UNIVERSITÉ ANTONINE
Faculté d'ingénieurs en Informatique, Multimédia, Réseaux et Télécommunications

Document 2 – Examples and Implementations

Course: Applications avancées avec C#

Presented by: Elie Matta et al.

Copyright © 2010-2011, eliematta.com. All rights reserved

Prepared by Elie Matta et al.

Table of Contents

Introduction 3
Example 1 – The DNS 4
A.Introduction 4
B..NET Framework Configuration 4
a.Creating a new permission set 4
b.Creating a new code group 6
c.Exclusive and LevelFinal 8
C.Code 9
D.Implementation 9
Example 2 – Registry 11
A.Introduction 11
B.Codes 11
a.classmain 12
b.Button #1 12
c.Button #2 12
d.Button #3 12
C.Implementation 13
i.Scenario 1 13
ii.Scenario 2 13
iii.Scenario 3 14
Example 3 – Environnement variables 16
A.Introduction 16
B.Code 16
C.Configuration 17
D.Implementation 19
Example 4 – User Interface 23
B.Code 24
C.Implementation 25
Example 5 – Web access 27
A.Introduction 27

Copyright © 2010-2011, eliematta.com. All rights reserved |

B..NET Framework Configuration 27
Modifying permission set 27
C.Code 30
D.Implementation 31
Example 6 – Printing 32
A.Introduction 32
B..NET Framework Configuration 32
Modifying permission set 32
C.Code 35
D.Implementation 35
Example 7 – The active directory 37
A.Introduction 37
B..NET Framework Configuration 37
a.Creating a new permission set 37
b.Creating a code group 40
C.Code 43
D.Implementation 43
Note: 45

Copyright © 2010-2011, eliematta.com. All rights reserved |

Question 5

Introduction

After reading the security .NET framework security, it is time now to explore all the theory in detailed examples separated as follows:

1) The DNS
2) Registry
3) The environment variables
4) User interface
5) Web access
6) Printing
7) The directory service

Two tools shipped with the .NET Framework SDK allow us to configure security: CASpol.exe(Code Access Security Policy Tool) and MSCorCfg.msc (.NET Configuration Tool). The first is a command-line utility, the second a Microsoft Management Console (MMC) snap-in. The graphical user interface of the MSCorCfg MMC snap-in is easier to use and allows you to visualize the overall security configuration more readily. CASpol is quicker and can be used in scripts or batch files. We will use one of the two tools each time we want to configure our implementation. The namespace System.Security.Permissions will be included is in most of our examples.
Please note that we separated this implementation to seven examples to give each example a clearer step by step implementation at each and every example; We could have managed to enclose all of them in one single application with seven buttons testing the seven different permissions.

Copyright © 2010-2011, eliematta.com. All rights reserved |

Example 1 – The DNS

Source code path: Examples/Example1/dnstest

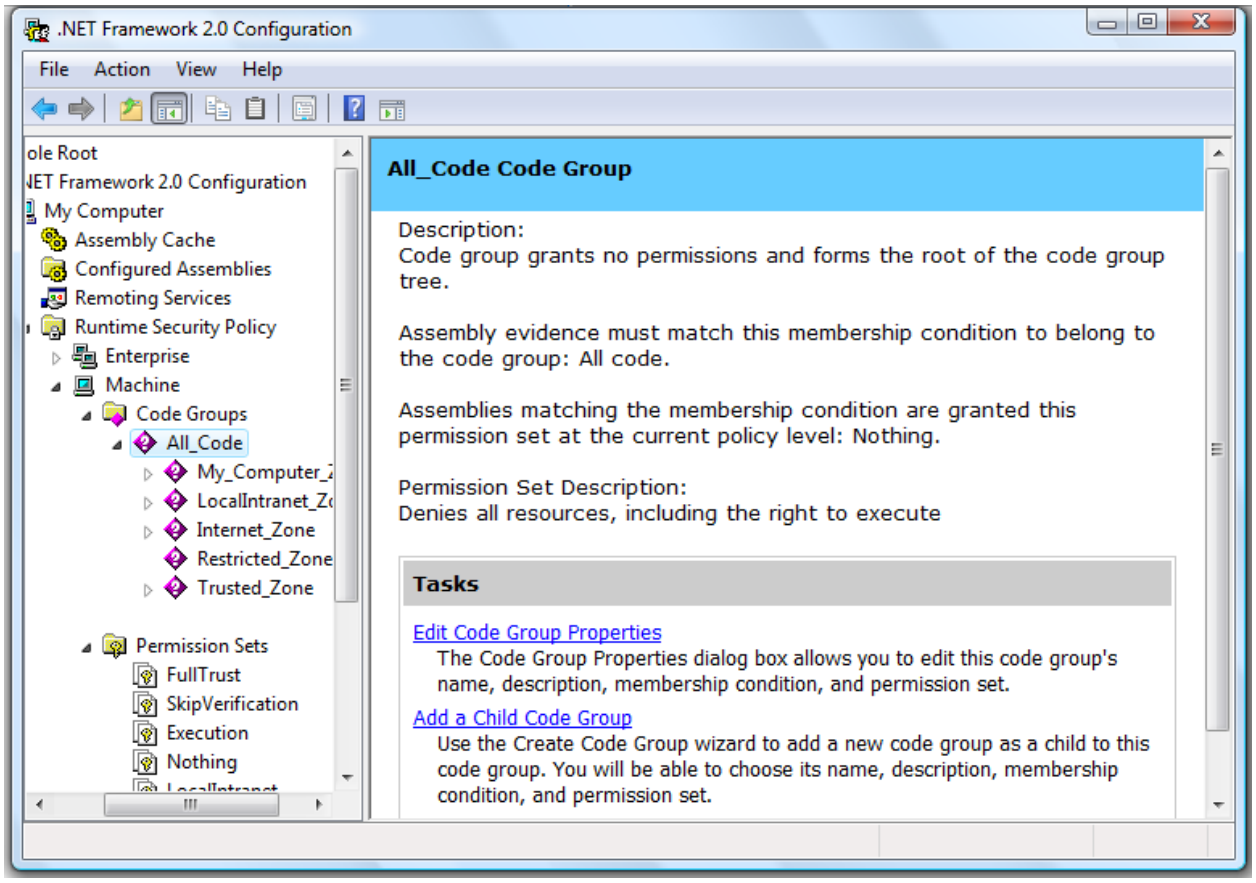
Using strong name – Declarative security

A. Introduction

Our example consist of demanding simple permission to resolve www.google.com website, we will assign our program to a new permission of our creation and a code group.
We will use a Console application which we will add on it two namespaces: System.Net and System.Security.

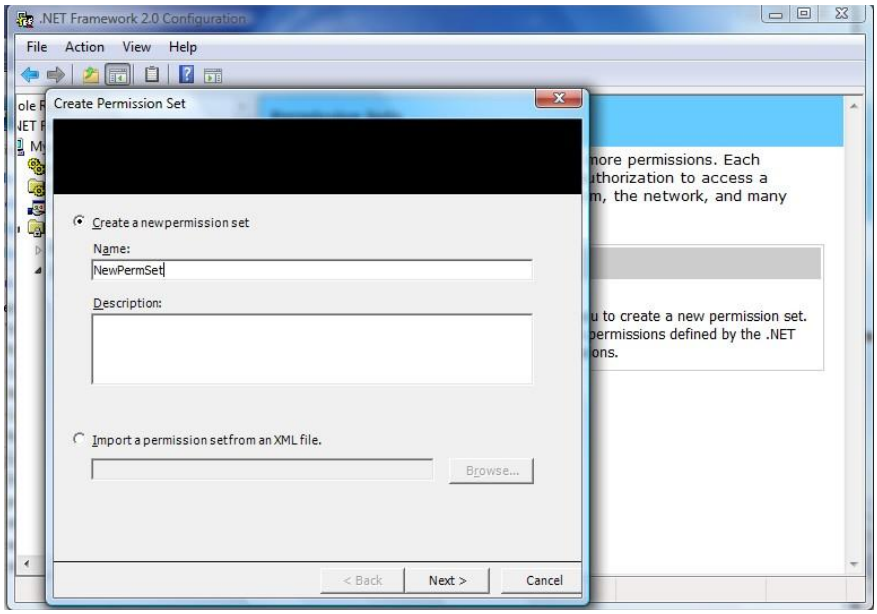
B. .NET Framework Configuration

In this example we will use the .NET Framework configuration tool to configure a new permission and a new code group.

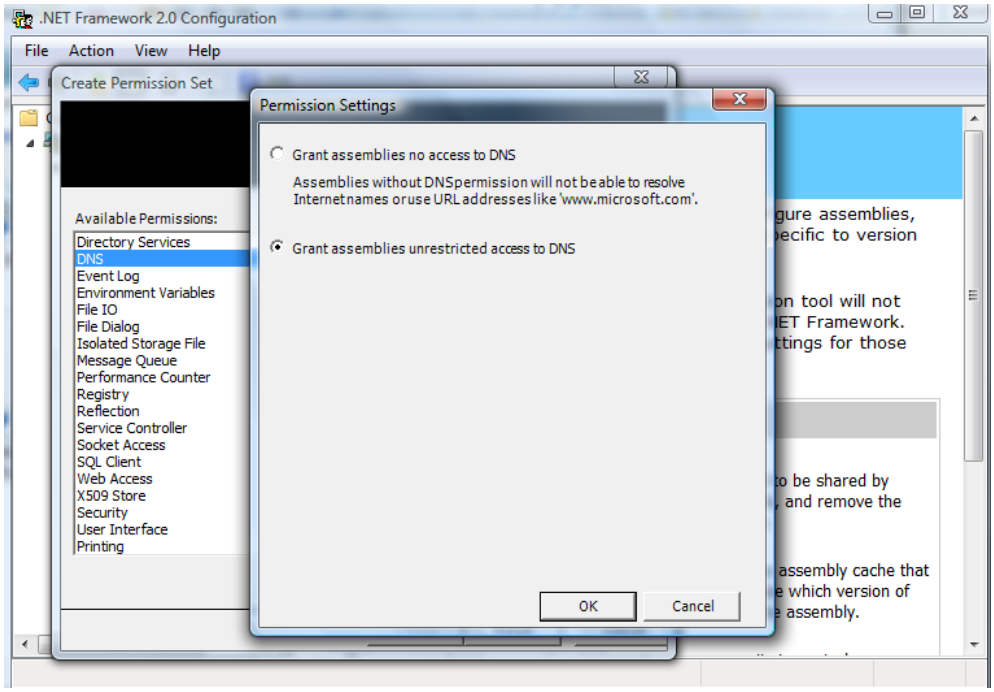


a. Creating a new permission set

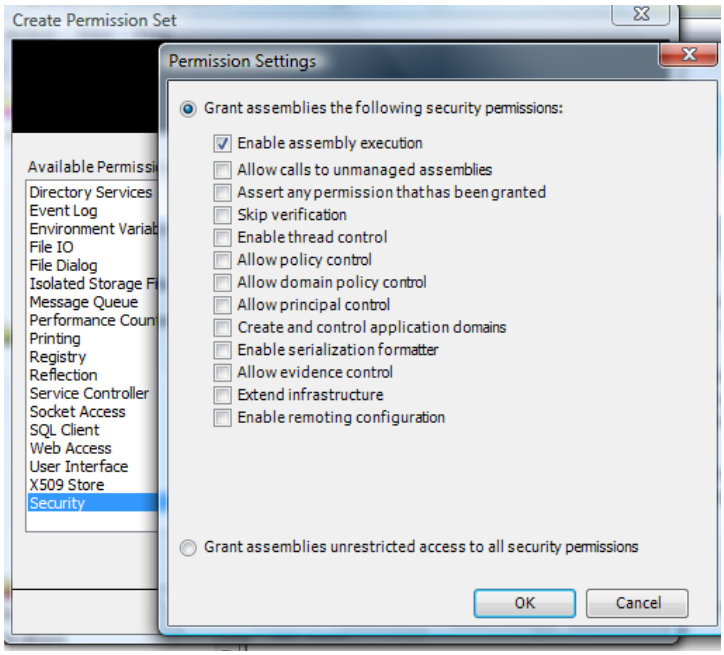
Expand the **Runtime Security Policy** node. You can see the security policy levels - Enterprise, Machine and User. We are going to change the security settings in Machine policy. First we are going to create our own custom permission set. Right click the **Permission Sets** node and choose **New**. We will name it *NewPermSet*.



In the next figure, we can add permissions to our permission set. In the left panel, we can see all the permissions supported by the .NET Framework. Now get the properties of **DNS** permission. Set "**Grant assemblies unrestricted access to DNS**"

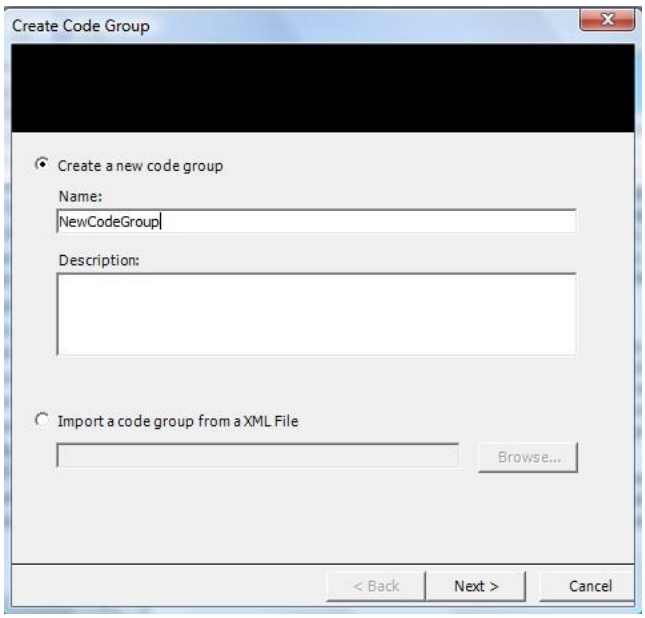


We should add also the **Security permission** to add the permission to execute the code, by selecting security and then checking **Enable assembly execution**.



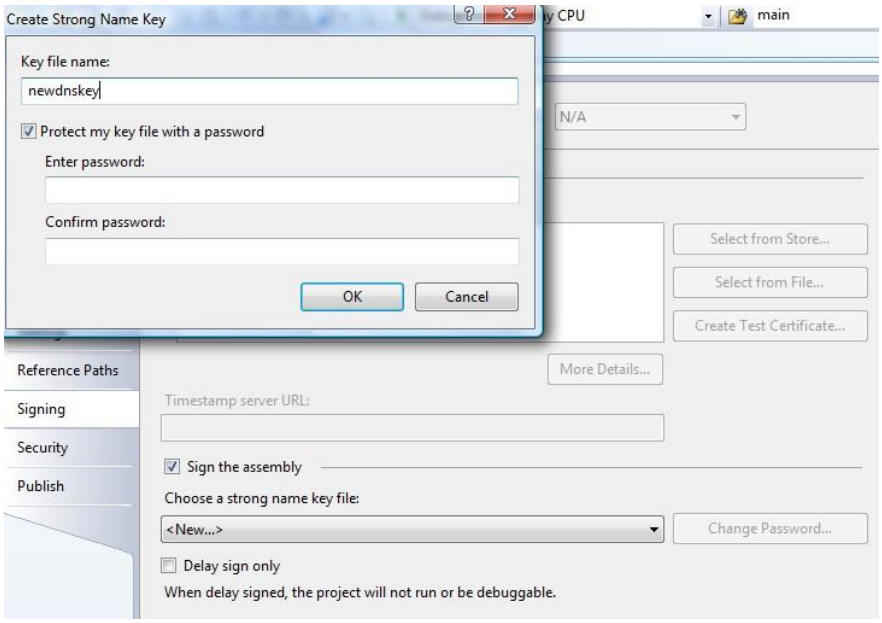
b. ***Creating a new code group***

Now we will create a code group and set some conditions, so our assembly will be a member of that code group. Notice that in the code groups node, **All_Code** is the parent node. Right Click the **All_Code** node and choose **New**. You'll be presented with the **Create Code Group wizard**. We are going to name it *NewCodeGroup*.

