users
- **Administrator**
 - All settings in one place, easy to customize
 - Understandable policy model (need Beta feedback)
 - Security administration tool coming in Beta 2
- **Developer**
 - Can focus on app logic, security comes for free
 - But, easy to use and extend when necessary (i.e., protecting a new shared resource)

Minimizing Security Flaws

- **Typesafe code**
 - Managed code verified for typesafety at runtime
 - Eliminates most common security problems
 - Buffer overrun attacks
 - Reading private state or uninitialized memory
 - Access arbitrary memory in process space
 - Transfer execution to arbitrary location in process
- **Developers can use 'least privilege'**
- **Code access security blocks most 'luring' attacks**

Questions?

Where do you want to go today?®

Microsoft®