Configuring App Permissions and S2S Authentication

Lab Time: 60 minutes

Lab Folder: C:\Student\Modules\06_AddinSecurity\Lab

Lab Overview: In this lab you will get your first hands-on experience working with security-related issues in the SharePoint Add-in model. You will begin by creating and testing a SharePoint-hosted add-in that requires extra permission to program against the host web. After that, you will create a provider-hosted add-in that makes CSOM and SharePoint REST API calls to the SharePoint host web using server-to-server (S2S) authentication.

Exercise 1: Setup Lab Environment

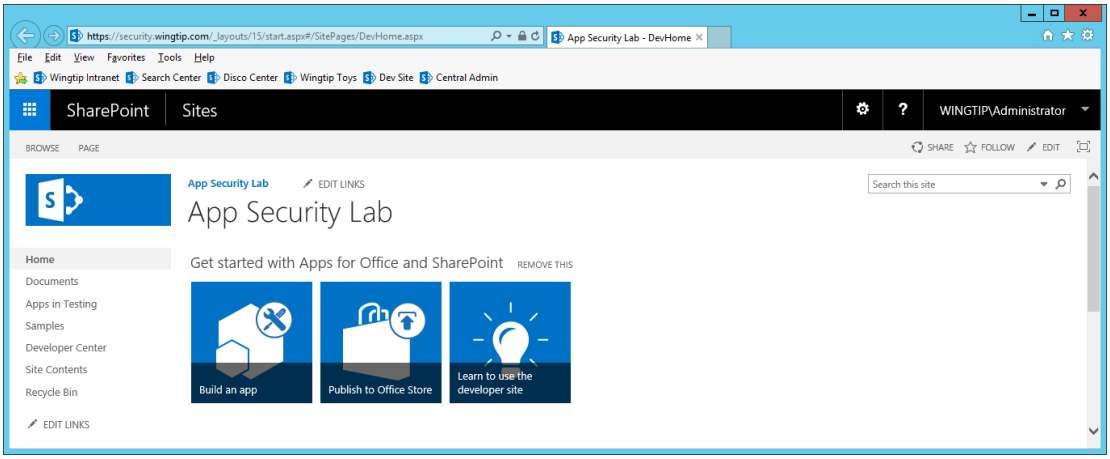
In this exercise you will setup your environment.

All exercises in this lab assume you will work in a new site collection, https://security.wingtip.com.

1. Setup a new site collection for this lab:
   a) Ensure you are logged into the **WingtipServer** server as **WINGTIP\Administrator**.
   b) Using Windows Explorer, navigate to the folder at the following path.

```
C:\Student\Modules\06_AddinSecurity\Lab\
```

   c) Locate the PowerShell script named **SetupLab.ps1** and execute it by right-click it and selecting **Run with PowerShell**.
   d) When the script completes, it will launch a new browser and navigate to the lab site collection.

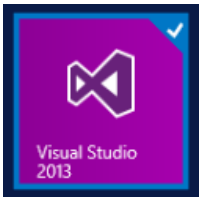


   e) Close the PowerShell console window.

Exercise 2: Create a SharePoint-Hosted App that Requires Custom Permissions

In this exercise you will create a SharePoint-Hosted App that uses the client-side object mode (CSOM) to retrieve the title of the site at the top of the current site collection. This CSOM call will fail unless the app requests and is granted the proper permissions during the installation of the app.

2. Launch **Visual** Studio **2015** as administrator:
   a) Windows Keyboard Key → Right click on the **Visual Studio 2015** tile and select **Run as administrator**.



