## Developing a Sandboxed Solution with Web Parts

**Lab Time:** 45 minutes

**Lab Folder**: C:\Student\Labs\SandboxSolutions

**Lab Overview:** The support for sandboxed solutions provides an important new deployment mechanism for SharePoint. With a development methodology that is the same for ‘full’ solutions the key focus point is not on what to build, but on what can be built. In this lab exercise you will construct a basic Web Part that will call into the SharePoint API to retrieve some information. Next it will try and use SPSecurity to try and elevate privileges. The third and last action that is added is an attempt to initiate a HTTP connection to an external site.

**Lab Setup Requirements**

* Before you begin this lab, you must run the batch file named **SetupLab.bat** in the **[[LAB FILES]]** folder. This batch file runs a PowerShell script which creates a new SharePoint site collection at the location **http://intranet.wingtip.com/sites/Sandbox**.

### Exercise 1: Creating the Visual Studio 2010 Project

In this first exercise you will create a new SharePoint project and set it up for sandboxed deployment. You will also author and test the first version of the Web Part.

1. Before you can deploy sandboxed solutions to the SharePoint server, your server must be configured for this.
2. Open **SharePoint 2010 Central Administration**.
3. Choose **System Settings » Manage Services on Server**.
4. Locate the **Microsoft SharePoint Foundation Sandbox Service** and check its status. If it’s **Stopped**, click the **Start** hyperlink to start the service.
5. Open Visual Studio 2010 and choose to create a new project. Pick **Empty SharePoint Project** as the template which you find under the **Visual C# » SharePoint » 2010** template group. Name the project **SandboxedWebPart**. A wizard will appear allowing you to configure the project.
6. Complete the **SharePoint Customization Wizard** using the following information:

**Debugging site:** http://intranet.wingtip.com/sites/sandbox

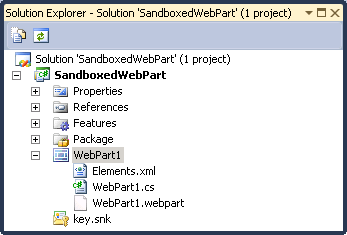
**Deploy as a sandboxed solution**: selected

1. Using the **Solution Explorer**, right-click the **SandboxedWebPart** project and select **Add » New Item**. Complete the dialog that appears using the following information.

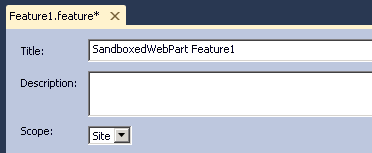
**Visual C# » SharePoint » 2010**

**Template:** Web Part

**Name:** WebPart1



1. Using the **Solution Explorer**, expand **Features**, right-click **Feature1** and choose **View Designer**. In the designer that appears, verify the **Scope**. It should be set to **Site**.



1. In the WebPart1.cs tab that opens, locate the WebPart1 class. Clear the code inside the class definition. Now add the following code:

protected override void CreateChildControls()

{

Label message = new Label();

Controls.Add(message);

Controls.Add(new WebControl(HtmlTextWriterTag.Br));

Button testButton1 = new Button();

testButton1.Text = "Test 1";

testButton1.Click += delegate

{

message.Text = String.Format("This site contains {0} lists",

SPContext.Current.Web.Lists.Count);

};

Controls.Add(testButton1);

}

1. Right click the project in Solution Explorer and choose **Deploy** in order to deploy the web part.
2. Open a browser and navigate to the **http://intranet.wingtip.com/sites/sandbox** site.
3. Using the ribbon, select **Page » Edit Page**.
4. Select a **Web Part Zone** and then using the ribbon select **Editing Tools »** **Insert » Web Part**.
5. On the Web Part pane, select the **Custom** group, then select **WebPart1** and click the **Add** button.
6. On the ribbon, click the **Stop Editing** button to return to display mode.



1. On the **WebPart1**, click the **Test 1** button in order to test calling the context site collection. The page should refresh displaying a message of how many lists are in the current site:



In this exercise you created a new sandboxed Web Part that contained code that is allowed to run in the sandbox.

### Exercise 2: Calling SPSecurity

In this second exercise you will add code to the Web Part that will call SPSecurity. You will re-deploy and test the Web Part by simply starting debugging from Visual Studio.

1. In the WebPart1 class, add the following code to the CreateChildControls() method that you have added earlier. This code will utilize the SPSecurity object.

