

**Module 10:
Securing Windows
Forms Applications**

Overview

- Security in the .NET Framework
- Using Code Access Security
- Using Role-Based Security

Lesson: Security in the .NET Framework

- Security Basics
- What is Evidence?
- What are Permissions?
- How Does Code Access Security Work?
- Animation: Code Access Security
- What is Role-Based Security?
- What is Authentication?
- What is Authorization?

Security Basics

■ Code Access Security

Permissions granted to code based on:

- Evidence
- Permissions and Permission Sets
- Security Policy

■ Role-Based Security

Permissions granted to users based on:

- User name
- Roles

Windows group(s)

Generic and custom principals and identities

Microsoft .NET Passport

What is Evidence?

- A set of information about the identity and origin of an assembly
- Used by the .NET Framework security system at load time to determine the permissions that an assembly receives based on existing security policy
- Examples of Items that Make Up Evidence
 - Strong name signature, code publisher, location, and zone
 - Other custom-defined items
- Strength of Evidence

What are Permissions?

Code access permissions are the rights to access certain computing resources

Examples of built-in permission classes

UIPermission, PrintingPermission, WebPermission,
IsolatedStorageFilePermission

How Does Code Access Security Work?

Code Groups

Gather evidence for assembly

Assign assembly to code group(s)

Assembly gets *UNION* of permission sets for its code groups

Repeat this same security check for all security policy levels

Policy Levels

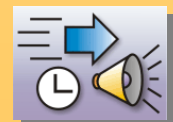
- Enterprise
- Machine
- User
- Application domain (optional)

Per assembly grant
Assembly gets *INTERSECTION* of permission sets for all policy levels

Each assembly in an application can have a different permission grant

Animation: Code Access Security

In this animation, you will see how the permission grant for a Microsoft .NET assembly is determined based on evidence and policy levels



What is Role-Based Security?

- Identity

- Typically consists of user's log on name

- Principal

- Typically consists of role(s) associated with a user

- Roles can be

 - Microsoft Windows user and group

 - Or -

 - Custom (using generic principals and identities)

- **Authenticated identity generally == identity + principal**

What is Authentication?

- Process of finding and verifying the identity of a user
- Done against some authentication authority

Examples of authentication mechanisms:

Operating system (NTLM or Kerberos v. 5)

.NET Passport

Application-defined mechanisms

What is Authorization?

- Process of determining whether a user's request to do something is allowed to proceed
- Happens after authentication and is based on the user's authenticated identity

