

# **.NET Framework Security: Lessons Learned from Five Years of Shipping Partially-Trusted Code**

**Brian A. LaMacchia**

Software Architect  
Office of the CTO, Microsoft

Stevens/Columbia/IBM Research Security & Privacy Day  
November 14, 2005

# Agenda

- ❖ **Brief overview of the .NET Framework and its security system**
- ❖ **10 lessons learned from V1 & V1.1**
  - **How we addressed key deficiencies in V2**
- ❖ **Looking forward to V2.1, V3**
  - **Open problems**

# What is the .NET Framework?

- ❖ Microsoft's **cross-language** development platform
  - Execution environment/VM: Common Language Runtime (CLR)
  - Intermediate Language: MSIL (similar to bytecode)
  - Class libraries: “The Framework”
  - Language compilers (many, MS & 3<sup>rd</sup> party)
  - Development tools: “Visual Studio.NET”
- ❖ All “.NET languages” are first class players
  - Any language that compiles to MSIL can use and extend the .NET Framework
- ❖ CLR provides a secure execution environment for “partially-trusted code”

# **.NET Framework Timeline**

- ❖ **June 2000: V1 announced at 2000 Professional Developers Conference; V1 Beta 1 released to developers**
- ❖ **January 2002: V1 (“RTM”)**
- ❖ **April 2003: V1.1 (“Everett”)**
- ❖ **November 7, 2005: V2 (“Whidbey”)**
- ❖ **Service packs for V1 & V1.1 released along the way, too**

# Key Components of the Security System

- ❖ **Type safety and verification**
- ❖ **Permissions, demands and stack inspection**
- ❖ **Policy/trust management system for assigning permissions to assemblies**
- ❖ **Application deployment model**

# Type Safety & Verification

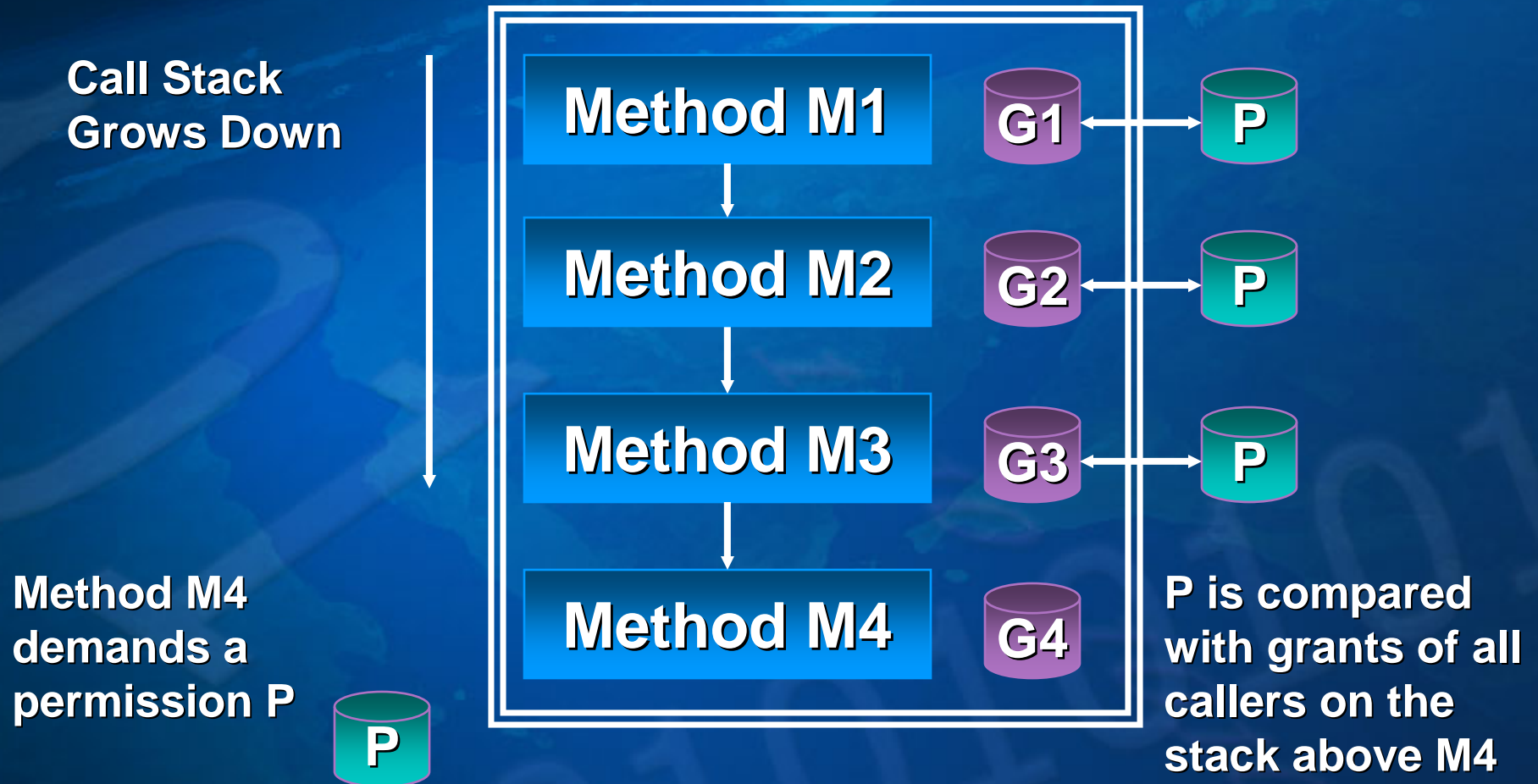
- ❖ CLR provides memory protection through type safety verification
- ❖ Every MSIL method is type-safe verified before being allowed to be called by another method
  - Verification happens as part of JIT compilation to binary
- ❖ The right to run unverifiable code is governed by a security permission (like any other privileged operation)