3. Create a new SharePoint provider-hosted app project in Visual Studio 2015.
   a) In Visual Studio select **File → New → Project**.
   b) Fill out the **New Project** dialog as follows…
      i) Find the **SharePoint Add-in** template under the **Installed → Templates → Visual C# → Office / SharePoint → Apps** section.
      ii) **Name**: AppPermissionsLab
      iii) **Location:** C:\Student\Modules\06_AddinSecurity\Lab
      iv) Click **OK**
   c) In the **New SharePoint Add-in** wizard, use the following values to complete the wizard and click **Finish**.
      i) **What SharePoint site do you want to use for debugging your app?** https://security.wingtip.com
      ii) **How do you want to host your app for SharePoint?** SharePoint-hosted
      iii) Click **Finish**
4. Open the **Default.aspx** file which defines the app's user interface and replace the HTML content inside the Content control with the ContentPlaceHolderID of **PlaceholderMain** with the following HTML content.

```
<asp:Content ContentPlaceHolderID="PlaceHolderMain" runat="server">

  <div id="toolbar">
    <input type="button" id="cmdGetSiteInfo" value="Get Site Info" />
  </div>

  <div id="content_box" style="margin-top: 12px; font-size: 14pt;" />

</asp:Content>
```

5. Inside the **AppPrmissionsLab** project open **AppManifest.xml** by double-clicking on it.
   a) Change the Title: **App Permissions Lab**
   b) Save and close **AppManifest.xml**
6. Inside Default.aspx, update the page title by update the text inside the Content control with the ContentPlaceHolderID of **PlaceHolderPageTitleInTitleArea** with the following text.

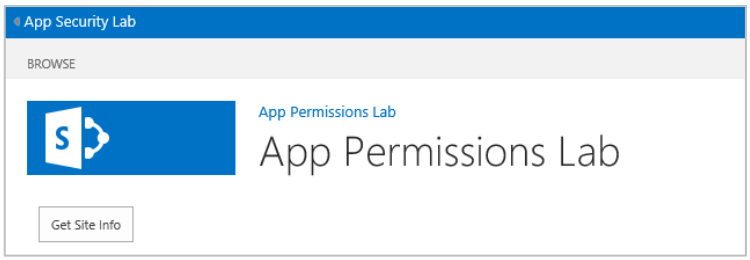
```
<asp:Content ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server">
  App Permissions Lab
</asp:Content>
```

   a) When you are done. Save your changes and close Default.aspx.
7. Open the **App.js** file located in the **Scripts** folder, which defines your app's behavior. Delete all the code from App.js and replace it with the following JavaScript code.

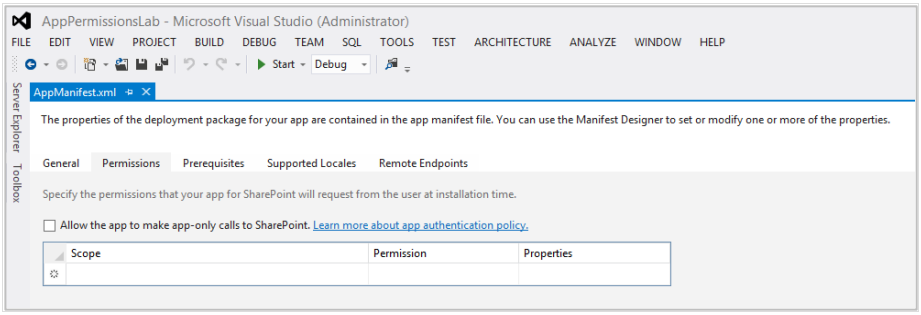
```
// you can copy-and-pasted this code from C:/Student/Modules/AppSecurity/Lab/StarterFiles/App.js.txt
$(onPageReady);

function onPageReady() {
  $("#cmdGetSiteInfo").click(onGetSiteInfo);
}

function onGetSiteInfo() {

  // create URL to make REST call into host web
  var hostWebUrl = getQueryStringParameter("SPHostUrl");

  var requestUri = "../_api/SP.AppContextSite(@target)/web/" +
                   "?$select=Id,Title,Url" +
                   "&@target='" + hostWebUrl + "'";


  // issue web service request against host web
  $.ajax({
    url: requestUri,
    type: "GET",
    headers: { "ACCEPT": "application/json;odata=verbose" }
  }).then(onComplete, onError);

}
```

```
function onComplete(data) {
  $("#content_box").text("Host Web Title: " + data.d.Title);
};

function onError(error) {
  var errorNumber = error.status;
  var errorMessage = errorNumber + " - " + error.statusText;
  $("#content_box").text("Error: " + errorMessage);
}

function getQueryStringParameter(paramName) {
  var querystring = document.URL.split("?")[1];
  if (querystring) {
    var params = querystring.split("&");
    for (var index = 0; (index < params.length) ; index++) {
      var current = params[index].split("=");
      if (paramName.toUpperCase() === current[0].toUpperCase()) {
        return decodeURIComponent(current[1]);
      }
    }
  }
}
```