Notice as you type the type SPSecurity, Visual Studio’s IntelliSense does not recognize it as a valid object, but once you type the entire line out, Visual Studio won’t add the red underlined squiggle line indicating a syntax error. This is because when creating sandboxed solutions, you are still building against the full SharePoint object model, but Visual Studio uses a different IntelliSense file to help the developer know which objects they will not have access to when the component runs in the sandbox.

protected override void CreateChildControls()

{

// CODE OMMITTED FOR BREVITY

Controls.Add(testButton1);

Button testButton2 = new Button();

testButton2.Text = "Test 2";

testButton2.Click += delegate

{

try

{

SPSecurity.RunWithElevatedPrivileges(

delegate

{

using (SPSite siteCollection = new SPSite(SPContext.Current.Site.ID))

{

SPWeb site = siteCollection.OpenWeb(SPContext.Current.Web.ID);

message.Text = String.Format("This site contains {0} lists",

site.Lists.Count);

}

});

}

catch (Exception e)

{

message.Text = e.Message;

}

};

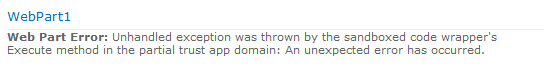
Controls.Add(testButton2);

}

1. Right click the project in Solution Explorer and choose **Deploy** in order to deploy the web part.
2. If you have your browser still open from previous exercise, press **[F5]** to refresh the page. If you closed the browser, open it again and navigate to **http://intranet.wingtip.com/sites/sandbox**.



1. On the **WebPart1**, click the **Test 2** button in order to test the call to SPSecurity.
2. Notice that an exception is thrown. It shows that this Web Part, which is running in the sandbox, cannot include a reference to the SPSecurity type:



1. Close the browser.

In this exercise you added code to your sandboxed Web Part that included a reference to SPSecurity which is not permitted within the sandbox.

### Exercise 3: Calling a CAS secured method

In this exercise you will add code to the Web Part that tries and perform a CAS security protected task. In the sample a HTTP connection is attempted.

1. In the WebPart1 class, add the following code to the CreateChildControls() method that you have added earlier. This code will issue a standard Web request.

protected override void CreateChildControls()

{

// CODE OMMITTED FOR BREVITY

Controls.Add(testButton1);

// CODE OMMITTED FOR BREVITY

Controls.Add(testButton2);

Button testButton3 = new Button();

testButton3.Text = "Test 3";

testButton3.Click += delegate

{

try

{

System.Net.HttpWebRequest.Create("http://intranet.wingtip.com");

}

catch (Exception e)

{

message.Text = e.Message;

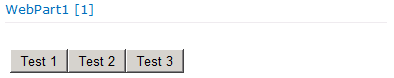
}

};

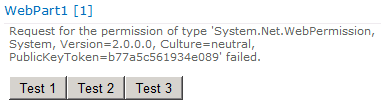
Controls.Add(testButton3);

}

1. On the **Debug** menu, choose **Start Debugging** in order to deploy and test the Web Part, or just press **[F5]**. When the browser opens, it will automatically take you to the **http://intranet.wingtip.com/sites/Sandbox** site.



1. On the **WebPart1**, click the **Test 3** button in order to test creating an HTTP connection.
2. Notice that an exception is thrown because the sandbox is running in a special CAS policy that blocks all System.Net.WebPermission demands, as the HttpWebRequest object does.



1. Close the browser in order to stop debugging.

In this exercise you added code that is blocked by the sandbox CAS policy because it demands a permission not allowed by the CAS policy.

### Exercise 4: Creating a Solution Validator

In this exercise you create a custom solution validator that will check the strong name of the assembly in a sandbox solution to ensure that the public key token matches a specifically allowed token. If it does not, the solution validator will display an error upon activation. Otherwise the solution will activate as normal.