# Permissions, Demands & Stack Inspection

- ❖ A permission is a set (or subset) of capabilities with respect to a resource
  - Ex: FileIOPermission(READ, "c:\")
- ❖ Most permissions are **code-access permissions** and have stack-walking semantics
  - A demand for a code-access permission must be satisfied by grants to every stack frame above the demanding frame
- ❖ **Stack-walking is a defense against luring attacks**
  - Less-trusted code tricking more-trusted code into performing protected operations

# Stack-walking Semantics

Each method has a set of corresponding grants



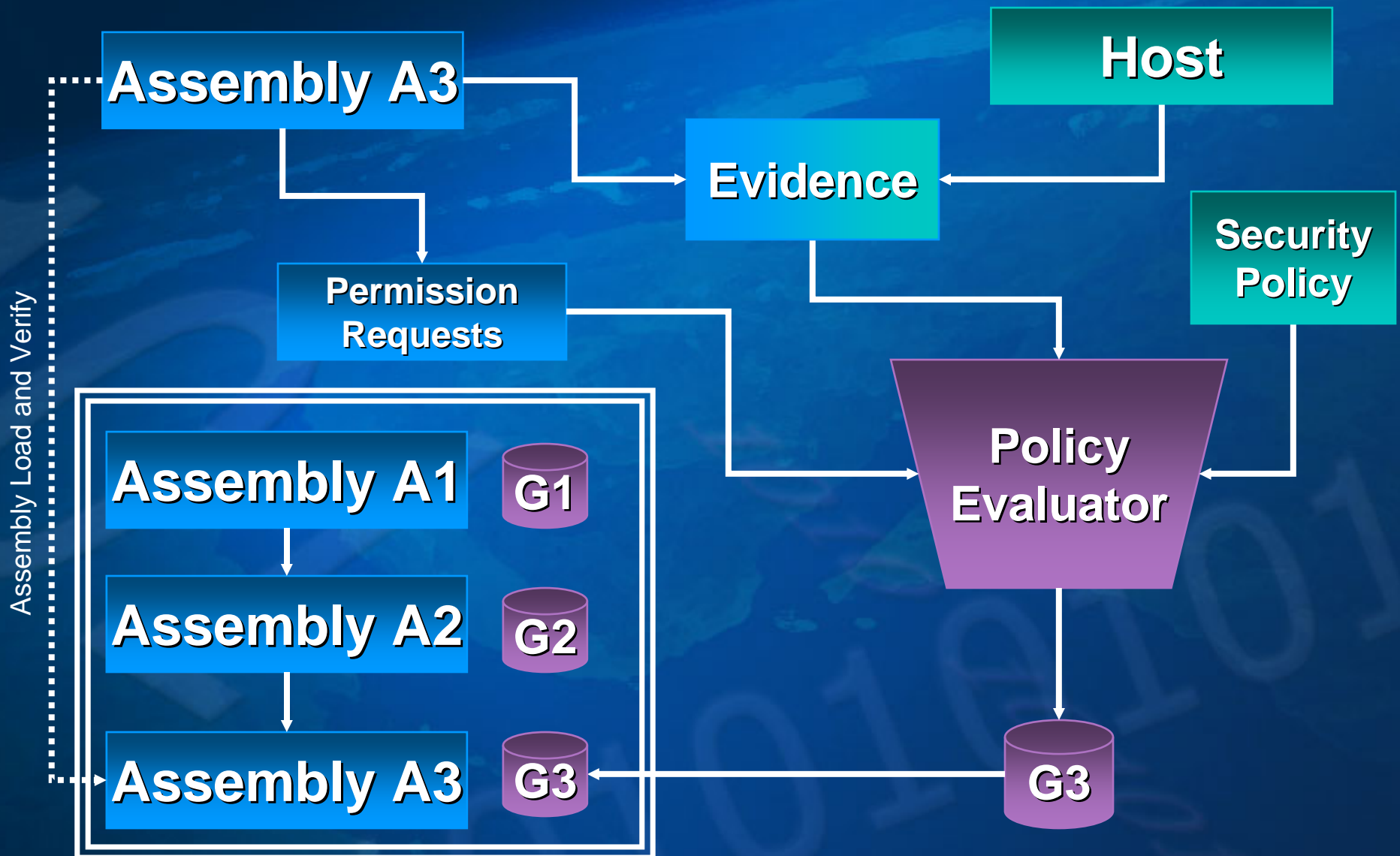
# Policy Evaluation in the CLR

- ❖ **Policy evaluation** is the process of determining the set of **permissions** to grant to code based on **evidence** known about that code
  - Evidence is typically info about a code assembly (e.g. publisher identity, URL)
  - We assign rights (permissions) to assemblies in the CLR, just as we assign rights to groups of users.

# Application Deployment

- ❖ **Assemblies are the core unit of code development and distribution**
  - **Similar to a .DLL or shared library**
- ❖ **Assemblies are the minimum code unit that have identity**
- ❖ **Permissions are granted on an assembly-wide basis**
  - **All methods in the same assembly have the same rights**
- ❖ **Applications are collections of assemblies dynamically assembled based on referenced classes**
  - **“Soup of objects” == “Soup of assemblies”**

# .NET Security in One Slide





# 10 Lessons Learned

# **Lesson #1: Know Your Customers (Developers, Admins & Users)**

# #1: Know Your Customers

- ❖ **No matter how great your security system is, it is useless if you users turn it off or ignore it**
- ❖ **Some common reasons why they might do that**
  - **Too hard to develop in the security model.**
  - **The system is too hard to administer; can't easily figure out the correct security policy or map it to real-world requirements**
  - **The system is too restrictive – it doesn't let them get their work done**

# #1: Know Your Customers (2)

## ❖ Developers

- Some security checks could be performed declaratively (attributes on methods), but not all of them
- Education and training needs to be factored into the equation