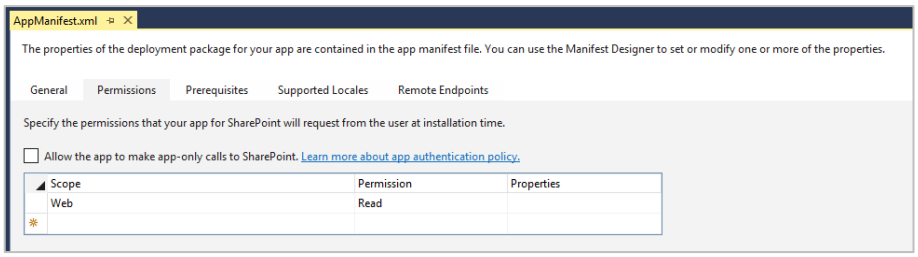
8. Make a few observations about the JavaScript code you just added to App.js.
   a) It uses jQuery to wire up an event handler to the click event of the **cmdGetSiteInfo** command button.
   b) It uses the jQuery **$.ajax** function to make a call into the new SharePoint REST API.
   c) It uses a SharePoint REST URI to retrieve the site properties including the ID, Title and Url.
   d) It uses the best practice of targeting the host web with an URL containing **SP.AppContextSite**.
   e) The app does not possess default permissions to access the host web.
9. When you are done, save your changes and close App.js.
10. Build and test your application by pressing **[F5]** or using the Visual Studio main menu: **Debug → Start Debugging**.
11. Once the app has been deployed, Internet Explorer will launch and navigate to the app's start page in the app web.



12. Click the **Get Site Info** button.
    a) At this point, the JavaScript code behind the button should fail while attempting to make a REST API call due to a lack of permissions.
    b) Notice how the SharePoint API prompts the user for new credentials for 3 attempts and then fails the attempt.
13. Stop the debugging session by closing the browser.
14. Return to Visual Studio and the **AppPermisionsLab** project.
15. Open the App Manifest in the visual designer and navigate to the **Permissions** tab.
    (Double Click on **AppManifest.xml**)



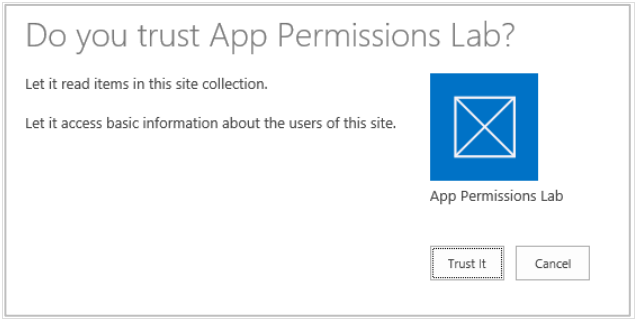
16. Add a permission request scoped at current **Web** for **Read** permissions.