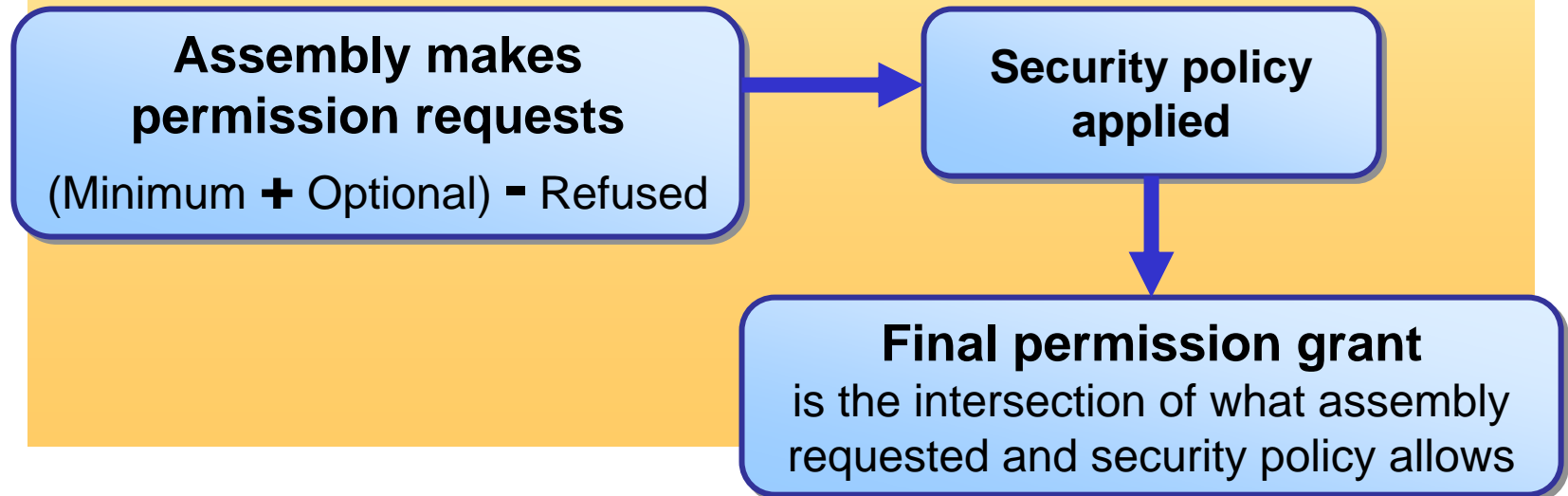
Lesson: Using Code Access Security

- How to Use Code Access Security
- How to Make Assembly Permission Requests
- Demonstration: Administering Security Policy Settings
- How to Test the Code Access Security of an Application
- Practice: Adding Permission Requests

How to Use Code Access Security

Use attributes to make assembly permission requests

- Minimum permission sets
 - If not available, assembly will not load and **PolicyException** is thrown
- Optional permission sets
 - Useful, but code is still able to run effectively without them
- Refused permissions
 - Allows your code to ensure protection of certain resources



How to Make Assembly Permission Requests

1

Add attributes to the assembly information file

Be sure to include assembly scope on permission request attributes

```
' Add attributes to the AssemblyInfo source file
' Request for a specific permission
<Assembly:UIPermission( _
    SecurityAction.RequestMinimum, _
    Window:=UIPermissionWindow.SafeTopLevelWindows)>

' Request for a permission set
<Assembly:PermissionSet( _
    SecurityAction.RequestMinimum, _
    Name:="LocalIntranet")>
```

Assembly scope

2

Add code to handle security exceptions

[CodeExample](#)

Demonstration: Administering Security Policy Settings



In this demonstration, you will see how to use the .NET Framework Configuration tool (Mscorcfg.msc) to administer security policy settings

How to Test the Code Access Security of an Application

- Use the Code Access Security tool (Caspol.exe) to:

- View policy information

```
Caspol -l
```

List code groups and permission sets

```
Caspol -lg
```

List code groups

```
Caspol -lp
```

List permission sets

- Add code groups to test applications with different permission sets

```
Caspol -ag 1 -url file:///C:/test/* Internet  
-n Test_Group -exclusive on
```

Add code group

```
Caspol -cg Test_Group LocalIntranet
```

Change
permission set

```
Caspol -rg Test_Group
```

Remove code group

- Do *not* turn off security

Practice: Adding Permission Requests

In this practice, you will

- View the behavior of an application that contains no permission requests
- Add permission requests to an assembly and then test the behavior of the application in different security contexts

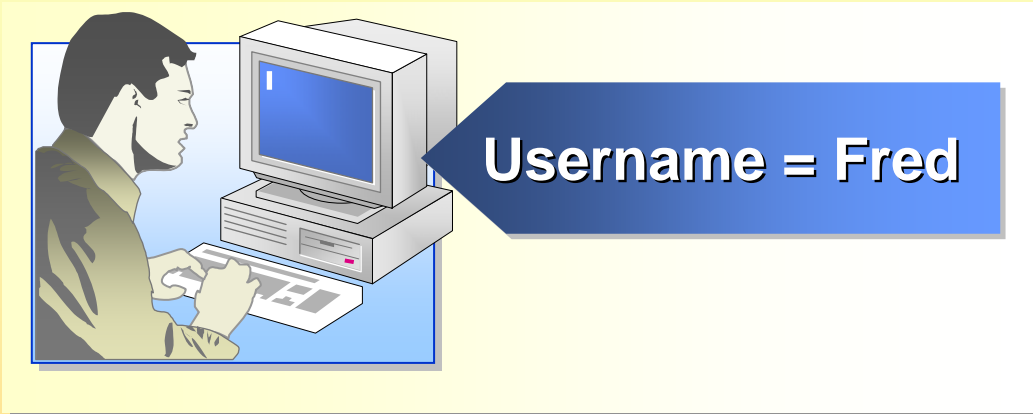
Begin reviewing the objectives for
this practice activity