## ❖ Admins

- Administration tools are amazingly important, but they tend to be left to the end of the project

## ❖ End users

- Need to get more than “dancing hippos” if you want broad acceptance

# Lesson #2: Know Your Own Developers

## #2: Know Your Own Developers

- ❖ In particular, understand their strengths and limitations
- ❖ Can all of your developers understand and internalize the security model?
  - The greatest security model in the world will be ineffective if the developers who have to use it aren't trained properly to understand it.
  - Not every developer needs to be able to write security-critical code
- ❖ Developer education is key
  - Has to be early enough in the product cycle to be meaningful

# **Lesson #3: It's Worth Being Paranoid About the Verifier**

# #3: Verifier Paranoia

- ❖ **Correctness of the MSIL type-safety verifier was a top concern during early development of the CLR**
  - Many of the early attacks against the JVM were exploits of the Java verifier
- ❖ **How do you verify the correctness of a verifier?**
  - Build two verifiers, preferably one in a “formal” language like ML, and compare
  - Better yet, build three verifiers and compare each against the other two



# **Lesson #4: Visibility is not Security**

# #4: Visibility is not Security

- ❖ **Object-oriented languages have notions of visibility**
  - **Ex: private, friend, protected, public**
- ❖ **Developers & users get confused and tend to think of these visibility rules as security guarantees**
  - **They're not in the .NET Framework**
  - **We needed to educate developers better to the differences**

# **Lesson #5: Users Confuse Identity with Trust**

## #5: Users Confuse Identity with Trust

- ❖ **The .NET Framework introduced the concept of strong names as a means for identifying code assemblies**
  - **Strong names use public keys as namespace prefixes; similar to SDSI names**
  - **Strong names are only valid if signed with the corresponding private key**
- ❖ **Strong names are an identity mechanism, but not a trust mechanism**
  - **Nevertheless, many users & admins saw a “crypto-protected name” and started writing policies that depended on them, without a formal trust mechanism**