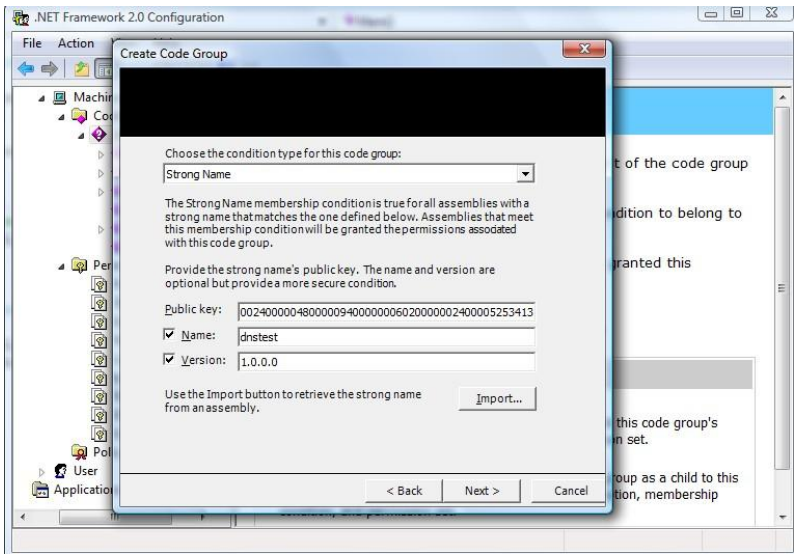


Copyright © 2010-2011, eliematta.com. All rights reserved |

In the next figure, you have to provide a condition type for the code group. Now these are the **evidence** that we mentioned in Document 1 – page 4. For this example, we are going to use the **Strong Name** condition type. First, sign your assembly (by using sn.exe per example or from the Visual studio by right-clicking on the project name, properties, signing like in the figure below) with a strong name and build the project.

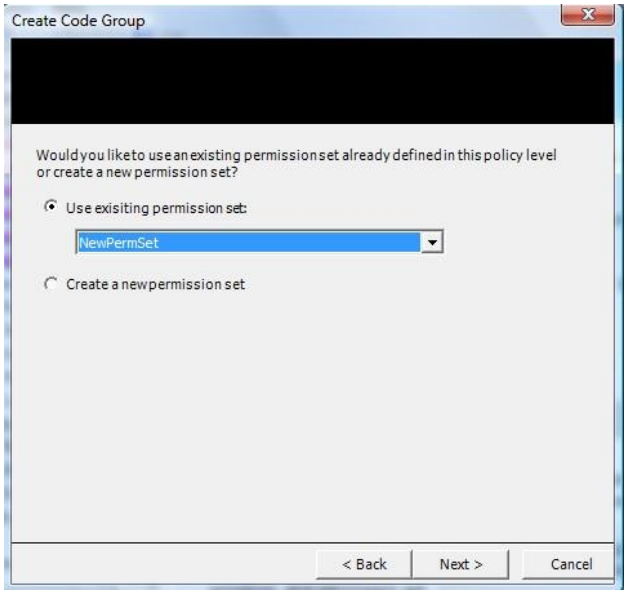


Back to the .NET configuration tool, now press the **Import** button then we select our assembly. Public Key, Name and Version will be extracted from the assembly, so we don't have to worry about them.



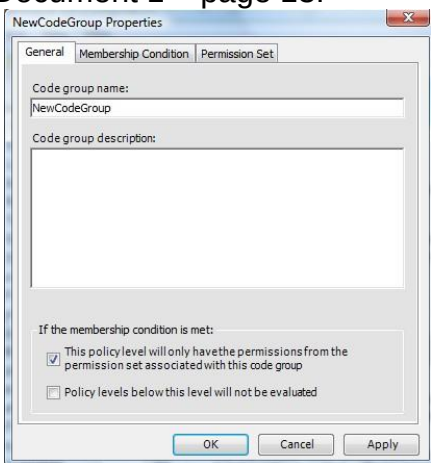
Now move on to the next figure. We have to specify a permission set for our code group. Since we have already created one – *NewPermSet*, select it from the list box.

Copyright © 2010-2011, eliematta.com. All rights reserved |



c. ***Exclusive and LevelFinal***

If no one haven't messed around with the default .NET configuration security settings, our assembly should already belong to another built-in code group - My_Computer_Zone. When permissions are calculated, if a particular assembly falls into more than one code group within the same policy level, the final permissions for that assembly will be the union of all the permissions in those code groups. To calculate permissions you should reference to the Document 1 – page 1, now we need to run our assembly only with our permission set and that is *NewPermSet* associated with the *NewCodeGroup*. So we have to set another property to do just that. Right click the newly created **MyCodeGroup** node and select **Properties**. Check the check box saying "**This policy level will only have the permissions from the permission set associated with this code group.**" This is called the Exclusive attribute. If this is checked then the run time will never grant more permissions than the permissions associated with this code group. The other option is called LevelFinal. These two properties come into action when calculating permissions and they are explained in Document 1 – page 13.



Copyright © 2010-2011, eliematta.com. All rights reserved |

C. Code

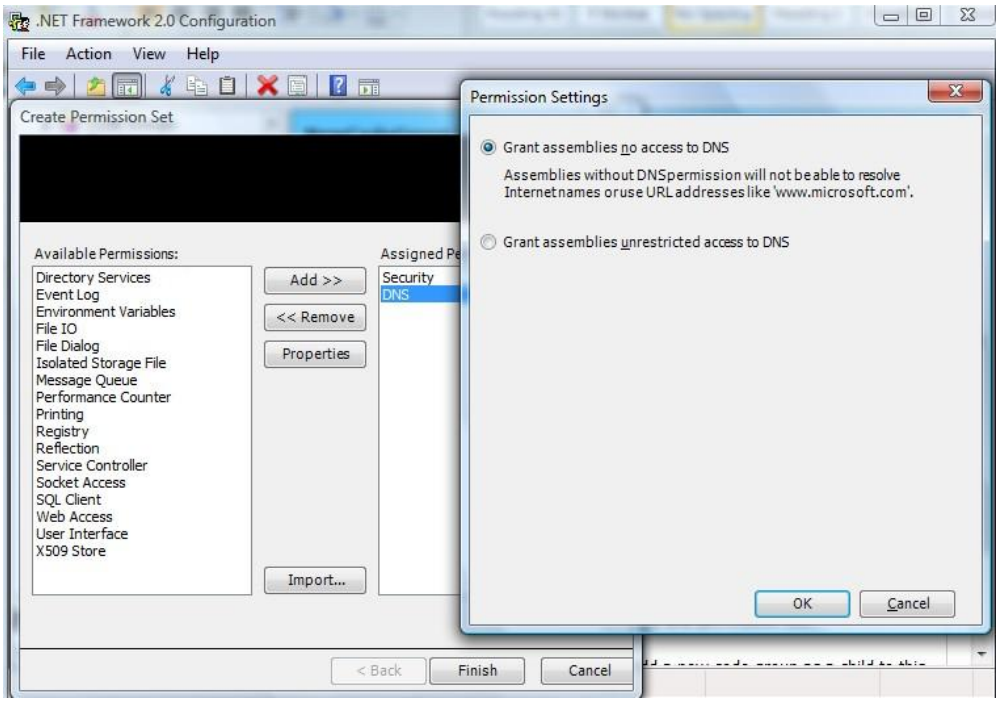
```
//Uses the DnsPermissionAttribute to restrict access only to those who have
permission.
        [DnsPermission(SecurityAction.Demand, Unrestricted = true)]
//Declarative security
        public class MyClass
        {

            public static IPAddress GetIPAddress()
            {
                IPAddress ipAddress =
Dns.Resolve("www.google.com").AddressList[0];
                return ipAddress;
            }
            public static void Main()
            {
                try
                {
                    //Grants Access.
                    Console.WriteLine(" Access granted\n The assigned IP
Address is :" +

MyClass.GetIPAddress().ToString());
                }
                // Denies Access.
                catch (SecurityException securityException)
                {
                    Console.WriteLine("Access denied");
                    Console.WriteLine(securityException.ToString());
                }
```

D. Implementation

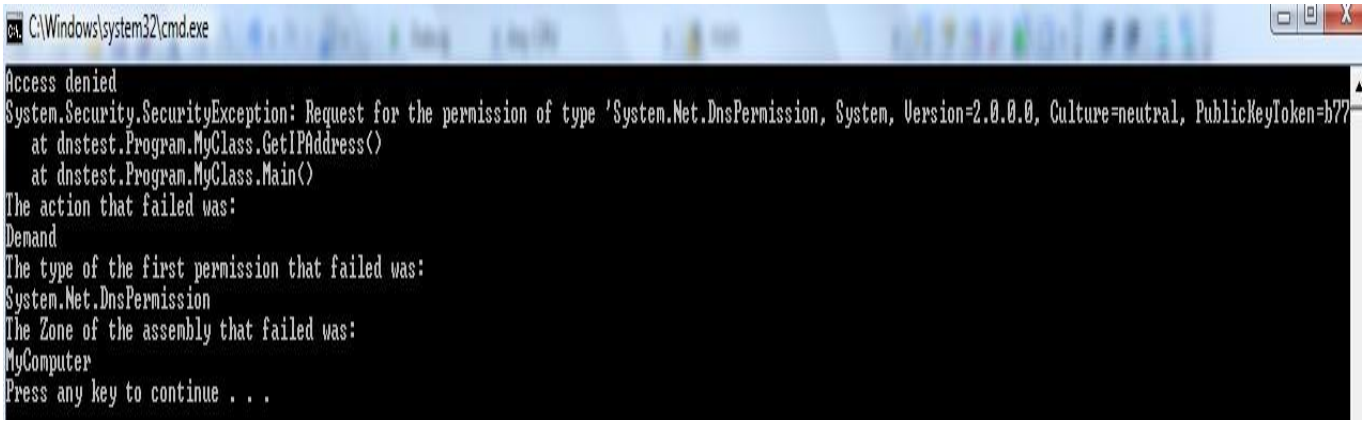
It's time to run the code. What we have done so far is, we have put our code into a code group and given unrestricted access to DNS. Run the code it should work fine, resolving correctly the www.google.com website as shown below:

But if we changed the permission set for our *NewPermSet* setting the DNS permission to **Grant assemblies no access to DNS** in the .NET configuration then we will have an error:

Copyright © 2010-2011, eliematta.com. All rights reserved |



When we run the code it will fail to load the DNS because it has no permission



Example 2 – Registry

Implementation: Examples/Example2/Implementation

Using declarative security – XML configuration file

A. Introduction

A little reminder on the basics of the three levels to get started: “Fully trusted” code can do whatever the user can do (which might be limited if the user is not using a Windows admin account). “No trust” code can’t do anything. Anything in between is called “partial trust.” Obviously, there are many degrees of partial trust – the code might have almost no privileges at all, or it might be able to do everything except reading the registry or something.
We also remind you that that there are three main ways of causing code to run with partial trust:

1. If the assembly is loaded from a network share then it will run with the “LocalIntranet” permission set (by default this enforces a number of restrictions such as no registry, no file IO, limited reflection and so on, we will see this in our example).
2. If your assembly refuses one or more permissions (per example you declare that you don’t want to be able to perform reflection) then your assembly is by definition partially trusted.
3. You can configure your ASP.NET application to run with a particular trust level. This is defined in config files; “medium trust” means no registry, no reflection, no file IO outside your app’s virtual directory.

Notice that an assembly might be fully trusted or partially trusted depending on the runtime circumstances. The assembly might be fully trusted if loaded from the C: drive, but will be partially trusted if loaded from a network share.

Next, a couple of handy definitions:
Caller – Code that calls other code
Callee – Code that is called by other code
So if method ClientMethod() calls method ServerMethod() then ClientMethod is the caller and ServerMethod is the callee.
If the calling method (the caller) is in an assembly that is running with partial trust then it is a “partially trusted caller.” In this case, if the callee’s assembly does not have the AllowPartiallyTrustedCallers (APTC) attribute then the called code will not run – regardless of the trust level of the callee’s assembly. You will get an exception.

B. Codes

The example has two strongly named assemblies: PartialTrustTest.exe and PartialTrustTestLib.dll. There is also a config file called PartialTrustTest.exe.config that is needed for one of the scenarios.

Copyright © 2010-2011, eliematta.com. All rights reserved |

a. classmain

```
public class classmain
        {
            public static void TestRegistryAccess()
            {
                RegistryKey test = Registry.LocalMachine;
                test.OpenSubKey("Software", true);
            }



            [RegistryPermission(SecurityAction.Assert,
                                Unrestricted = true)] //Declarative security
            public static void AssertRegistryPermissionAndTestAccess()
            {
                RegistryKey test = Registry.LocalMachine;
                test.OpenSubKey("Software", true);
            }
        }
```

b. Button #1

```
RegistryKey localMachine = Registry.LocalMachine;
            try
            {
                localMachine.OpenSubKey("Software", true);
                MessageBox.Show("Registry key was opened", "Partial Trust
Test", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
            }
            catch (SecurityException exception)
            {
                MessageBox.Show("Can't open registry key: " +
exception.Message + "; stack = " + exception.StackTrace, "Partial Trust
Test", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
```

c. Button #2

```
try
            {
                classmain.TestRegistryAccess();
                MessageBox.Show("Registry key was opened", "Partial Trust
Test", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
            }
            catch (SecurityException exception)
            {
                MessageBox.Show("Can't open registry key: " +
exception.Message + "; stack = " + exception.StackTrace, "Partial Trust
Test", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
```

d. Button #3

```
try
```

Copyright © 2010-2011, eliematta.com. All rights reserved |

```
            {
                classmain.AssertRegistryPermissionAndTestAccess();
                MessageBox.Show("Registry key was opened", "Partial Trust
Test", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
            }
            catch (SecurityException exception)
            {
                MessageBox.Show("Can't open registry key: " +
exception.Message + "; stack = " + exception.StackTrace, "Partial Trust
Test", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
```

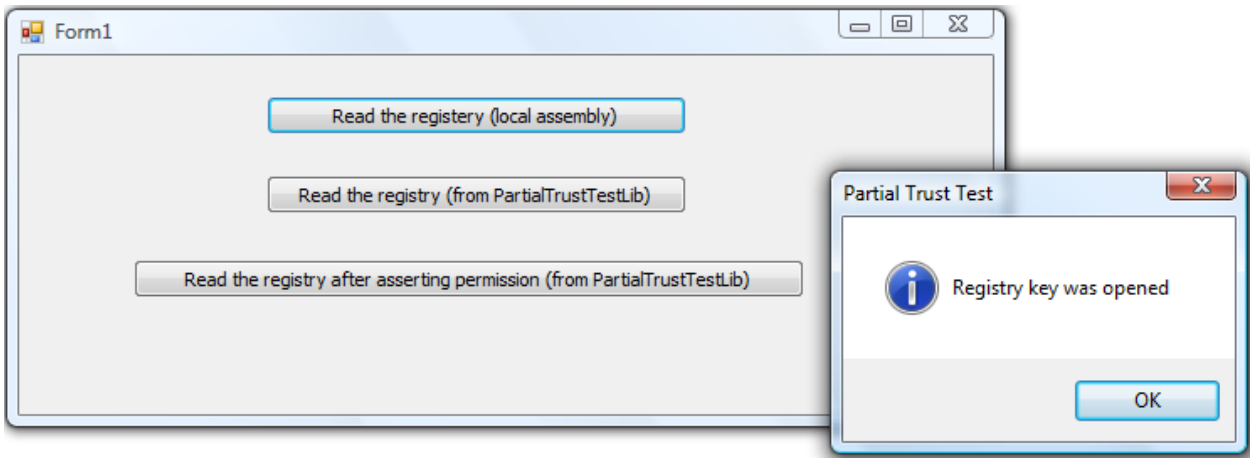
C. Implementation

i. Scenario 1

Start by placing all the files together in a directory on your C: drive and running the exe. You will see a form with three buttons. They all do the same thing: they try to read the registry. The only difference is in the methods that they use to call the RegistryKey.OpenSubKey method:

1. Read the registry (local assembly) – calls a method in the EXE
2. Read the registry (from PartialTrustTestLib) – calls a method in the DLL (see code at bottom of email)
3. Read the registry after asserting permission (from PartialTrustTestLib) – calls a method in the DLL that asserts the right to read the registry (I'll explain this below, but note that it has nothing to do with the Asserts used in testing)

Click each of the buttons in turn; you should find that they all work fine. This is because both assemblies are fully trusted.