15 min



Lesson: Using Role-Based Security

- Check identity of the user



- Check the role of the user



How Role-Based Security Works

- Authentication and authorization

- Identities

- Windows identity
- Generic identity
- Custom identity

- Principals

- Windows principal
- Generic principal
- Custom principal

How to Create WindowsPrincipal and WindowsIdentity Objects

- Choice of two techniques

- Creating objects for a single validation

```
Dim MyIdentity As WindowsIdentity = _  
    WindowsIdentity.GetCurrent()  
Dim MyPrincipal As WindowsPrincipal = _  
    new WindowsPrincipal(MyIdentity)
```

- Creating objects for repeated validation

```
AppDomain.CurrentDomain.SetPrincipalPolicy( _  
    PrincipalPolicy.WindowsPrincipal)  
Dim MyPrincipal As WindowsPrincipal = _  
    System.Threading.Thread.CurrentPrincipal  
    As WindowsPrincipal
```

[CodeExample](#)

How to Create GenericPrincipal and GenericIdentity Objects

- Create and initialize a GenericIdentity object

```
Dim MyIdentity As New GenericIdentity("User1")
```

- Create and initialize a GenericPrincipal object and attach it to the current thread

```
Dim MyStringArray(2) As String  
MyStringArray(0) = "Manager"  
MyStringArray(1) = "Employee"  
Dim MyPrincipal As New GenericPrincipal(MyIdentity, _  
    MyStringArray)  
System.Threading.Thread.CurrentPrincipal = MyPrincipal
```

How to Use Principals and Identities to Control Access to an Application

After you have created a principal object

- Use the **Name** property of the principal's identity object to check the user's name

```
' Assume a valid principal is in MyPrincipal
If (String.Compare(MyPrincipal.Identity.Name, _
    "DOMAIN\Fred", True)=0) Then
' Permit access to some code
```

- Use the principal's **IsInRole** method to check role membership

```
' Assume a valid principal is in MyPrincipal
If (MyPrincipal.IsInRole( _
    "DOMAIN\Administrators")) Then
' Permit access to some code
```

Demonstration: Using Role-based Security to Control Access to an Application



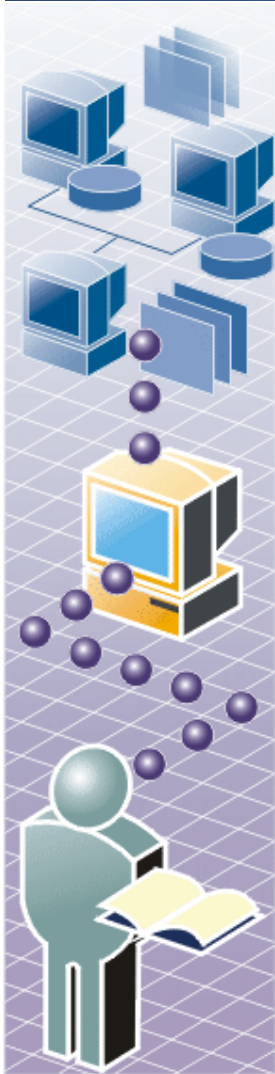
In this demonstration, you will see how an application uses role-based security

Review

- Security in the .NET Framework
- Using Code Access Security
- Using Role-Based Security

Lab 10.1: Adding and Testing Permission Requests

Exercise 1: Adding and Testing Permission Requests



Course Evaluation