# **Lesson #6: It's Hard to Hide Secrets in a Garbage- Collected Environment**

## #6: GC makes it hard to reliably erase secrets from memory

- ❖ We teach developers that they don't have to explicitly manage memory in a GC'd environment, but that's not quite true
  - They have to worry about manually clearing sensitive data as soon as they are done with it
  - We don't want the GC to ever copy sensitive objects, but pinning objects hurts perf
- ❖ V2 feature: SecureString
  - Specialized class for handling secrets like passwords
  - Proper use has impact throughout the class libraries (e.g. GUI, XML processing, etc.)



# **Lesson #7: Async is Important**

# #7: Async is Important

- ❖ **Asynchronous design patterns are becoming more and more important**
  - **Ex: Posting tasks to thread pools, async I/O, message passing, distributed computation**
- ❖ **Execution context of the callee isn't necessarily that of the caller**
  - **But stack inspection assumes it is**
  - **Naively flowing security context information across async calls is expensive (slow)**
    - **Doing it fast is complex**

# #7: Living with Asynchronicity

- ❖ If asynchronous programming patterns become predominant, alternatives mechanisms for providing fine-grained access control become more attractive
  - Data tainting
  - Information flow-control
  - Execution history-based techniques (Abadi & Fournet, NDSS'03)

# **Lesson #8: Get the App Model Right**

# #8: Get the App Model Right

- ❖ A single API set/framework/SDK can support multiple application models
- ❖ The corresponding security system has to work for all of these models
  - But it should best support and facilitate the model that developers use most
- ❖ CLR V1 security was aimed at rich clients built from a “soup of objects” from multiple sources
  - But it turns out most of our developer customers don’t do this – they write “single-source” applications

# The “ClickOnce” Model

- ❖ An “application-based model” built on top of V1 security primitives
- ❖ Key features added for V2
  - An “application” now has an identity of its own
    - Collection of assemblies + metadata, defined by an XML manifest
    - Applications are self-describing
  - The application becomes the common unit of code deployment, not the assembly
- ❖ Applications that run “in the sandbox” with the default set of permissions “just run”
- ❖ Applications that require “elevated” permissions (more than the default) have to declare additional necessary grants in the manifest
  - User consent required to run, but decisions can be persisted based on code signer identity

# **Lesson #9: Development, Debugging and Deployment Environments Need to Match**

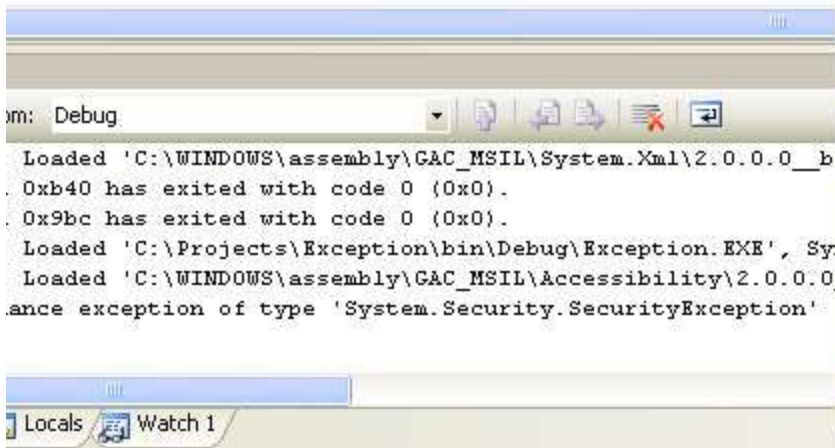
## #9: Make Sure Dev, Debug & Deploy Environments Match

- ❖ If we want developers to write partial-trust code, then they must be able to test & debug their code in a partial-trust environment
  - We also have to give them tools to help determine whether their code can exist in a limited execution context
- ❖ This was difficult to do in V1
  - Dev tools ran full-trust by default
  - Limited source code analysis tools

# Additional V2 Features to Improve the Partial Trust Development Environment

- ❖ **Better security error information**
  - “Precisely what failed and why?”
  - Failed assembly info, its permission grant set, attempted security action, demanded permission set
- ❖ **Code analysis tool to help determine what permissions an assembly requires to run**
  - Helps determine if an app fits into the default sandbox
  - Outputs estimate of minimum set of permissions required to run application
- ❖ **“Debug in Zone”**
  - Debug applications in different CAS security contexts

```
private void button1_Click(object sender, EventArgs e)
{
    FileStream fs = new FileStream("C:\\boot.ini", FileMode.Open);
}
```



**SecurityException was unhandled**

Request for the permission of type 'System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0, PublicKeyToken=b77a5c561934e089' failed.