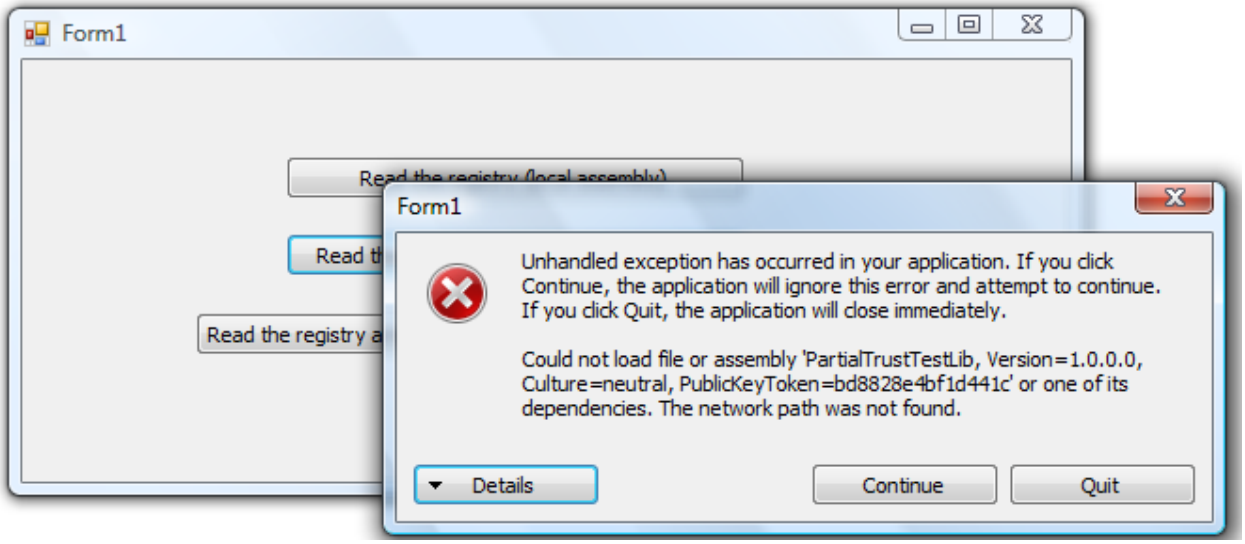
ii. Scenario 2

Move the DLL to a network drive – let's call that the U: drive and ensure you delete it from the C: drive so that we can be sure which one is being loaded.

Copyright © 2010-2011, eliematta.com. All rights reserved |

Open PartialTrustTest.exe.config in Notepad and uncomment the line.
Replace my username (Vista) with yours in the href attribute.
Now run the EXE.

Button 1 is still OK, but the other two buttons throw a SecurityException. This is because the DLL is now running with the LocalIntranet permission set and as such it can't read the registry.
The APTC attribute is not relevant here because the caller is fully trusted (it is the callee that is partially trusted).



iii. Scenario 3

This is the most interesting one. To set up this scenario, follow these steps:

1. Delete PartialTrustTest.exe.config
2. Move the EXE to your network drive (U: drive)
3. Move the DLL back to your C: drive
4. Open the .NET Framework 2.0 Configuration utility and add the DLL to your Global Assembly Cache (GAC) or use the cmd: gacutil.exe /i PartialTrustTestLib.dll
5. Now run the EXE

Note that the DLL has the APTC attribute, which it needs here because the exe is no longer running with full trust.

Button 1 fails, because the EXE is now running with the LocalIntranet permission set.

Button 2 fails. Why? The DLL is now fully trusted (it is in the GAC and it doesn't refuse any permissions). But the EXE does not have RegistryPermission. This is an example of tempting (the unprivileged EXE asks the privileged DLL to ask the .NET Framework).

Copyright © 2010-2011, eliematta.com. All rights reserved |

To prevent this, the RegistryKey.OpenSubKey method demands that all its callers have RegistryPermission. This demand causes the CLR to do a 'stack walk', checking that each caller in the stack has the required permission.

Button 3 succeeds. This is because it calls a method in the DLL that 'asserts' RegistryPermission.

Notice that the assertion didn't help in scenario 2, because the DLL didn't have RegistryPermission.

To summarise:

Scenario	EXE	DLL	Btn 1	Btn 2	Btn 3
1	C: drive	C: drive	OK	OK	OK
2	C: drive	U: drive	OK	Fails – callee doesn't have RegistryPermission	Fails – callee doesn't have Registry Permission
3	U: drive	GAC	Fails – callee doesn't have RegistryPermission	Fails – stack walk discovers that EXE doesn't have RegistryPermission	OK – DLL asserts RegistryPermission

Using CAS correctly involves ensuring that your code can't be used maliciously. Our DLL allows partially trusted callers and one of its methods asserts RegistryPermission. Once that DLL is installed in the GAC these two settings lower the security bar considerably. Any assembly that can run on our CLR can load our DLL and use our method to read the registry. Before adding the assertion we should (a) check our method carefully to make sure it can't be used maliciously, and/or (b) apply extra restrictions to our method. The easiest way to restrict the method is to add a Demand to the method. This is where we demand that the callers meet a certain requirement (not the same one that we're asserting, obviously – if they have that then there's no point in asserting it)

Copyright © 2010-2011, eliematta.com. All rights reserved |

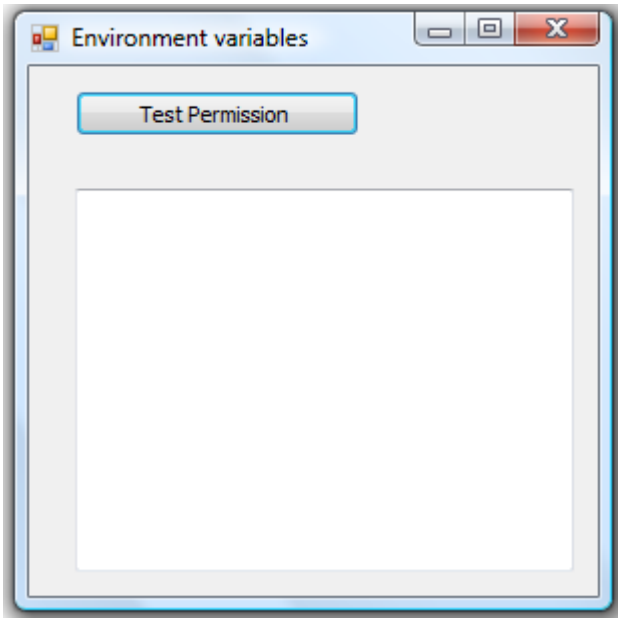
Example 3 – Environnement variables

Source code path: Examples/Example3/envartest

Using XML condition file – Imperative security

A. Introduction

In this example, we will use the CASpol.exe tool to implement our work:
The test assembly we'll use is a simple Windows Forms application with a button and a text box. When the user clicks the button, the application demands two arbitrary security permissions, one after the other: a *File IO* permission to read the C:\Windows directory, and an *Environment* permission to read the USERNAME environment variable. If a demand succeeds, a simple message is added to the text box. If either demand fails—that is, if the assembly isn't granted the requested permission—the corresponding error string is added to the text box instead:



B. Code

And in the button:

```
private void btnTestPerms_Click(object sender, EventArgs e)
        {
            try
            {
                FileIOPermission p = new FileIOPermission(
                    FileIOPermissionAccess.Read, "C:\\WINNT");
                p.Demand();
                textBox1.Text += "FileIOPermission OK\r\n";
            }
            catch (Exception ex)
            {
```

Copyright © 2010-2011, eliematta.com. All rights reserved |

```
            textBox1.Text += ex.Message + "\r\n";
        }
        try
        {
            EnvironmentPermission p = new EnvironmentPermission(
                EnvironmentPermissionAccess.Read, "USERNAME");
            p.Demand();
            textBox1.Text += "EnvironmentPermission OK\r\n";
        }
        catch (Exception ex)
        {
            textBox1.Text += ex.Message + "\r\n";
        }

    }
```

To create a permission set with CASpol, we first need to create an XML file detailing the individual permissions we want (elie.xml):

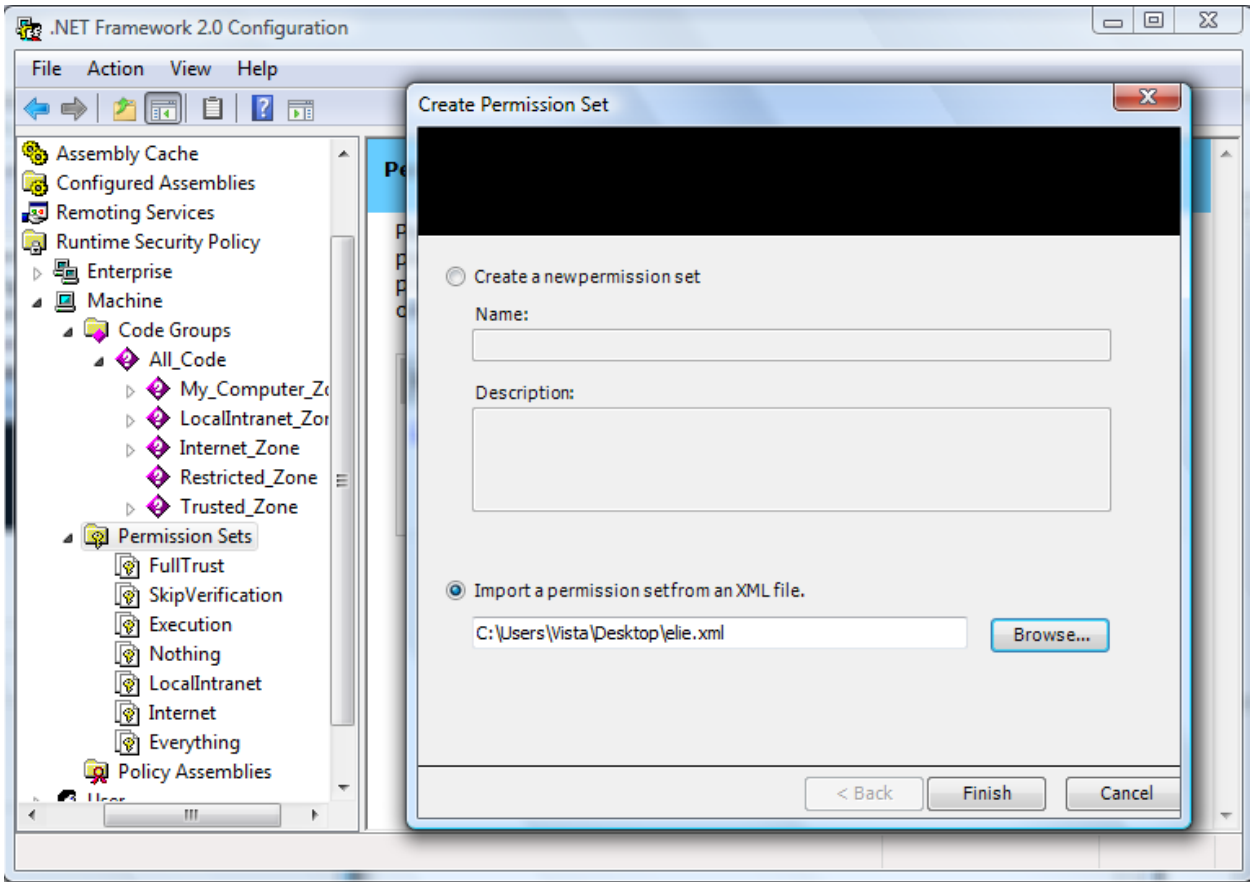
```
<PermissionSet class="NamedPermissionSet"
   version="1"
   Name="ElieTest"
   Description="Permission set containing my custom permission">
   <IPermission class=
      "System.Security.Permissions.EnvironmentPermission"
      Read="USERNAME"/>
   <IPermission class=
      "System.Security.Permissions.FileIOPermission"
      Read="C:\Windows"/>
   <IPermission class=
      "System.Security.Permissions.SecurityPermission"
      Flags="Execution"/>
   <IPermission class=
      "System.Security.Permissions.UIPermission"
      Unrestricted="true"/>
</PermissionSet>
```

The <*PermissionSet*> root element must be present and has a class attribute with the value *NamedPermissionSet* or *System.Security.NamedPermissionSet*. For this version of the .NET Framework, the version attribute is *1*. The *Name* attribute is the name of the permission set as it appears in the MSCorCfg tree list, and the *Description* attribute is any arbitrary description of the permission set, which appears in the MSCorCfg right-hand pane. The <*PermissionSet*> element can contain any number of <*IPermission*> elements, which represent the permissions in the permission set—these can be Framework library classes or custom permission classes.

C. Configuration

Copyright © 2010-2011, eliematta.com. All rights reserved |

Now we can start applying the XML as a first step to the security policy in one of two ways. The first approach uses MSCorCfg: select the Permission Sets node for the policy level you want (in this case Machine), and click the Create New Permission Set link. Select the **Import a permission set from an XML file** option and we specified the path to the XML file containing the permissions, as shown in this figure:



Using CASpol instead, you use this command line:

caspol -*machine* -addpset elie.xml

where –*machine* indicates the policy level to add the permission set to, and elie.xml is the file containing the required permissions. CASpol will prompt to make sure you want to proceed—enter **Y** to confirm. Whichever way you've set up these permissions, you can now examine them in MSCorCfg. (You might need to close and reopen MSCorCfg before the changes are displayed.) You can also list the permissions with this CASpol command line:

caspol -*machine* -listpset

This command will produce a listing of all permission sets for the specified policy level. Somewhere in the middle of this list we should recognize your custom permission set:

Copyright © 2010-2011, eliematta.com. All rights reserved |

8. ElieTest (Permission set containing my custom permission) =
<PermissionSet class="System.Security.NamedPermissionSet" version="1"
Name="ElieTest" Description="Permission set containing my custom permission">
  <IPermission class="System.Security.Permissions.EnvironmentPermission, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1" Read="USERNAME" />
  <IPermission class="System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1" Read="C:\Windows" />
  <IPermission class="System.Security.Permissions.SecurityPermission, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1" Flags="Execution" />
  <IPermission class="System.Security.Permissions.UIPermission, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1" Unrestricted="true" />
</PermissionSet>

The second operation is to add a new code group, including its membership condition and permission set. In the following command, All_Code is the parent code group to which we want to add a new child, the membership condition is a specified file URL, the permission set is *ElieTest*, the new code group will be named Elie_Group, and we want this to apply exclusively:

caspol -addgroup All_Code -url file://C:/environmentpermtest/* ElieTest -n Elie_Group -exclusive on

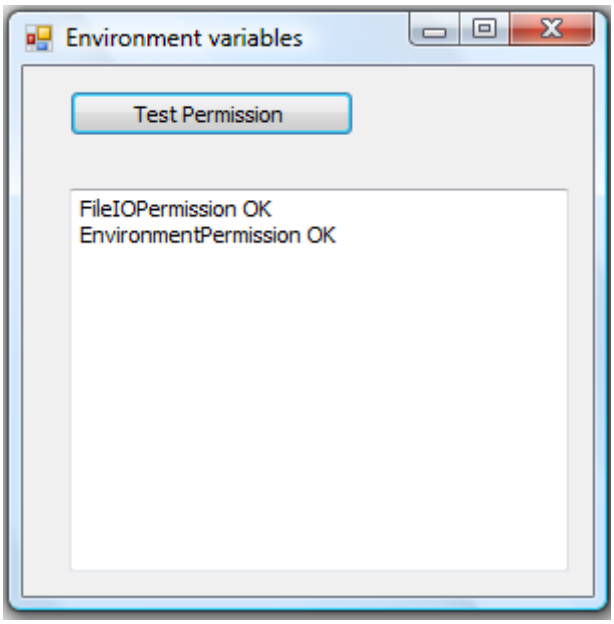
```
c:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE>caspol -addgroup All_Code -url file://C:/environmentpermtest/* ElieTest -n Elie_Group -exclusive
Microsoft (R) .NET Framework CasPol 2.0.50727.4016
Copyright (c) Microsoft Corporation.  All rights reserved.

The operation you are performing will alter security policy.
Are you sure you want to perform this operation? (yes/no)
y
Added union code group with "-url" membership condition to the Machine level.
Success
```

We created the C:\environmentpermtest directory to test it in our implementation.