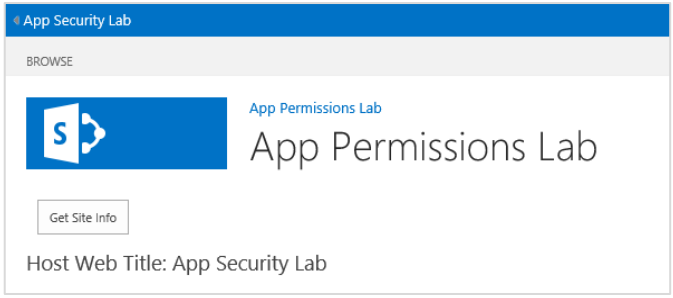
Note: When you made this change in the visual designer, it automatically added the following XML into the appmanifest.xml:

```
<AppPermissionRequests>
  <AppPermissionRequest Scope="http://sharepoint/content/sitecollection/web" Right="Read" />
</AppPermissionRequests>
```

17. Save your changes and close **AppManifest.xml**.
18. Now it is time to test the app a second time. Build and test your application by pressing **[F5]** or **Debug → Start Debugging**.
19. This time the app installation involves user interaction because the app is making a permission request. You will be promoted to trust the App Permissions Lab as shown below:



20. Click the **Trust It** button to approve the app's permission request.
21. When the app's start page appears, click the **Get Site Info** button. This time the app should succeed in making the REST API call because it was granted the required permissions.



22. Close the browser windows and close the **AppPermissionsLab** project in Visual Studio.

In this exercise, you developed and tested an app that used security. You were required to add a permission request in order for the app to accomplish a specific task which required permissions beyond what permissions an app has by default.

Exercise 3: Create a Test Certificate for Working with S2S Trusts

In this exercise you will create a new self-signed certificate for a SharePoint app that will utilize the S2S authentication model. You will build the app in the next exercise, this exercise is about getting everything prepared. (**Note**: if you have already completed this exercise in a previous lab you may skip ahead to Exercise 4).

1. Look inside the folder for this lab and locate the folder **Script** at the following path.

```
C:\Student\Modules\06_AddinSecurity\Lab\Scripts
```

   a) Inside this folder you should be able to locate these two PowerShell scripts.

      i) CreateTestCertificateForS2STrust.ps1

      ii) CreateTrustedSecurityTokenIssuer.ps1

2. Open the PowerShell scripted named **CreateTestCertificateForS2STrust.ps1** in the PowerShell ISE.

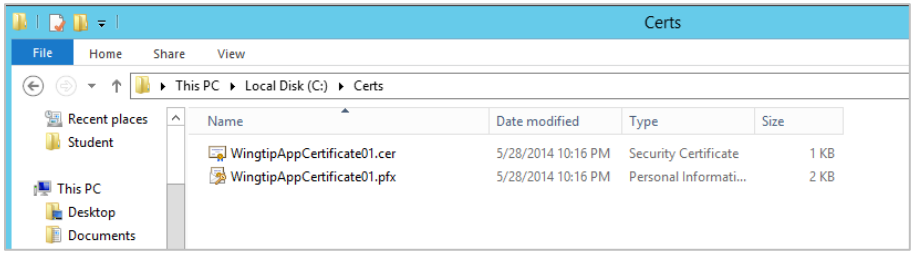
   a) Take a moment to review what this script does.

```
# create variable to call to makecert.exe command-line utility
$makecert =  $PSScriptRoot + "\makecert.exe"

$certname = "WingtipAppCertificate01"
$password = ConvertTo-SecureString "Password1" -AsPlainText -Force
$startdate = (Get-Date).ToString("MM/dd/yyyy")
$enddate =  ((Get-Date).AddYears(2)).ToString("MM/dd/yyyy")

# delete any pre-existing certifcates with same name
Get-ChildItem Cert:\CurrentUser\My | ? {$_.Subject -eq "CN=$certname"} | Remove-Item

Write-Host "Creating new x509 Certificate with subject name of $certname"
$silentResult = & $MakeCert -r -pe -n "CN=$certname" -b $startdate -e $enddate -ss my -eku -sy 12

$cert = Get-ChildItem Cert:\CurrentUser\My | ? {$_.Subject -eq "CN=$certname"}

# create local directory to export SSL certificate files
$outputDirectory = "c:\Certs\"
New-Item $outputDirectory -ItemType Directory -Force -Confirm:$false | Out-Null

$publicCertificatePath  =  $outputDirectory + $certname + ".cer"
Write-Host "Exporting public key to $publicCertificatePath"
$out = Export-Certificate -Type CERT -FilePath $publicCertificatePath -Cert $cert -Force

$privateCertificatePath = $outputDirectory + $certname + ".pfx"
Write-Host "Exporting password-protected private key to $privateCertificatePath "
$out = Export-PfxCertificate -FilePath $privateCertificatePath -Cert $cert -Password $password -Force
```