1. Open Visual Studio 2010 and choose to create a new project. Pick **Empty SharePoint Project** as the template which you find under the **Visual C# » SharePoint » 2010** template group. Name the project **StrongNameSolutionValidator**. A wizard will appear allowing you to configure the project.
2. Complete the **SharePoint Customization Wizard** using the following information:

**Debugging site:** http://intranet.wingtip.com/sites/sandbox

**Deploy as a farm solution**: selected

1. With the project created, the first step is to create the object that will serve as the solution validator. Right-click the project and select **Add » New Item….** When prompted, select **Class** from the **Visual C#** root node category. Give it a name of AssemblyValidator.cs.
2. First step is to add a handful of using statements to the top of the file to make object references easier. Add the following using statements to the top of the AssemblyValidator.cs class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

using System.Text;

using System.Reflection;

using System.Runtime.InteropServices;

using Microsoft.SharePoint.Administration;

using Microsoft.SharePoint.UserCode;

1. Next, modify the class declaration to be decorated with a GuidAttribute attribute using a unique GUID and set it to inherit from the SPSolutionValidator class:

namespace StrongNameSolutionValidator

{

[GuidAttribute("DFC6CA0B-BD8E-462b-BD00-FF0DA24601D8")]

public class AssemblyValidator : SPSolutionValidator

{

**Note:** Feel free to use any GUID in the in the GuidAttribute attribute. To create a new GUID, use the GUIDGEN.EXE tool found in the following location: C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin

1. The next step is to define the class constructors and also to declare a constant that will contain the valid public key token. Solutions that contain signed assemblies that do not have this public key token will be considered invalid solutions.

public class AssemblyValidator : SPSolutionValidator

{

private const string VALID\_PUBLIC\_KEY\_TOKEN = "C86F2CE9BDCB9B80";

public AssemblyValidator() { }

public AssemblyValidator(string name, SPUserCodeService service)

: base(name, service) { }

1. All solution validators should implement two methods on the base SPSolutionValidator class. The first is the ValidateSolution method which provides details to all files in the solution. Add the following code to the AssemblyValidator class:

public override void ValidateSolution(SPSolutionValidationProperties properties)

{

// loop through all files in the solution & check the key

foreach (SPSolutionFile file in properties.Files.Where(

f => string.Equals(

Path.GetExtension(f.Location),

".dll",

StringComparison.InvariantCultureIgnoreCase)))

{

ValidateAssembly(properties, file);

}

}

1. Next, implement the method ValidateAssembly which will be called for each assembly in the solution:

public override void ValidateAssembly(SPSolutionValidationProperties properties, SPSolutionFile assembly)

{

string tempFolder = Path.GetTempPath();

string tempFile = Path.GetTempFileName();

string tempPath = Path.Combine(tempFolder, tempFile);

try

{

// save the assembly to disk to reflect it

File.WriteAllBytes(tempPath, assembly.OpenBinary().ToArray());

// get the full name of the assembly & the public key token

AssemblyName name = AssemblyName.GetAssemblyName(tempPath);

string token = BitConverter.ToString(name.GetPublicKeyToken()).Replace("-", "").ToUpper();

// check if the key is the allowed key

if (token != VALID\_PUBLIC\_KEY\_TOKEN)

{

properties.Valid = false;

properties.ValidationErrorMessage = "Not a valid public key token.";

}

else

properties.Valid = true;

}

catch {}

finally

{

// clean up the assembly

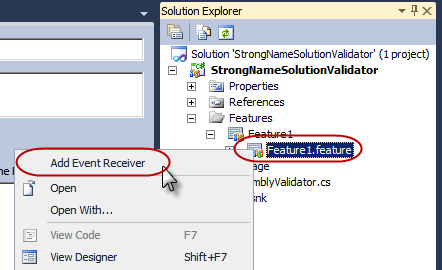
if (File.Exists(tempPath))

File.Delete(tempPath);

}

}

1. At this point the solution validator is complete. The next step is to install it by adding it to the user code service’s solution validators collection. An easy way to do this is using a farm scoped SharePoint Feature. When a farm scoped Feature is deployed it is automatically activated. The opposite is true when it is retracted. Therefore, using the simple WSP deploy/retract process and leveraging the Feature’s events, a simple solution validator installer can be implemented.   
     