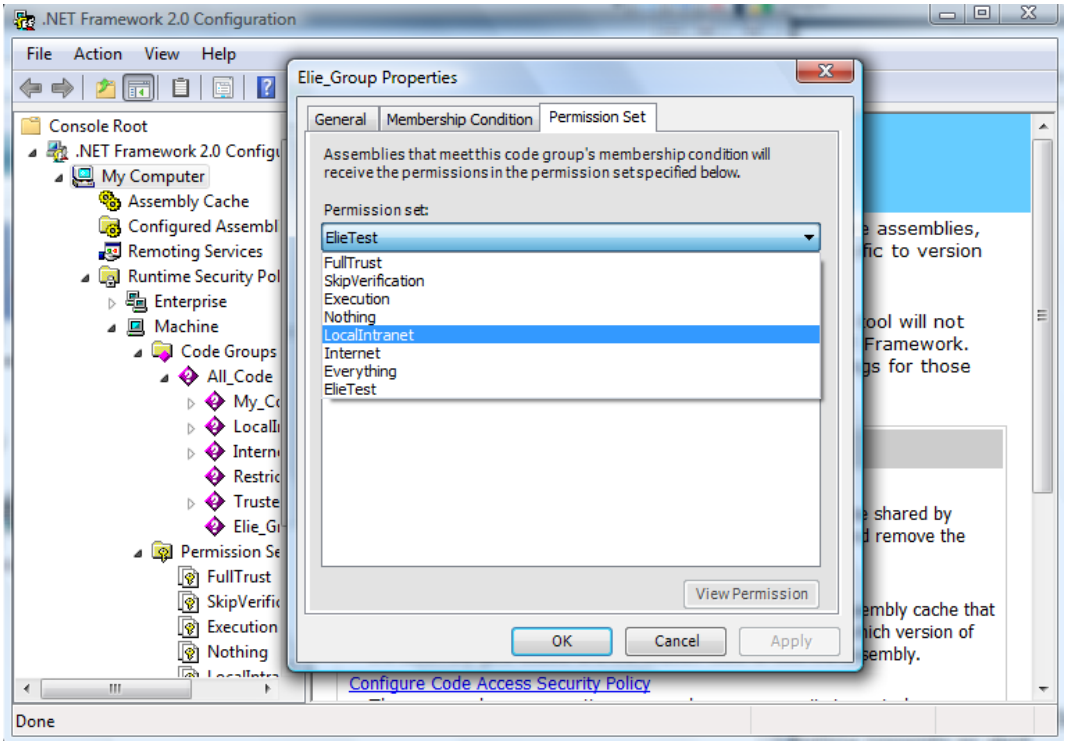
D. Implementation

Build and test the application as is before making any coding or security configuration changes

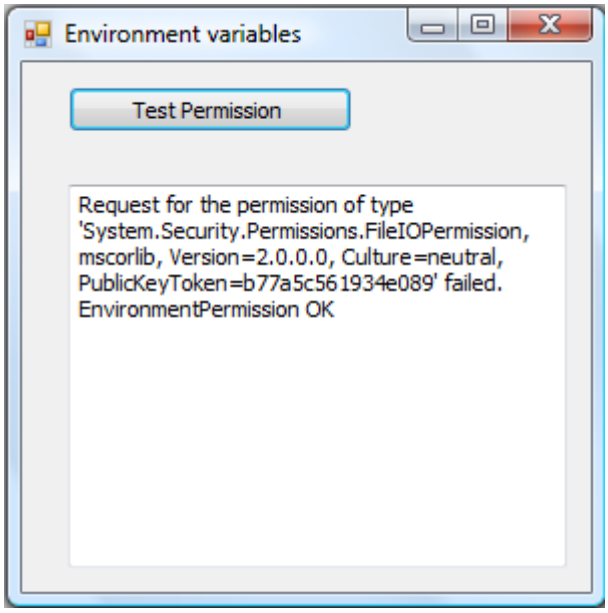
Copyright © 2010-2011, eliematta.com. All rights reserved |



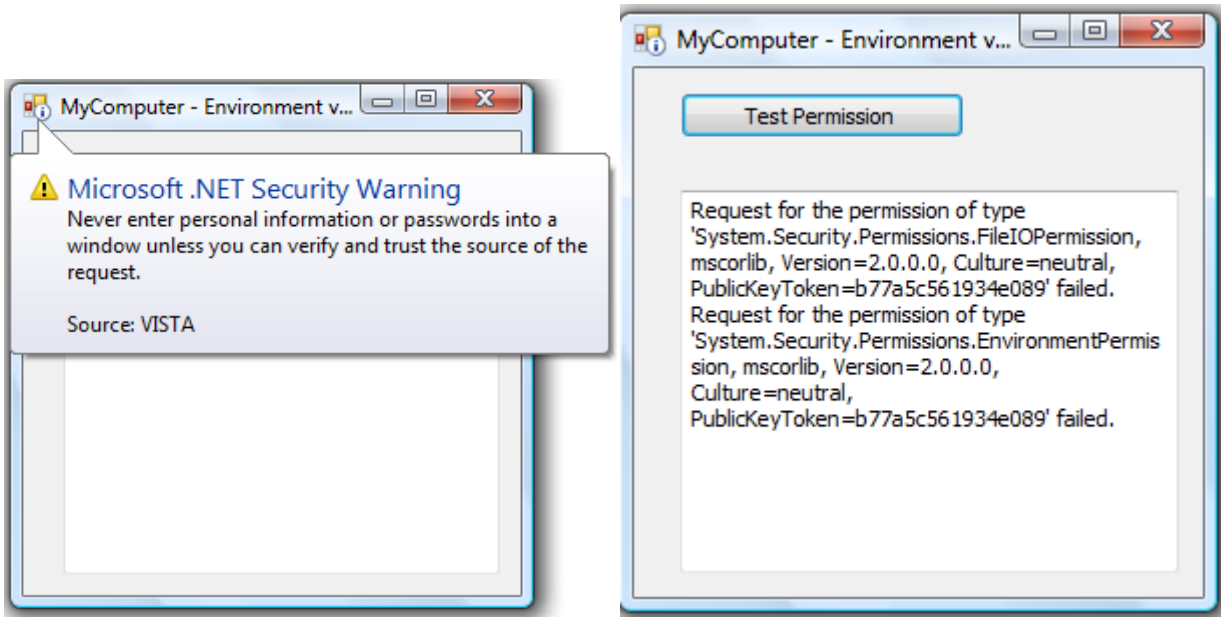
Now to test the application. From Windows Explorer, copy the envartest.exe assembly into the newly configured C:\environmentpermtest directory and double-click it to run it from there. At this stage, everything should run as before, with all permission requests granted. For the second test, in the MSCorCfg snap-in, select the Elie_Group code group and click the Edit Code Group Properties link. From the dialog box, select the Permission Set tab and change the permission set for this code group to, say, LocalIntranet. Then click OK.



We'll obtain the following when we try to run envartest.exe from C:\environmentpermtest



Now try to run the application again: you should find that although the application still executes and the request for the specific Environment permission succeeds, the request for File IO permission fails. Change the permission set again, this time to Internet. This time, although the application executes, the runtime presents an alert message (shown in the figure below) indicating that the assembly is running in a partially trusted security context and warning you that some functionality might not be available.
You can click the alert message to remove it. Indeed, when you click the Test button, you'll find that neither of the requested permissions has been granted.



Copyright © 2010-2011, eliematta.com. All rights reserved |

If we make one final test and change the permission set to *Nothing*, we'll find that the application won't even execute and the runtime will throw an exception. To round out your understanding of the various predefined permission sets (as well as any custom sets we've set up), we could experiment with replacing the permission requests as indicated in the following code, first to *AllAccess* for the root of C:\, and then to *PermissionState.Unrestricted*:

```
FileIOPermission p = new FileIOPermission(
//FileIOPermissionAccess.Read, "C:\\Windows");
//FileIOPermissionAccess.AllAccess, "C:\\");
PermissionState.Unrestricted);
```

Copyright © 2010-2011, eliematta.com. All rights reserved |

Example 4 – User Interface

Source code path: Examples/Example4/uipermset

Using URL – Imperative security

A. Introduction

In this example we will test the UIPermission with both FileIOPermission and RegistryPermission to test respectively the behavior of our window, the access to our local drives and to our registry.
We should first define the type of windows that our code is allowed to use, defined in this table:

Member name	Description
AllWindows	Users can use all windows and user input events without restriction.
NoWindows	Users cannot use any windows or user interface events. No user interface can be used.
SafeSubWindows	Users can only use SafeSubWindows for drawing, and can only use user input events for user interface within that subwindow. Examples of **SafeSubWindows** are a MessageBox, common dialog controls, and a control displayed within a browser.
SafeTopLevelWindows	• Users can only use SafeTopLevelWindows and **SafeSubWindows** for drawing, and can only use user input events for the user interface within those top-level windows and subwindows. • When it runs under **SafeTopLevelWindows** permission, your application: • Will show the DNS name or IP address of the Web site from which the application was loaded in its title bar. • Will display Balloon tool-tip when it first displays, informing the user that it is running under a restricted trust level. • Must display its title bar at all times. • Must display window controls on its forms. • Cannot minimize its main window on startup. • Cannot move its windows off-screen. • Cannot use the Opacity property on Form to make its windows less than 50% transparent.

Copyright © 2010-2011, eliematta.com. All rights reserved |

	• Must use only rectangular windows, and must include the window frame. Windows Forms will not honor setting FormBorderStyle to **None**. • Cannot make windows invisible. Any attempt by the application to set the Visible property on its **Form** objects to **False** will be ignored. • Must have an entry in the Task Bar. • Will have its controls prohibited from accessing the Parent property. By implication, controls will also be barred from accessing siblings - i.e., other controls at the same level of nesting. • Cannot control focus using the Focus method. • Will have restricted keyboard input access, so that a form or control can only access keyboard events for itself and its children. • Will have restricted mouse coordinate access, so that a form or control can only read mouse coordinates if the mouse is over its visible area. • Cannot set the TopMost property. • Cannot control the z-order of controls on the form using the BringToFront and SendToBack methods. • These restrictions help prevent potentially harmful code from spoofing attacks, such as imitating trusted system dialogs.

B. Code

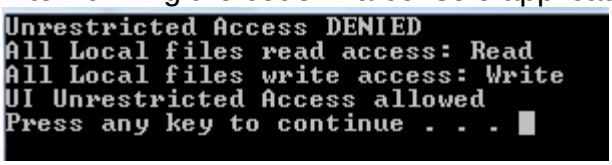
```
class MainClass

    {

        public static void Main()

        {

            RegistryPermission f = new
RegistryPermission(RegistryPermissionAccess.Read,
"HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0");


            if (f.IsUnrestricted())

                Console.WriteLine("Unrestricted Access allowed");

            else

                Console.WriteLine("Unrestricted Access DENIED");
```

Copyright © 2010-2011, eliematta.com. All rights reserved |

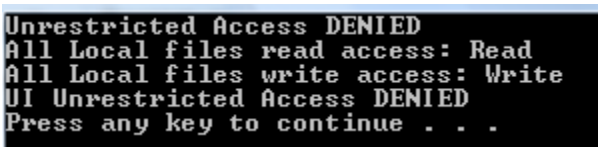
```
            FileIOPermission fileIO = new
FileIOPermission(PermissionState.None);

            Console.WriteLine("All Local files read access: {0}",
FileIOPermissionAccess.Read);

            Console.WriteLine("All Local files write access: {0}",
FileIOPermissionAccess.Write);



            UIPermission ui = new
UIPermission(UIPermissionWindow.AllWindows,
UIPermissionClipboard.AllClipboard);

            if (ui.IsUnrestricted())

                Console.WriteLine("UI Unrestricted Access allowed");

            else

                Console.WriteLine("UI Unrestricted Access DENIED");


        }

    }
```

C. ***Implementation***

After running the code in a console application, we will have a result like this:

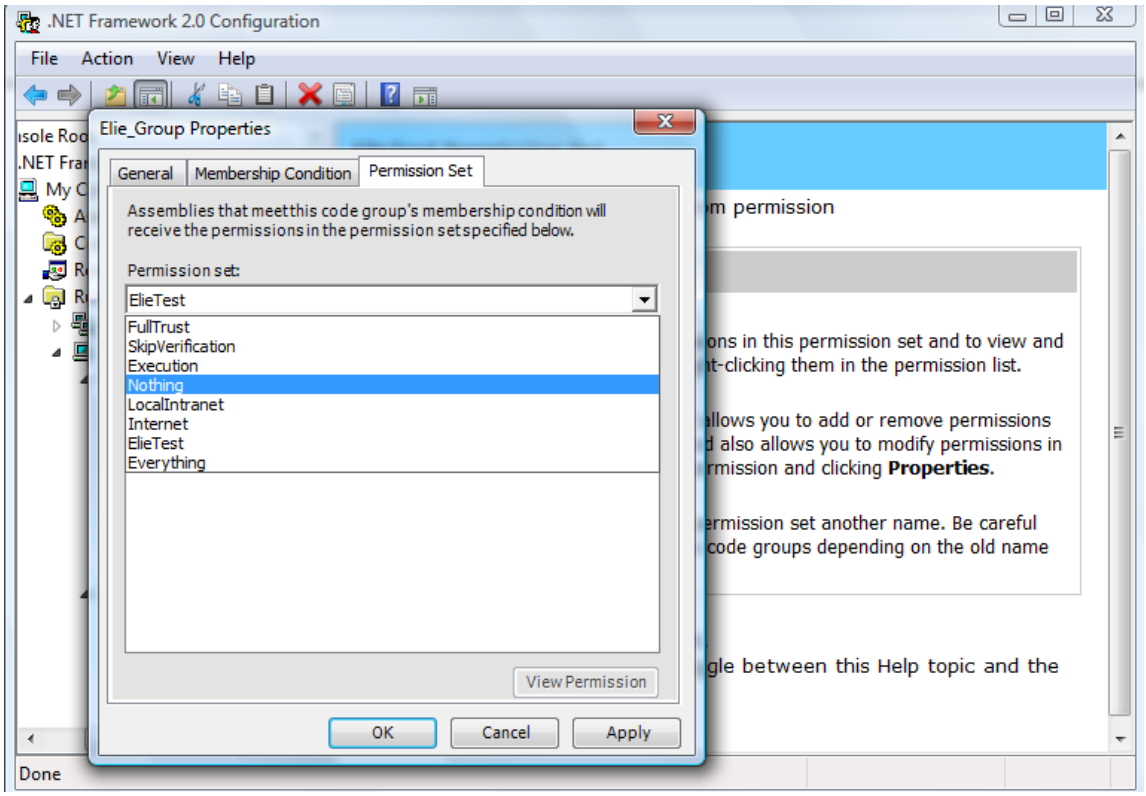


That's because we are running the code while grating access to see and use the window normally but we don't have permission to the required registry path.
If we simply change the UIPermissionWindow to NoWindows then we'll obtain:



