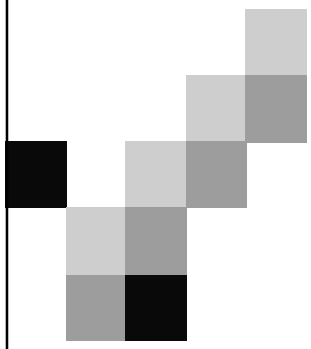


# Security and the .NET Framework

## Code Access Security

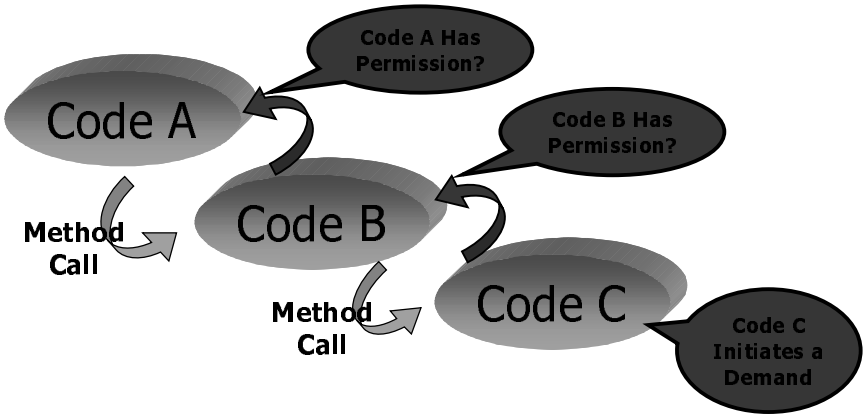
- Enforces security policy on code
  - Regardless of user running the code
  - Regardless of whether the code is in the same application with other code
    - Other code can be more, less, or equally privileged
  - When code attempts a restricted action the system throws a SecurityException
- Code Access Security is the cornerstone of security on the Framework
- Much of the Framework infrastructure is necessary for CAS to work
  - Managed heap, JIT compilation, Assemblies, etc.

# The Idea Behind CAS

- Assembly == Code in Code Access Security
  - Unit of versioning, deployment and execution
  - Assembly is also a unit of security
  - All code in a single assembly share the same permissions
- Applications are always comprised of code from multiple assemblies
  - The .exe assembly
  - Assemblies in the Framework Class Library
  - Custom libraries, mobile code, etc.
- When a thread crosses an assembly boundary, it also crosses a security boundary
- Before a sensitive action is performed, the CLR walks up the call-stack
  - Assures each assembly in the stack-walk has necessary permissions
  - This stack-walk is called a *Demand*

# Demand

- Demand must be satisfied by all callers
  - Ensures all code in causal chain is authorized
  - Code cannot exploit other code with more privilege



# CAS in Action: A First Look

```
using System;
using System.IO;
using System.Security;

class App{
   public static void Main(string[] args){
      StreamReader reader = new StreamReader(args[0]);
      Console.WriteLine(reader.ReadToEnd());
   }
}
```

- Creates StreamReader object
  - StreamReader reads file internally access file
  - Potentially protected resources

# Rational for CAS

- No longer is all code running in a single user session awarded the same rights
  - Example: User launches a word-processor and it has access to the file system
    - The word-processor loads and runs a script downloaded from a network/Internet -- the script's file system access is limited
  - In this example all code is running natively in the same system process
- Increase granularity of security
  - User-logon no longer the smallest unit of security
  - User does not want to switch logon sessions simply to run partially trusted code

# Important Scenarios

- Mobile Code
  - Browser-hosted forms, network installs, distributed applications
  - Network scripts run locally
  - Email embedded macros and scripts
  - Code downloaded and executed locally
- ISP Scenario
  - ISP sells web-hosting to many parties
  - Web code executes natively on ISP machines
  - Code does not require security review

# Scenario #1: Mobile Code

- Advantages of mobile code
  - Executes locally for performance and rich features
  - Not restricted to the limitations of markup or scripts
    - Rich features like animations and drag-and-drop
- Why Code Access Security is necessary
  - Without managed code and CAS mobile code must be scripted or fully trusted
    - Scripted code is slow, limited features
    - Fully trusted code (ActiveX)
      - Bothers users with dialog boxes requesting trust
      - Once established, full trust can be exploited by rogue web-sites
  - CAS enables partial trust of mobile code
    - No dialogs, less exploitable
    - Rich access to GUI API, high performance
    - Best of both worlds

# Scenario #2: ISP Scenario

- Advantages of active server code (CGI, ISAPI, ASP.NET)
  - High performance (improved features/speed over scripted solutions)
  - Dynamic generation of HTML (not restricted to static content)
- With unmanaged code active servers are fully trusted by host (ISP)
  - CGI .exe's or ISAPI DLL's have full access to the system or process
  - One site can undermine the functions of another site
    - Maliciously, or through code error
    - Potentially the whole server can be undermined
  - Security management at the process level is problematic
    - Difficult to administer
    - Doesn't perform well with a minimum of one process per site
  - Result: ISP's disallow active server code
- CAS enables partial trust
  - ASP.NET page can run in proc with other sites
  - Page object for one sight cannot gain access to objects or resources of other sites
  - System resources are not generally available
    - ASP.NET applications can be given access to subsets of system resources such as a directory or registry tree