**Troubleshooting tips:**

- Store application data in isolated storage.
- When deploying an Office solution, check to make sure you have fulfilled all necessary security requirements.
- Use a certificate to obtain the required permission(s).
- If an assembly implementing the custom security object references other assemblies, add the referenced assemblies to the project.
- Get general help for this exception.
- Search for more Help Online...

**Actions:**

- Add Permission to the project
- View Detail...
- Copy exception detail to the clipboard

**View Detail**

Exception snapshot:

System.Security.SecurityException	{"Request for the permission of type 'System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0, PublicKeyToken=b77a5c561934e089' failed."}
Action	System.Security.Permissions.SecurityAction.Demand
Data	{System.Collections.ListDictionaryInternal}
Demanded	{<IPermission class="System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0, PublicKeyToken=b77a5c561934e089" name="System.Security.Permissions.FileIOPermission" />}
DenySetInstance	null
FailedAssemblyInfo	{Exception, Version=1.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089}
FirstPermissionThatFailed	{<IPermission class="System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0, PublicKeyToken=b77a5c561934e089" name="System.Security.Permissions.FileIOPermission" />}
GrantedSet	"<PermissionSet class="System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0, PublicKeyToken=b77a5c561934e089" name="System.Security.Permissions.FileIOPermission" />"
HelpLink	null
InnerException	null
Message	"Request for the permission of type 'System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0, PublicKeyToken=b77a5c561934e089' failed."

OK

App1\* Form1.cs [Design]

Application

Build

Build Events

Debug

Settings

Resources

Reference Paths

Signing

**Security\***

Publish

Configuration: N/A Platform: N/A

Specify the code access security permissions that your ClickOnce application requires in order to run.  
[Learn more about code access security...](#)

Enable ClickOnce Security Settings

This is a full trust application

This is a partial trust application

ClickOnce Security Permissions

Zone your application will be installed from:

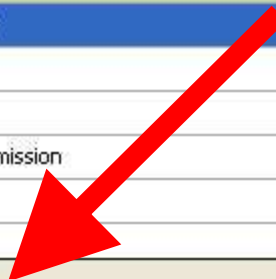
Internet

Permissions required by the application:

Permission	Setting	Included
EnvironmentPermission	(Zone Default)	
FileDialogPermission	(Zone Default)	<input checked="" type="checkbox"/>
FileIOPermission	(Zone Default)	
IsolatedStorageFilePermission	(Zone Default)	<input checked="" type="checkbox"/>
ReflectionPermission	(Zone Default)	
RegistryPermission	(Zone Default)	

Calculate Permissions Properties... Reset

Advanced...



**Lesson #10: You Can't  
Please All of the People  
All of the Time**

# #10: You Can't Please All of the People All of the Time

- ❖ **Whatever security features you ship, some of your users will complain about what you didn't ship**
  - **This is especially true if you're layering a security system on top of another existing system, and don't provide complete wrappers for & access to the underlying system.**
  - **Ex: we chose to implement new security features, like XMLDSIG, instead of adding wrapper classes for NT ACLs & X.509v3 certificates.**

# Some security features we wish we had in V1 (and shipped in V2)

- ❖ **X.509/PKI**
  - Improved X.509v3 handling
  - Certificate stores
  - PKCS#7 & CMS
- ❖ **XMLENC**
- ❖ **NTFS ACLs**



# Looking Forward

# Open Problems

- ❖ **Programming for multi- and many-core systems**
  - Concurrency
  - Async design patterns
- ❖ **Security for distributed systems**
  - Ex: Web Services apps, MANETs
  - Importance of fine-grained delegation
- ❖ **Changing notions of identity**
  - Code identity becoming ever more important
  - “Code” == Apps, components, instruction sequences, etc.
- ❖ **Monetization of bot-nets**
  - “Internet fraud pays”



**Questions?**

The background is a deep blue gradient. In the center, the Microsoft logo is displayed in a white, bold, italicized sans-serif font with a slight drop shadow. The background features faint, semi-transparent elements: a globe in the upper left, binary code (0s and 1s) scattered throughout, and a large, faint 'H' shape on the left side.

**Microsoft®**