Now if we put the uipermtest.exe in the C:\environmentpermtest while changing the Elie_Group permission to Nothing:

Copyright © 2010-2011, eliematta.com. All rights reserved |



The code will fail to start and an exception will be raised:

```
C:\>cd C:\environmentpermtest

C:\environmentpermtest>uipermtest.exe

Unhandled Exception: System.IO.FileLoadException: Could not load file or assembly 'uipermtest, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null' or
sion to execute. (Exception from HRESULT: 0x80131418)
File name: 'uipermtest, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null' ---> System.Security.Policy.PolicyException: Execution permission cannot
   at System.Security.SecurityManager.ResolvePolicy(Evidence evidence, PermissionSet reqdPset, PermissionSet optPset, PermissionSet denyPset, Permission
   at System.Security.SecurityManager.ResolvePolicy(Evidence evidence, PermissionSet reqdPset, PermissionSet optPset, PermissionSet denyPset, Permission
n checkExecutionPermission)
```

Copyright © 2010-2011, eliematta.com. All rights reserved |

Example 5 – Web access

Source code path: Examples/Example5/webacctest

Using strong name – Imperative security

A. Introduction

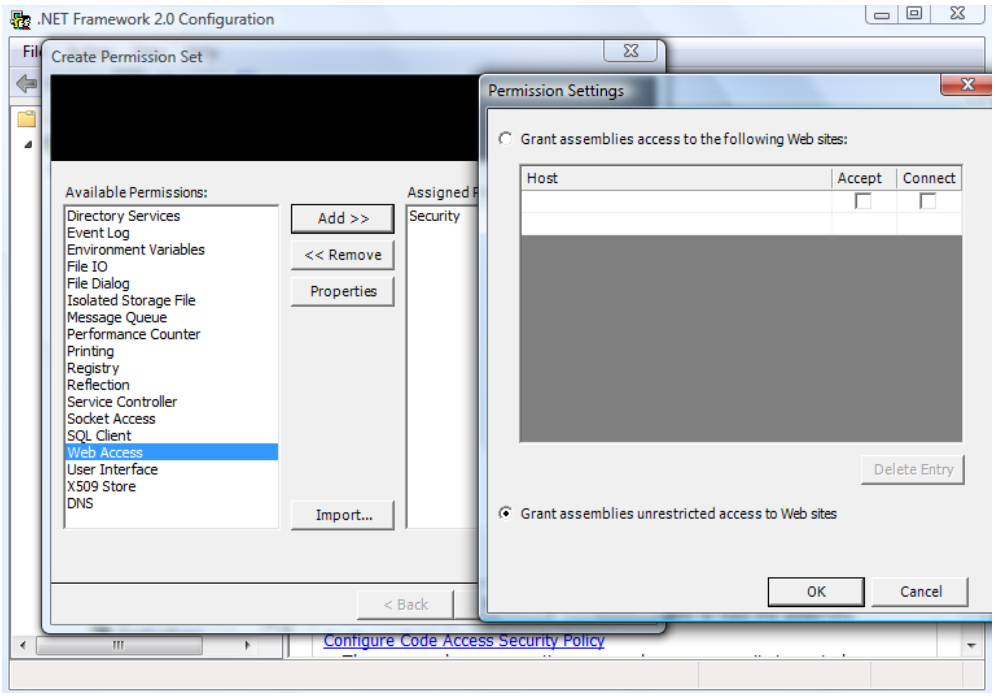
Our example consist of connecting and accepting webpermissions to google.com, msn.com and yahoo.com.
We will use regular expressions, also referred to as regex to provide a concise and flexible means for matching strings of text in this case on www.google.com, and IEnumerator for a simple iteration over a collection.
We will use a Console application adding the System.Text.RegularExpressions, System.Net and System.Collections namespaces.

B. .NET Framework Configuration

In this part we will use the permission set *NewPermSet* and code group *NewCodeGroup* already used in Example 1.

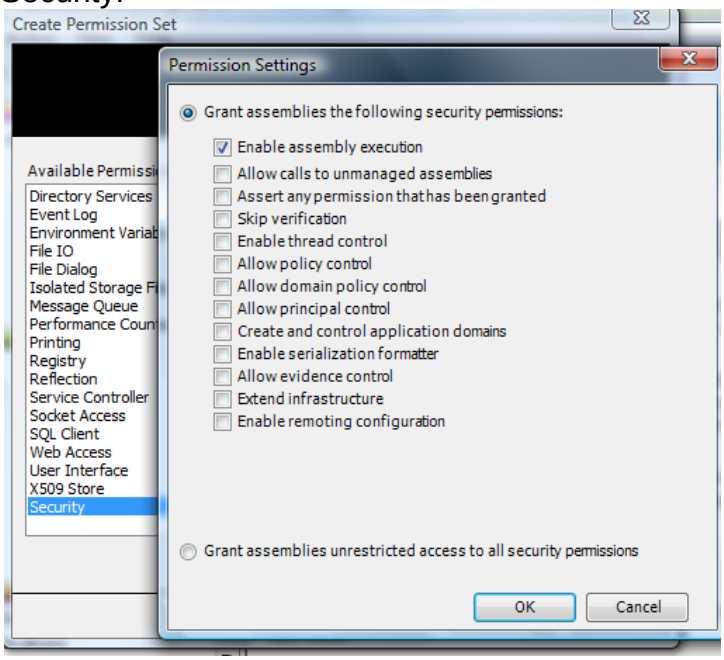
Modifying permission set

Expand the **Runtime Security Policy** node. You can see the security policy levels - Enterprise, Machine and User. We are going to change the security settings in Machine policy. We will add the Security and Web access and grant it unrestricted access because we will control it in the code.

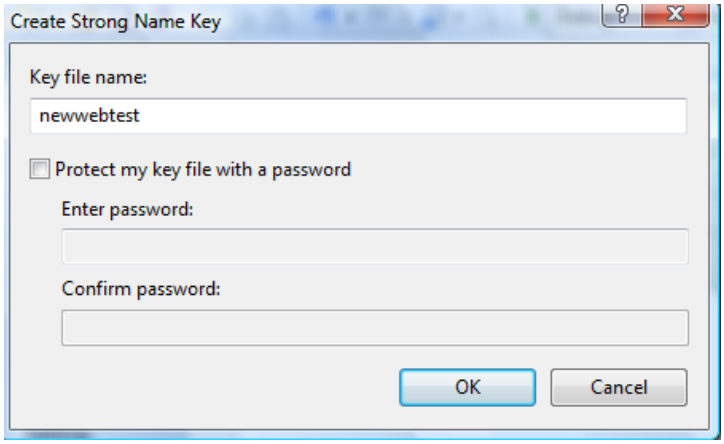


Copyright © 2010-2011, eliematta.com. All rights reserved |

Security:

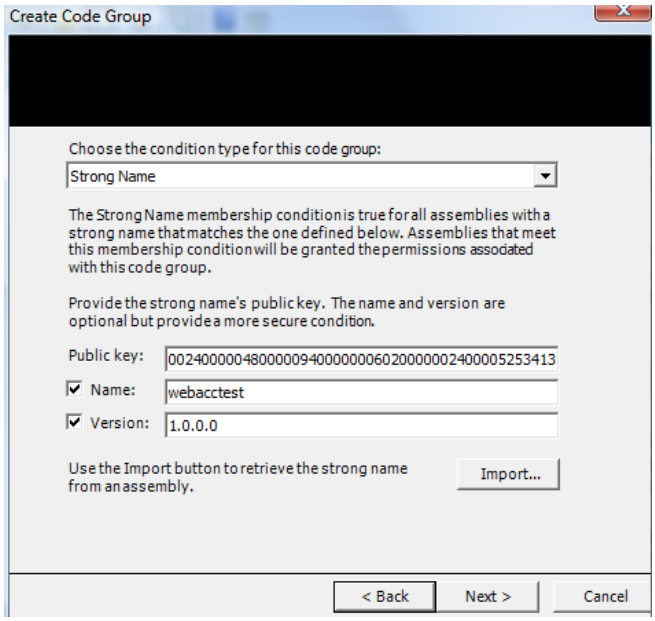


For this example, we are going to use the **Strong Name** condition type. First, sign your assembly (by using sn.exe per example or from the Visual studio by right-clicking on the project name, properties, signing) with a strong name and build the project.

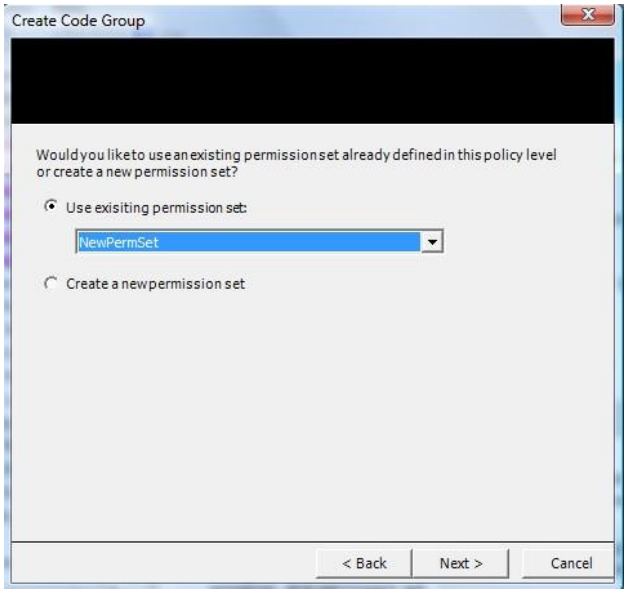


Back to the .NET configuration tool, now press the **Import** button and select your assembly. Public Key, Name and Version will be extracted from the assembly.

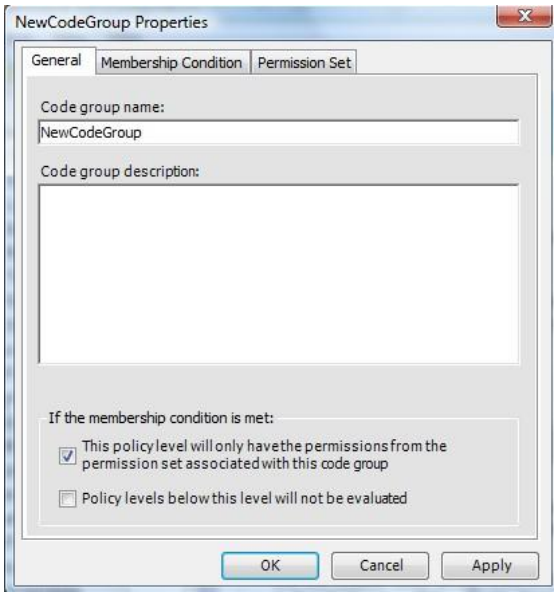
Copyright © 2010-2011, eliematta.com. All rights reserved |



Now move on to the next figure. We have to specify a permission set for our code group. Since we have already created one – *NewPermSet*, select it from the list box.



Now we will go to .NET configuration and set the option **"This policy level will only have the permissions from the permission set associated with this code group"**



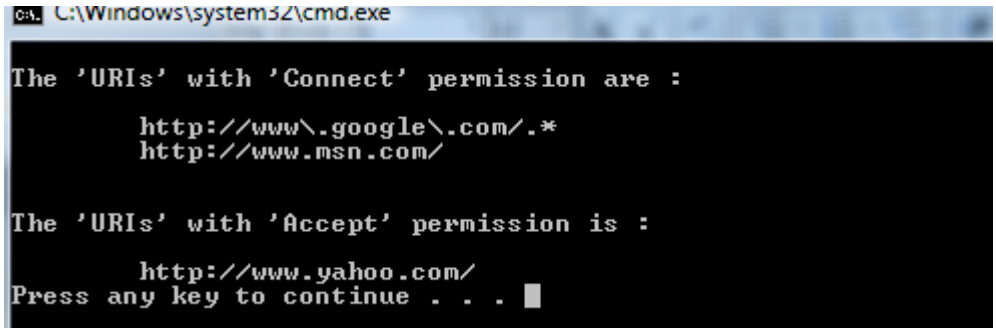
C. ***Code***

```
//  Create a Regex that accepts all URLs containing the host fragment
www.google.com.
            Regex myRegex = new Regex(@"http://www\.google\.com/.*");

            // Create a WebPermission that gives permissions to all the hosts
containing the same host fragment.
            WebPermission myWebPermission = new
WebPermission(NetworkAccess.Connect, myRegex);

            //Add connect privileges for a www.msn.com.
            myWebPermission.AddPermission(NetworkAccess.Connect,
"http://www.msn.com");

            //Add accept privileges for www.yahoo.com.
            myWebPermission.AddPermission(NetworkAccess.Accept,
"http://www.yahoo.com/");

            // Check whether all callers higher in the call stack have been
granted the permission.
            myWebPermission.Demand();

            // Get all the URIs with Connect permission.
            IEnumerator myConnectEnum = myWebPermission.ConnectList;
            Console.WriteLine("\nThe 'URIs' with 'Connect' permission are
:\n");
            while (myConnectEnum.MoveNext())
            { Console.WriteLine("\t" + myConnectEnum.Current); }

            // Get all the URIs with Accept permission.
            IEnumerator myAcceptEnum = myWebPermission.AcceptList;
            Console.WriteLine("\n\nThe 'URIs' with 'Accept' permission is
:\n");
```

```
while (myAcceptEnum.MoveNext())
{ Console.WriteLine("\t" + myAcceptEnum.Current); }
```

D. Implementation

It's time to run the code. What we have done so far is, we have put our code into a code group and given unrestricted access to Web Access. Run the code it should work fine, giving access to www.google.com/* which is everything that follows www.google.com and accepts connection from www.yahoo.com:



Copyright © 2010-2011, eliematta.com. All rights reserved |

Example 6 – Printing

Source code path: Examples/Example6/printtest

Using strong name – Declarative security

A. Introduction

Our example consist of demanding printing permission to print a document by a default printer, we will also use permission flags like RequestMinimum explained in Document 1 – page 5
We will use a Windows Form Application which we will add on it two namespaces: System.Drawing.Printing and System.Security.Permissions.
First there are essential types of access that we can allow to our code which are:

- **AllPrinting** – Provides full access to all printers.
- **DefaultPrinting** – Provides printing programmatically to the default printer, along with safe printing through showing a less restricted dialog box. DefaultPrinting is a subset(a subset is a set contained with another set) of AllPrinting.
- **NoPrinting** – Prevents access to printers. NoPrinting is a subset of SafePrinting.
- **SafePrinting** - Provides printing only from showing a restricted dialog box. SafePrinting is a subset of DefaultPrinting.