   Create a new Feature by right-clicking the **Features** node in the project and selecting **Add Feature**.
2. In the Feature designer, change the **Scope** selector to **Farm**.
3. Now create the Feature’s event handlers by right-clicking **Feature1.feature** and selecting **Add Event Receiver** as shown in the following image:



1. Uncomment the FeatureActivated() and FeatureDeactivating() methods.
2. First step is to add a handful of using statements to the top of the file to make object references easier. Add the following using statements to the top of the Feature1.EventReceiver.cs class:

using System;

using System.Runtime.InteropServices;

using System.Security.Permissions;

using Microsoft.SharePoint;

using System.Linq;

using Microsoft.SharePoint.Security;

using Microsoft.SharePoint.Administration;

1. Next add a private constant string that will hold the unique identifier of the solution validator:

[Guid("0ffb882c-bf96-4af4-ad32-a6a51c3f11e9")]

public class Feature1EventReceiver : SPFeatureReceiver

{

private const string VALIDATOR\_NAME = "PublicKeyTokenValidator";

public override void FeatureActivated(SPFeatureReceiverProperties properties)

{

1. Now, within the FeatureActivated() method, add the necessary code to “install” the validator: get a reference to the user code service, check to see if the validator is present and if not, create a new instance of it and add it to the validators collection.

public override void FeatureActivated(SPFeatureReceiverProperties properties)

{

SPUserCodeService userCodeService = SPUserCodeService.Local;

// try to get a reference to the custom validator

AssemblyValidator validator = (AssemblyValidator)userCodeService.SolutionValidators.Where(

s => s.Name == VALIDATOR\_NAME).FirstOrDefault();

// if none found, add it

if (validator == null)

{

validator = new AssemblyValidator(VALIDATOR\_NAME, userCodeService);

userCodeService.SolutionValidators.Add(validator);

userCodeService.Update();

}

}

1. Next address the “uninstall” routine by implementing the FeatureDeactivating() method:

public override void FeatureDeactivating(SPFeatureReceiverProperties properties)

{

SPUserCodeService userCodeService = SPUserCodeService.Local;

// try to get a reference to the custom validator

AssemblyValidator validator = (AssemblyValidator)userCodeService.SolutionValidators.Where(

s => s.Name == VALIDATOR\_NAME).FirstOrDefault();

// if found, remove it

if (validator != null)

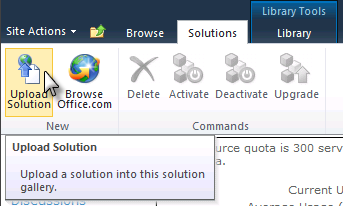
{

validator.Delete();

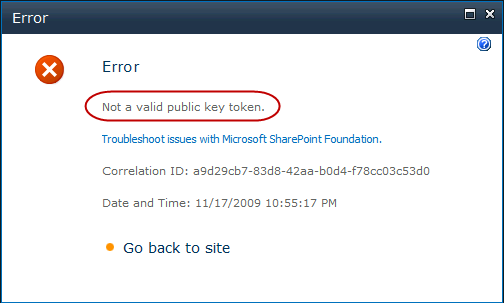
}

}

1. Lastly, build the project to ensure everything compiles ok by right-clicking the project title in the Solution Explorer tool window and select **Build**. Fix any errors that are reported.
2. Instead of pressing **[F5]** to debug the project, right-click the project title in the Solution Explorer tool window and select **Deploy**.
3. After the project deploys, open a browser and navigate to **SharePoint 2010** **Central Administration » System Settings** and select **Manage farm features**. Verify the Feature has been activated… your feature is likely named something like **StrongNameSolutionValidator Feature1**. If it is not active, go ahead and activate it.
4. Next, navigate to the **http://intranet.wingtip.com/sites/Sandbox** site. Select **Site Actions » Site Settings** and then click the **Solutions** link under the **Galleries** section.
5. Two sample sandbox solutions are provided for testing. Both solutions are identical except they were signed with two different private keys. Upload the two solutions (GoodSandboxSolution.wsp & BadSandboxSolution.wsp) found in **[[LAB FILES]]\Resources** the same way you upload any other file into a document library:



1. Both solutions will upload just fine, but try to activate each one. When activating the GoodSandboxSolution.wsp, it will activate just fine. However, when activating the BadSandboxSolution.wsp, an error dialog will be displayed. If the web.config file for the hosting Web Application has not be updated for debugging support, you will see the following dialog.



1. Now that you have built and tested a solution validator, you should remove it so it does not block future sandbox solutions you build from being deployed. Do this by deactivating the farm scoped Feature:
2. Open **SharePoint 2010 Central Administration**.
3. Select **System Settings » Manage farm Features**.
4. Click the **Deactivate** button on the **StrongNameSolutionValidator Feature1** Feature.

In this exercise you created a custom solution validator that checked the strong name of the assembly in a sandbox solution to ensure that the public key token matches a specifically allowed token. If it does not, the solution validator displayed an error upon activation. Otherwise the solution activated as normal.