# Understanding Security Zones

- The system establishes a zone for code (assembly)
  - Happens before code is executed
  - Zones are based on the source location of code
  - Zones are a subset of an advanced CAS feature called *evidence*

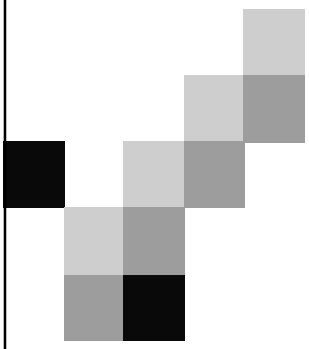
| Zone | Description |
|---|---|
| Local | Code executed from the local system. Code in this zone has full trust. |
| Intranet | Code executed from a share or URL on the enterprise network. Limited access to local resources. |
| Internet | Code downloaded from the Internet. Minimal access to local resources. |
| Restricted | Code in the restricted zone is not allowed to execute. |

# Practical Zones Programming

- Common .NET developer experience
  - Create an application, test it, fix bugs, code works fine
  - Developer then gives the app to someone who runs it from a network share
    - Application begins crashing inexplicably
    - Reason: The Intranet zone has fewer permissions than the Local zone.
  - Solution: Test your software in different zones
- Running managed software in different zones
  - If your software is an .exe then it is sure to be run in at least two zones
    - Local and Intranet
  - Your software should at least recover gracefully if a security exception is thrown
  - If your software is a reusable control, then it could feasibly be run from any of the zones

# Testing Zones

- You should test your software from the relevant zones
  - Run your software locally
  - Run it from a share
  - Run it from a URL on the internet
- Your software will almost certainly throw some exceptions when first tested in a more restricted zone
  - Handle the exceptions and gracefully shut down
  - Handle the exceptions and work around with restricted features
  - Don’t just let security exceptions crash your software!



# Demo Pad.exe

zoner Pad.exe  zoner /z:MyComputer Pad.exe  caspol -rsp Pad.exe

# Permissions

- Permissions are objects that the CLR references when performing a demand
- Permissions are granted to your assembly based on its zone (in addition to other assembly evidence)
- Permission objects themselves play an integral role in the demand process
  - The Demand() method calls virtual functions on the permission object when checking for a match
  - This involvement at the permission level makes the kinds of available permissions very flexible
- It is possible to design custom permissions for your code libraries
  - More on this in the advanced CAS session

## Some Frameworks Permissions

- FileIOPermission
- FileDialogPermission
- IsolatedStoragePermission
- UIPermission
- PrintingPermission
- WebPermission
- SocketPermission

- These are Just examples, the FCL defines many permissions

## Your Assembly is Loaded

- The system gathers *evidence* for your assembly
  - Digital signatures, Realm information
  - Zone information
- From evidence, your assembly is assigned one or more *code groups*
- Code groups define the permission sets to apply to your assembly
  - Permission sets are collections of permissions
- Once loaded, the system has a permission grant associated with your assembly

## Your Assembly's Code Executes

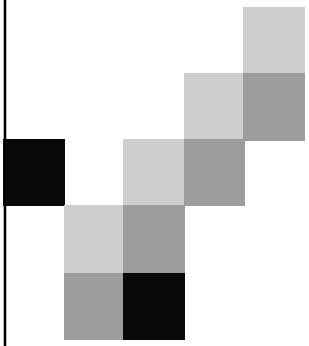
- Your code executes, and uses reusable objects
  - FCL, custom objects, etc
- Eventually, a method or constructor of an object will demand a security permission
  - Each assembly in call stack is checked for permission
  - If the demand reaches your assembly, your assembly's grant is checked for permission
    - If you have it, the demand continues up the stack
    - If you do not have the permission in your grant, a SecurityException is thrown
  - If the demand reaches the top of the stack, the demand has succeeded
    - The restricted action is performed

## CAS Applies to All Assemblies

- All assemblies get a grant upon loading
- All assemblies' grants are checked upon demand
- CAS is always aware of who initiates an action

# Rational for CAS: Summary

- Managed code makes CAS possible
  - Unmanaged code, impossible to implement CAS
- CAS enables local execution of code
  - Safe, even if code is not trusted
  - Opens the door to rich features
  - Removes the need for rigid code review
    - Third party code
    - Your software must still be reviewed for security
- CAS permissions based on
  - Code authentication
  - Call stack



# Security and the .NET Framework

The slides following this one contain the figure graphics for the tutorial that goes with this presentation.