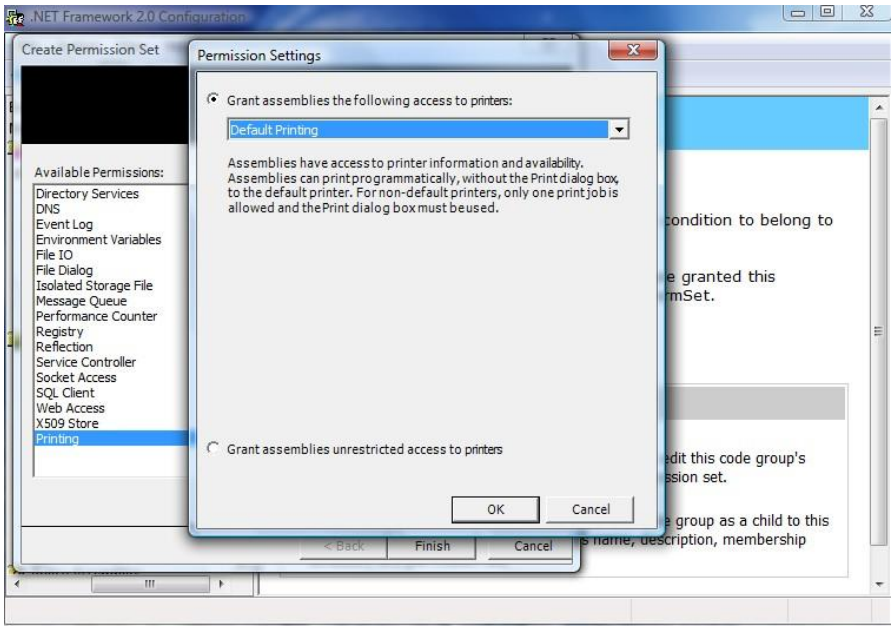
B. .NET Framework Configuration

In this part we will use the permission set *NewPermSet* and code group *NewCodeGroup* already used in Example 1.

Modifying permission set

Expand the **Runtime Security Policy** node. You can see the security policy levels - Enterprise, Machine and User. We are going to change the security settings in Machine policy. We will add the Security, User Interface and grant it unrestricted access, we will also add Printing with permission: Default Printing :

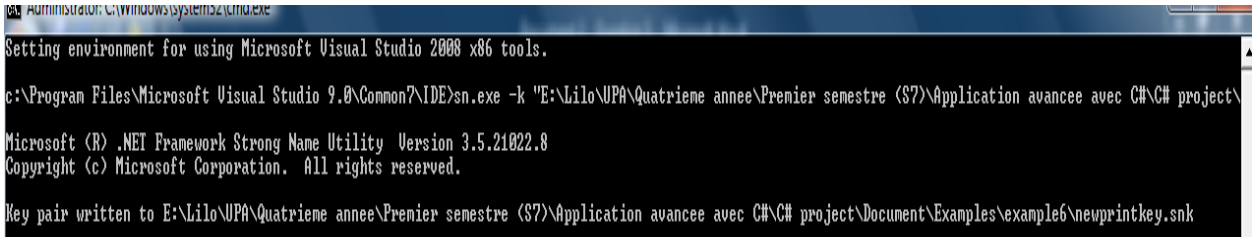
Copyright © 2010-2011, eliematta.com. All rights reserved |



For this example, we are going to use the **Strong Name** condition type. This time we will sign the assembly by using cmd (Command Prompt) by typing the following command in the correct path:

sn.exe –k “path\nameofthekey.snk”

In our case:
sn.exe -k "E:\Lilo\UPA\Quatrieme annee\Premier semestre (S7)\Application avancee avec C#\C# project\Document\Examples\example6\newprintkey.snk”



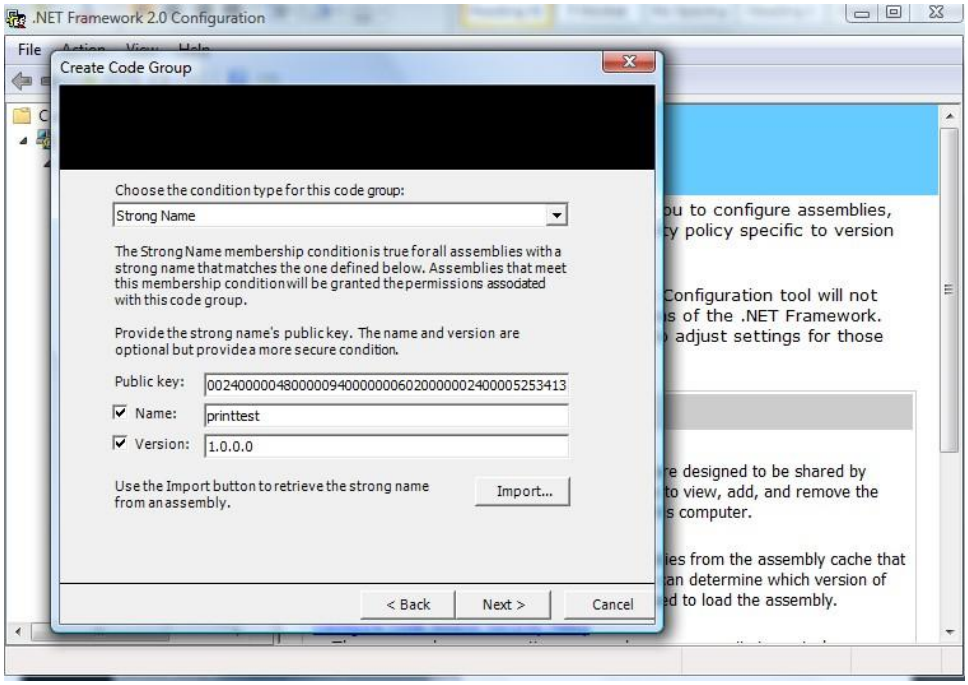
A key is generated, but it is not yet associated with the assembly of the project. To create this association, double-click the AssemblyInfo.cs file in Visual Studio .NET Solution Explorer. This file has the list of assembly attributes that are included by default when a project is created in Visual Studio .NET. Modify the **AssemblyKeyFile** assembly attribute in the code as follows:

```
[assembly: AssemblyDelaySign(false)]
[assembly: AssemblyKeyFile("E:\\Lilo\\UPA\\Quatrieme annee\\Premier semestre
(S7)\\Application avancee avec C#\\C#
project\\Document\\Examples\\example6\\newprintkey.snk")]
[assembly: AssemblyKeyName("")]
```

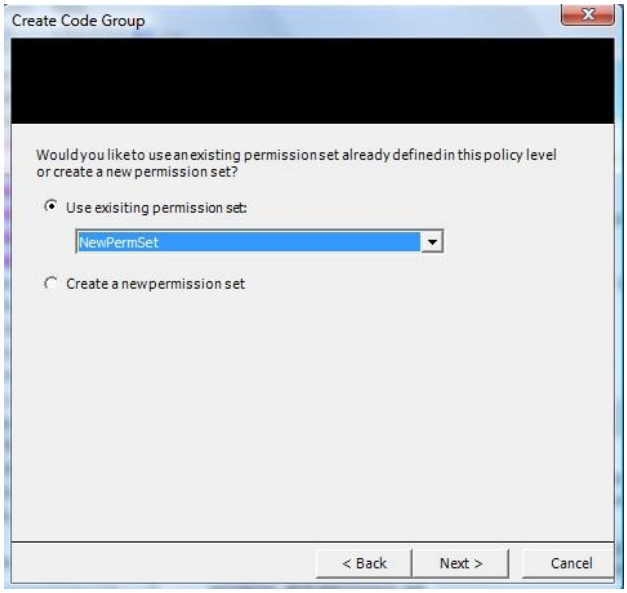
Copyright © 2010-2011, eliematta.com. All rights reserved |

Compile the project by clicking CTRL+SHIFT+B. You do not have to have any additional code to install a .dll file in the GAC.

Back to the .NET configuration tool, now press the **Import** button and select your assembly. Public Key, Name and Version will be extracted from the assembly.

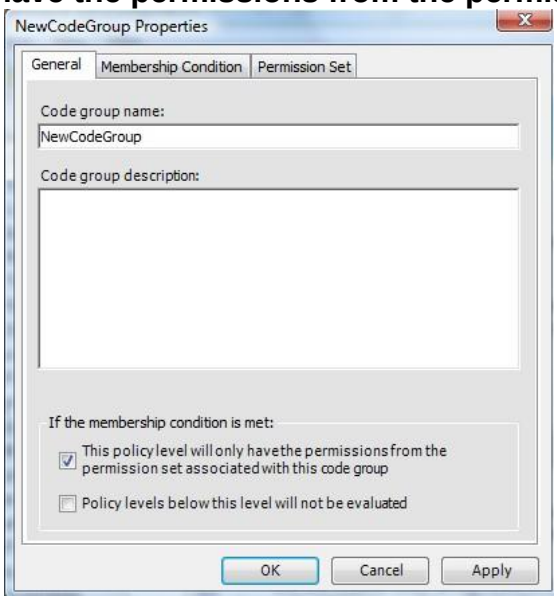


Now move on to the next figure. We have to specify a permission set for our code group. Since we have already created one – *NewPermSet*, select it from the list box.



Copyright © 2010-2011, eliematta.com. All rights reserved |

Now we will go to .NET configuration and set the option **"This policy level will only have the permissions from the permission set associated with this code group"**



C. Code

In the AssemblyInfo.cs:

```
[assembly: PrintingPermission(SecurityAction.RequestMinimum,Level =
PrintingPermissionLevel.DefaultPrinting)]
```

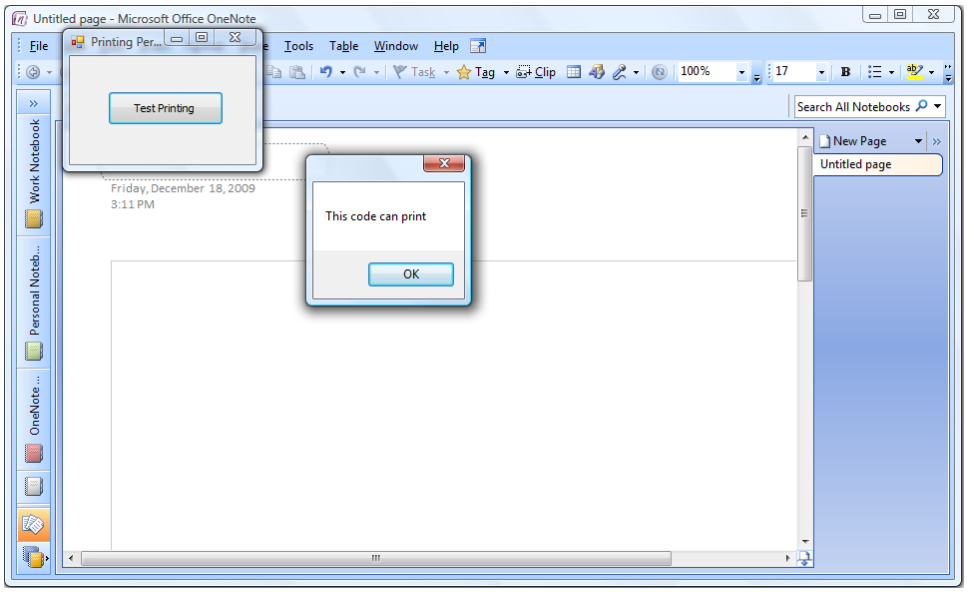
Button #1:

```
            try
            {
                PrintDocument mydoc = new PrintDocument();
                mydoc.Print();
                MessageBox.Show("This code can print");
            }
            catch (Exception ex)
            {
                MessageBox.Show("This code cannot print because "
+ex.Message);
            }
```

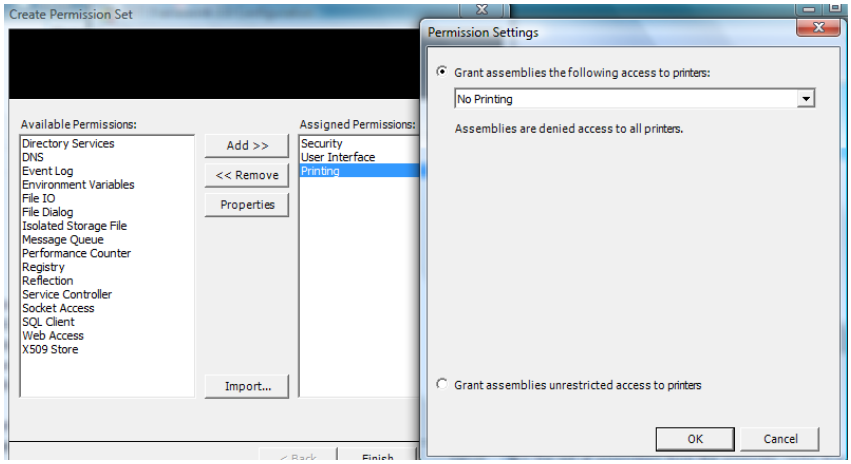
D. Implementation

It's time to run the code. What we have done so far is, we have put our code into a code group and given unrestricted access to User Interface and Security. Run the code it should work fine, printing a document by the default printer, in our case the default printer is to send the document to Microsoft Office OneNote:

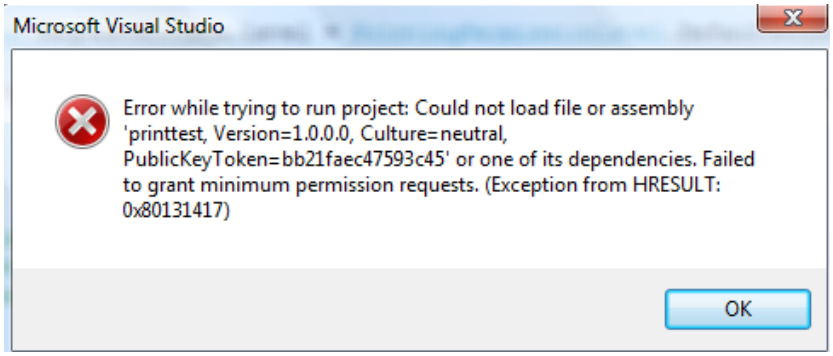
Copyright © 2010-2011, eliematta.com. All rights reserved |



On the other hand, if we go back to .NET configuration tool and change the printing permission to No Printing, we will have an error as soon as we try to runt the code because we don't have the minimum permission to run the code and to print.



The error:



Copyright © 2010-2011, eliematta.com. All rights reserved |

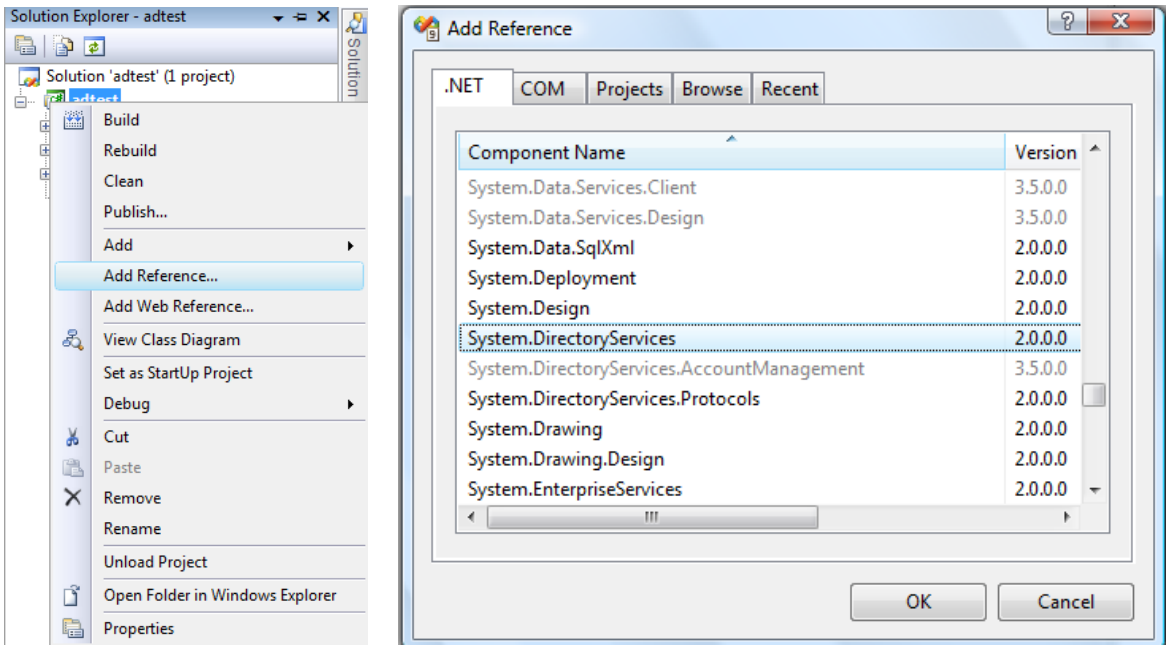
Example 7 – The active directory

Source code path: Examples/Example7/adtest

Using publisher – Imperative security

A. Introduction

Our example consist of connecting to the active directory on our computer and listing all the users that belongs to the administrators group.
We will use IEnumerator for a simple iteration over a collection.
We will also have to add the System.DirectoryServices namespace manually if it doesn't exist, as shown in the figure below:



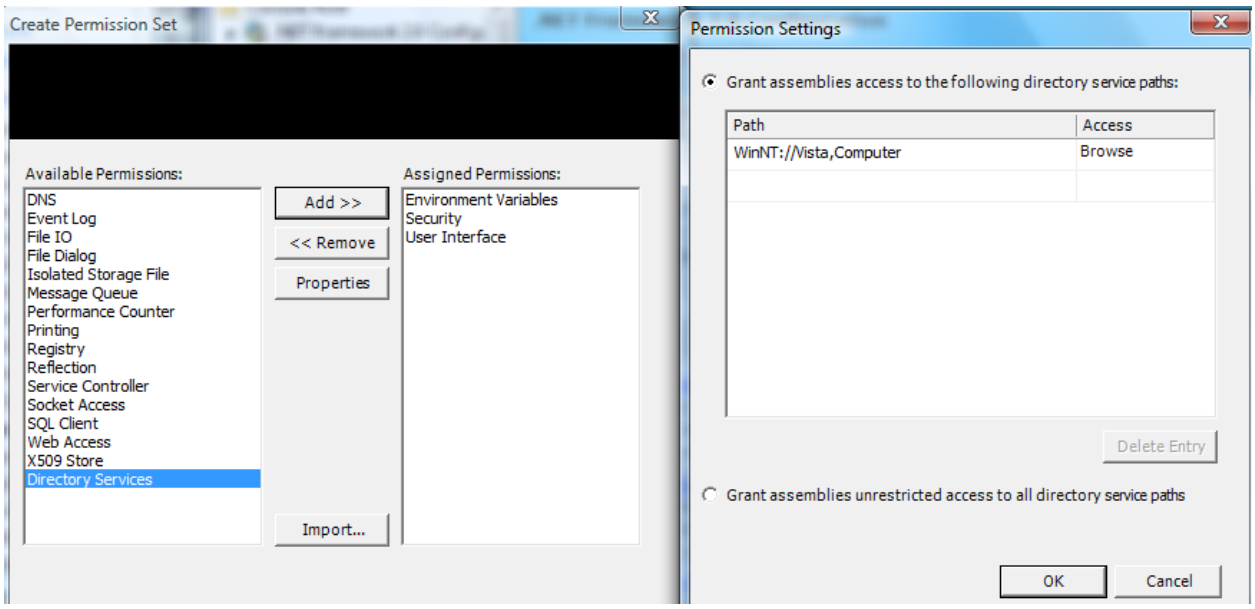
B. .NET Framework Configuration

In this part we will use a new code group *ADCodeGroup* and a new permission set *ADPermSet*.

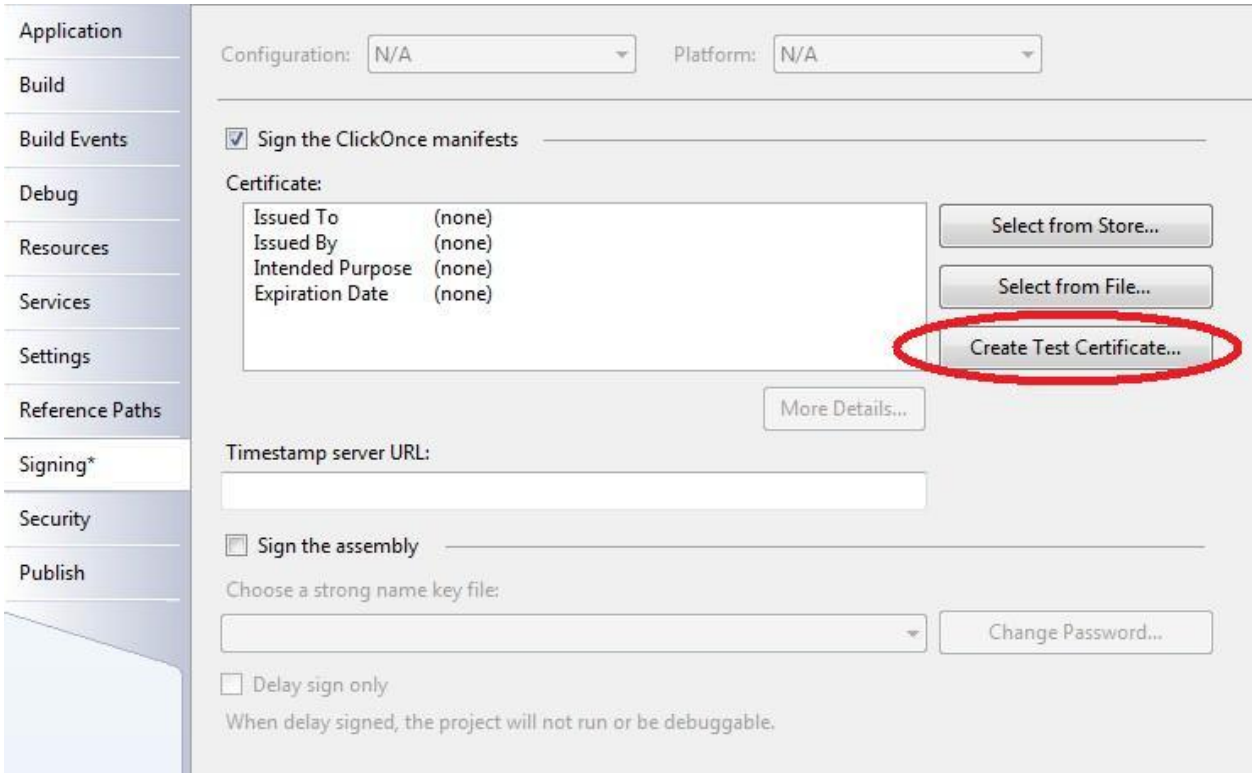
a. Creating a new permission set

We will add the Security, User Interface and Environment variables and grant them unrestricted access to be able to bring the names of the administrators, and we will add the Directory Services with path **"WinNT://Vista,Computer"** to browse through the active directory as shown in the next figure:

Copyright © 2010-2011, eliematta.com. All rights reserved |

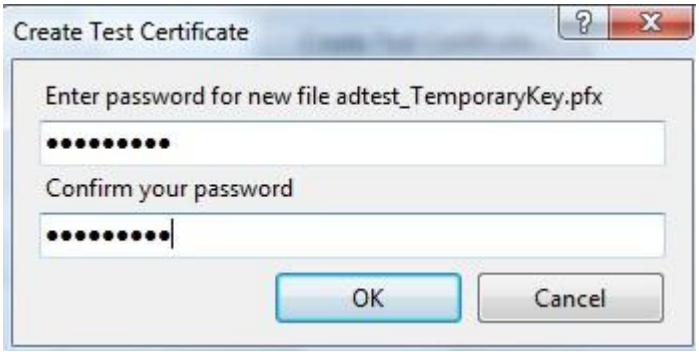


For this example, we are going to sign the certificate and import it to the **publisher** in .NET framework. First, we will sign our assembly by the ClickOnce manifests (from the Visual studio by right-clicking on the project name, properties, signing) by creating a test certificate.

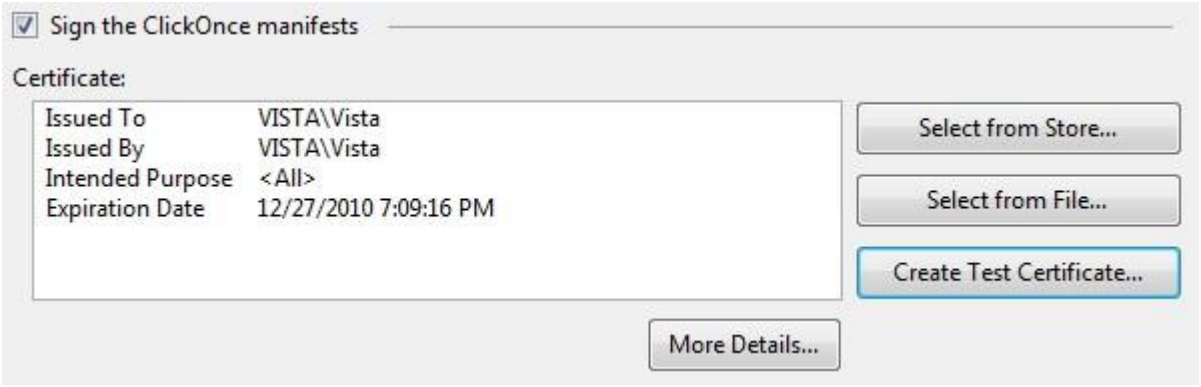


Then we will have to enter a password, we've put: *mynewpass*

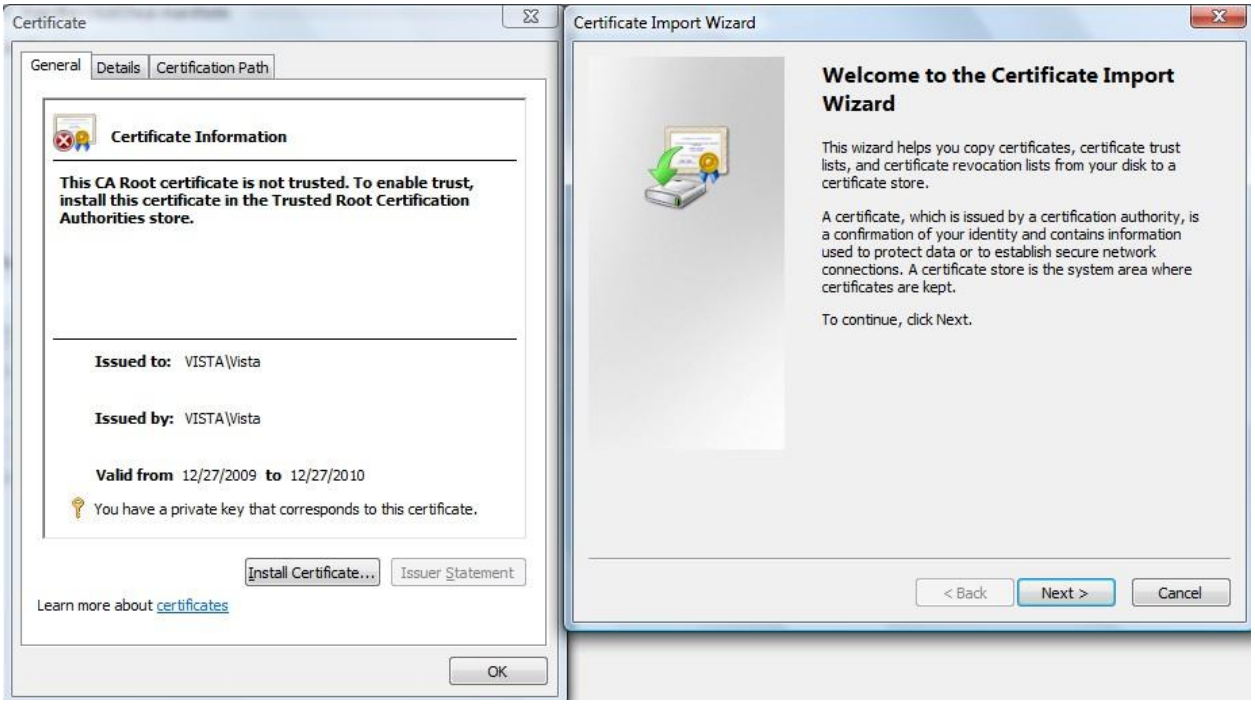
Copyright © 2010-2011, eliematta.com. All rights reserved |



Note that as from now on the certificate has been created as we can notice the (none) fields has been replaced by the newly created certificate:

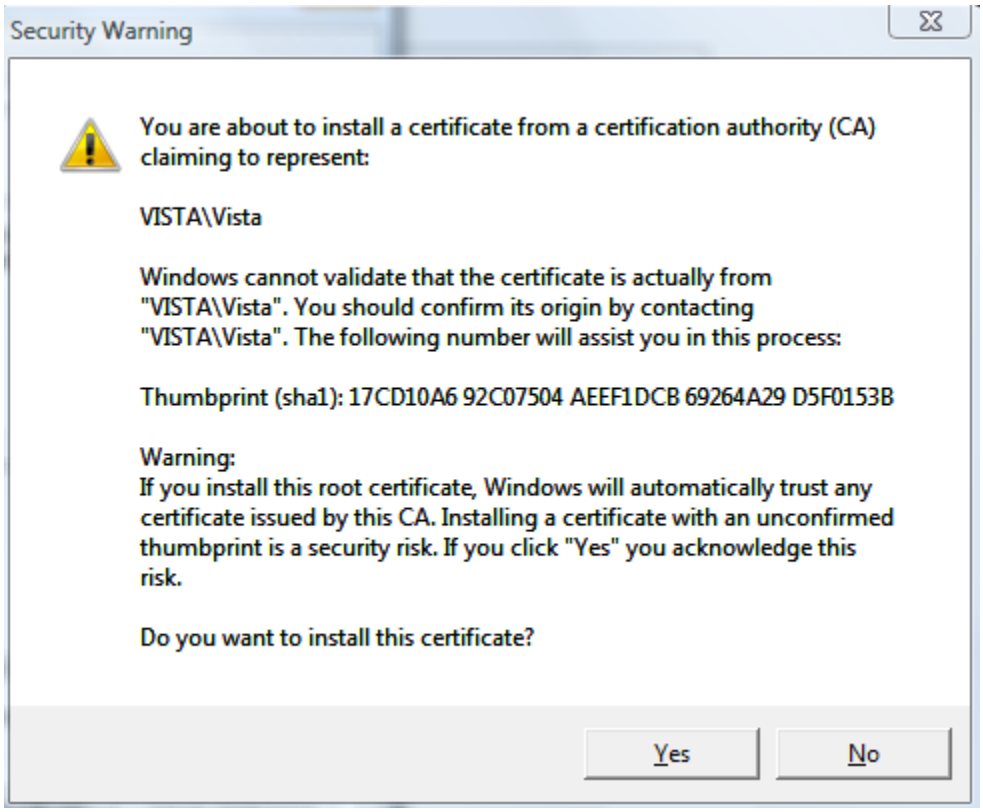


Then, click on the More Details button and then click on the install certificate:



Copyright © 2010-2011, eliematta.com. All rights reserved |

Click on next, then you should choose the Place all certificate in the following store and click Browse then select "**Trusted Root Certification Authorities**" to sign properly this certificate to a high level. Finally click on finish, you might have this security warning that tells you that a newly certificate has been added to the root certification list and it is not trusted by Microsoft:

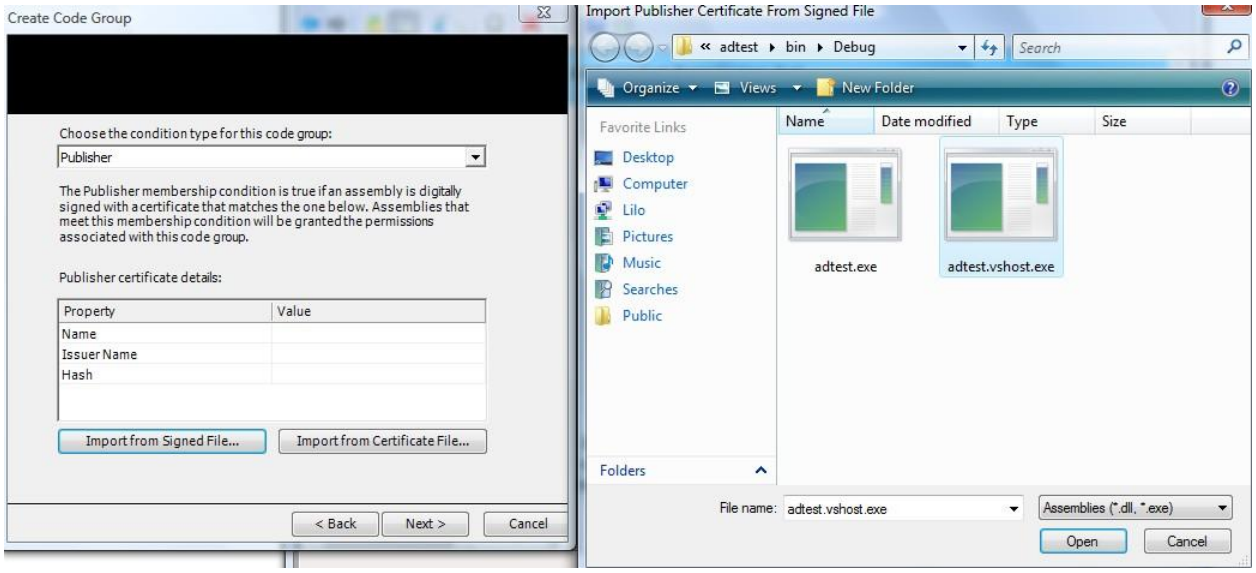


Click on yes, and the import should be successful.

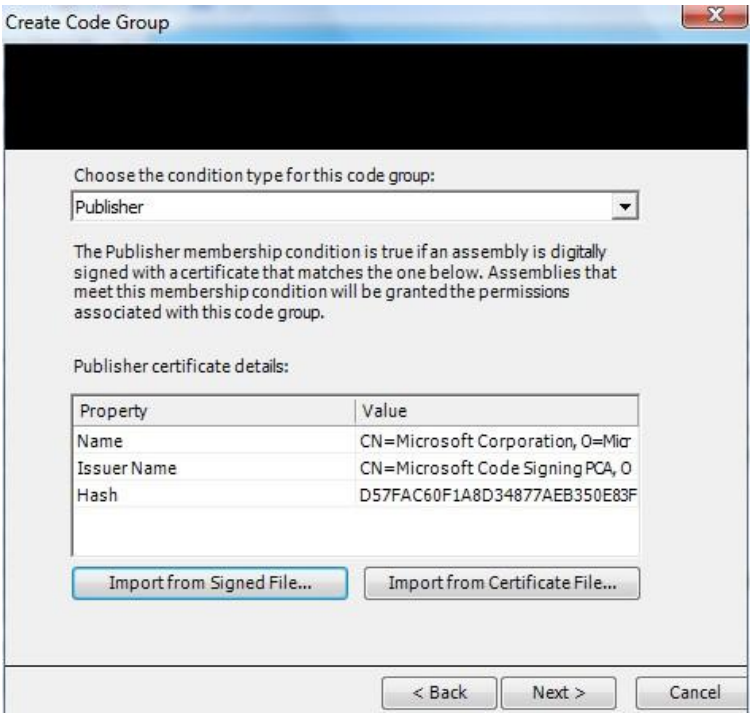
b. Creating a new code group

Back to the .NET configuration tool, we will choose the **Publisher** for the condition type, then click on Import from Signed file and choose the name of your project with an extension: **.vshost.exe** as shown in the figure below:

Copyright © 2010-2011, eliematta.com. All rights reserved |

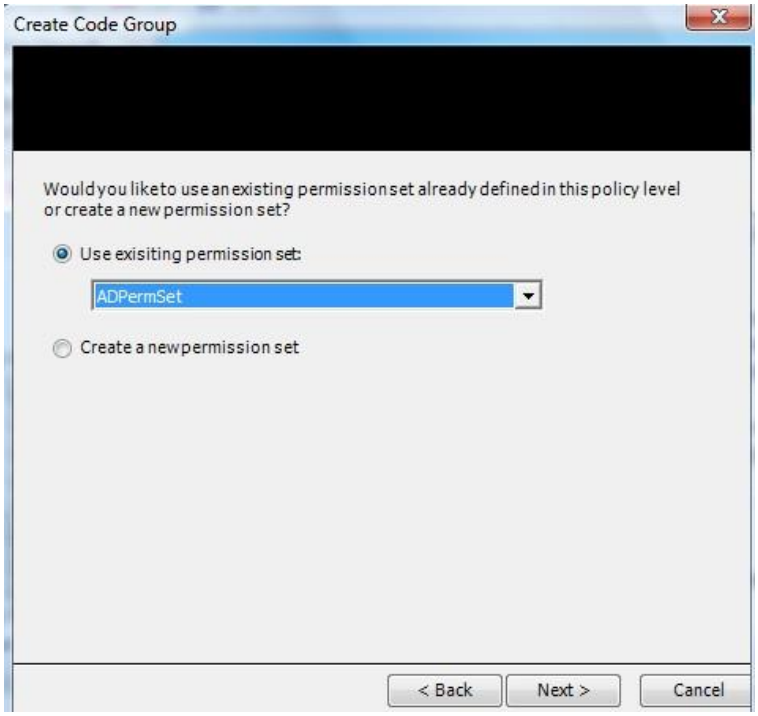


The import should be successful and all the fields should be filled automatically:

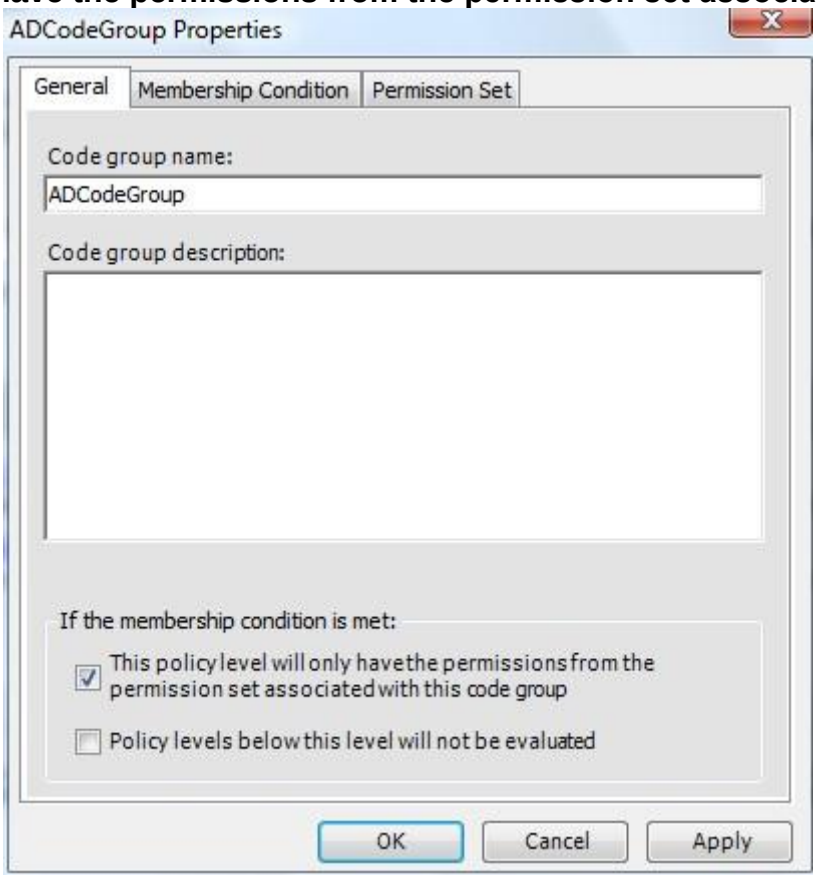


Click on Next, and choose *ADPermSet*:

Copyright © 2010-2011, eliematta.com. All rights reserved |



Now we will go to .NET configuration and set the option **"This policy level will only have the permissions from the permission set associated with this code group"**



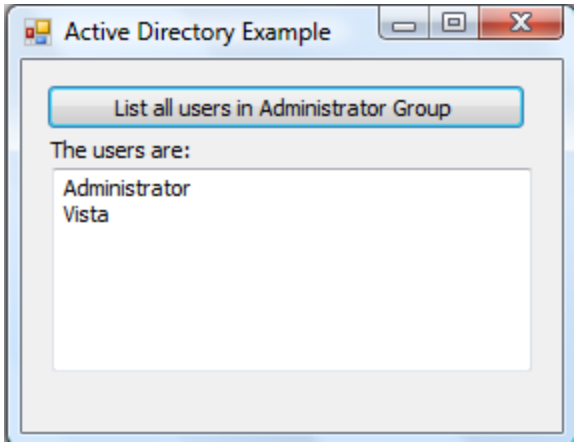
Copyright © 2010-2011, eliematta.com. All rights reserved |

C. Code

```
            try
            {
                DirectoryServicesPermission dsp = new
DirectoryServicesPermission(DirectoryServicesPermissionAccess.Browse,
"WinNT://" + Environment.MachineName + ",Computer");
                DirectoryEntry localmachine = new DirectoryEntry("WinNT://" +
Environment.MachineName + ",Computer");
                DirectoryEntry admgroup =
localmachine.Children.Find("administrators", "group");
                object members = admgroup.Invoke("members", null);

                foreach (object groupmember in (IEnumerable)members)
                {
                    DirectoryEntry member = new DirectoryEntry(groupmember);
                    textBox1.Text += "" + member.Name + "\r\n";

                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
```

D. Implementation

It's time to run the code. What we have done so far is, we have put our code into a code group and given unrestricted access to Security and User Interface and to Environment variables, and to browse into the Directory Services. Run the code it should work fine like shown in the next figure:

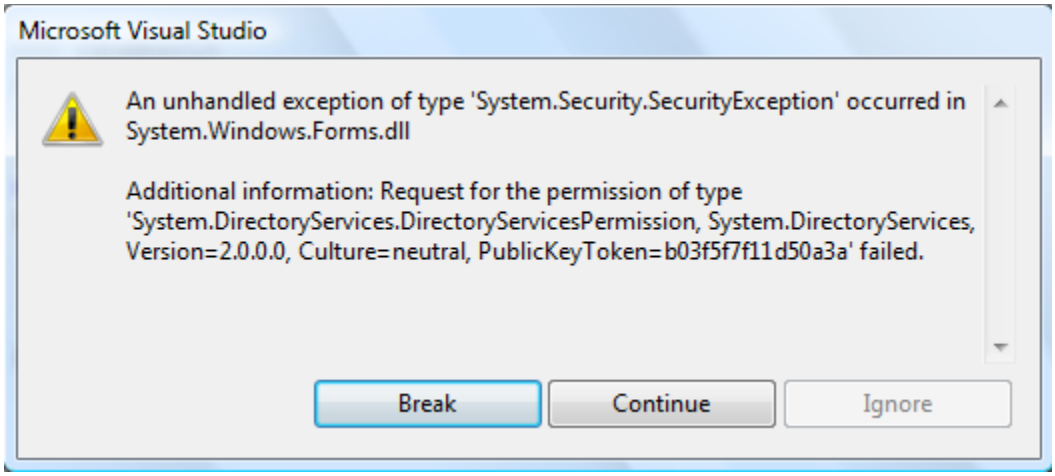


If we tried to modify the code to write into the active directory:

Copyright © 2010-2011, eliematta.com. All rights reserved |

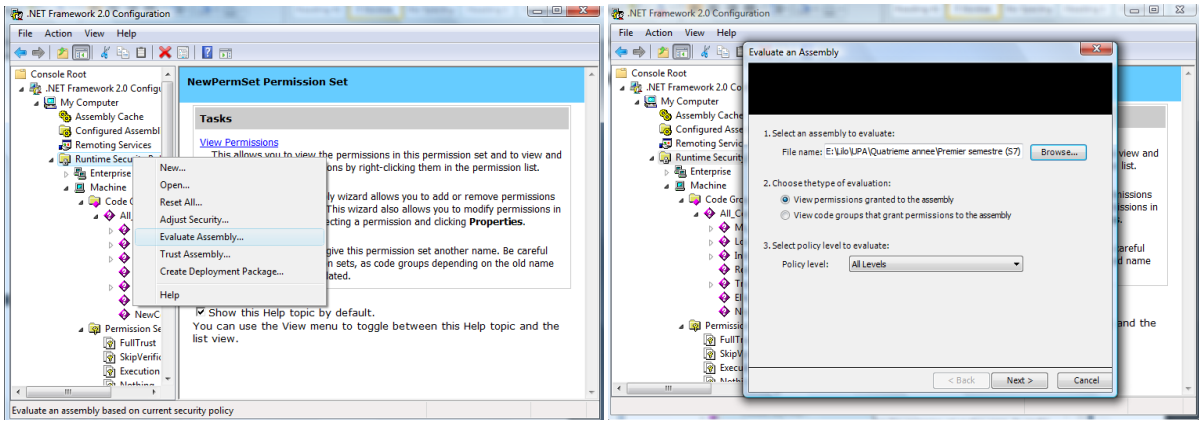
```
DirectoryServicesPermission dsp = new
DirectoryServicesPermission(DirectoryServicesPermissionAccess.Write,
"WinNT://" + Environment.MachineName + ",Computer");
```

Then we will have eventually an error:

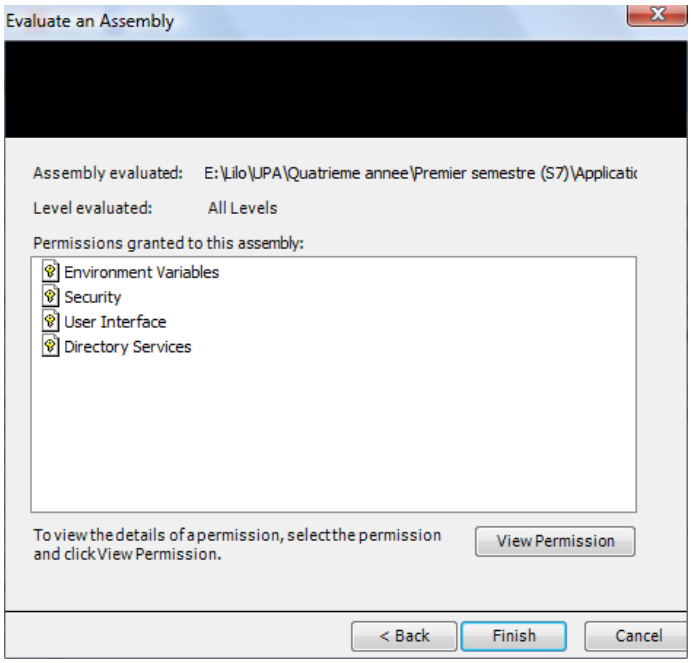


Note:

You can at any time evaluate the assembly to check at which permission or code group the selected assembly have, just follow the steps as shown in the figures below:



This is the evaluation of the adtest assembly:



Copyright © 2010-2011, eliematta.com. All rights reserved |