   b) Execute the script using the Windows PowerShell ISE.

      i) If you are prompted to change the Execution Policy when you run the script, click **Y** and press **Enter**.

      ii) After you execute this script, you should be able to see that it created two certificate files in the **C:\Certs** folder.



3. The next step is to create a trusted security token issuer for the certificate you have just created. Open the script named **CreateTrustedSecurityTokenIssuer.ps1** in the Windows PowerShell ISE and take a moment to review its content.

```
Add-PSSnapin "Microsoft.SharePoint.PowerShell"

$issuerID = "11111111-1111-1111-1111-111111111111"

# remove any pre-esisting with the same issuer ID
Get-SPTrustedSecurityTokenIssuer `
    | where {$_.Name -eq $issuerID } `
    | Remove-SPTrustedSecurityTokenIssuer -Confirm:$false

# get GUID for current SharePoint tenancy
$targetSiteUrl = "http://wingtipserver"
$targetSite = Get-SPSite $targetSiteUrl
$realm = Get-SPAuthenticationRealm -ServiceContext $targetSite

# parse together RegisteredIssuerName value
$registeredIssuerName = $issuerID + '@' + $realm

$publicCertificatePath = "C:\Certs\WingtipAppCertificate01.cer"
$publicCertificate = Get-PfxCertificate $publicCertificatePath

Write-Host
Write-Host "Using .cer file to register certificate as root authority with local SharePoint farm"
# this is new requirement for SharePoint 2016 - not required in sharePoint 2013
$silentResult = New-SPTrustedRootAuthority -Name "WingtipAppCertificate01" -Certificate
$publicCertificate

Write-Host
Write-Host "Using .cer file to register trusted token issuer in local SharePoint farm"
$secureTokenIssuer = New-SPTrustedSecurityTokenIssuer `
                     -Name $issuerID `
                     -RegisteredIssuerName $registeredIssuerName `
                     -Certificate $publicCertificate `
                     -IsTrustBroker

$secureTokenIssuer | Format-List Id, Name, RegisteredIssuerName
$secureTokenIssuer | Format-List Id, Name, RegisteredIssuerName, SigningCertificate `
                   |  Out-File -FilePath "SecureTokenIssuer.txt"


# configure SharePoint to suppport S2S Trusts with HTTP in addition to HTTPS
$serviceConfig = Get-SPSecurityTokenServiceConfig
$serviceConfig.AllowOAuthOverHttp = $true
$serviceConfig.Update()

Write-Host "All done..."
```

4. Execute **CreateTrustedSecurityTokenIssuer.ps1** using the Windows PowerShell ISE.
   a) When the script runs it create a text file named **SecureTokenIssuer.txt**.
   b) Open this file and observe that it contains information about the trusted security token issuer that you registered.
5. Close all the files you have opened in this exercise.

In this exercise you created a test certificate for configuring S2S trusts and registered a trusted security token issuer based on that test certificate. Now you can create a provider-hosted app that uses this trusted security token issuer as long as the provider-hosted app has access to the password-protected PFX file with the private key.

Exercise 4: Create a S2S High Trust SharePoint Provider-Hosted App

In this exercise you will create a new SharePoint Provider-Hosted app that will leverage the certificate and trusted security token issuer you created in the previous exercise to implement a high-trust app that utilizes S2S authentication.

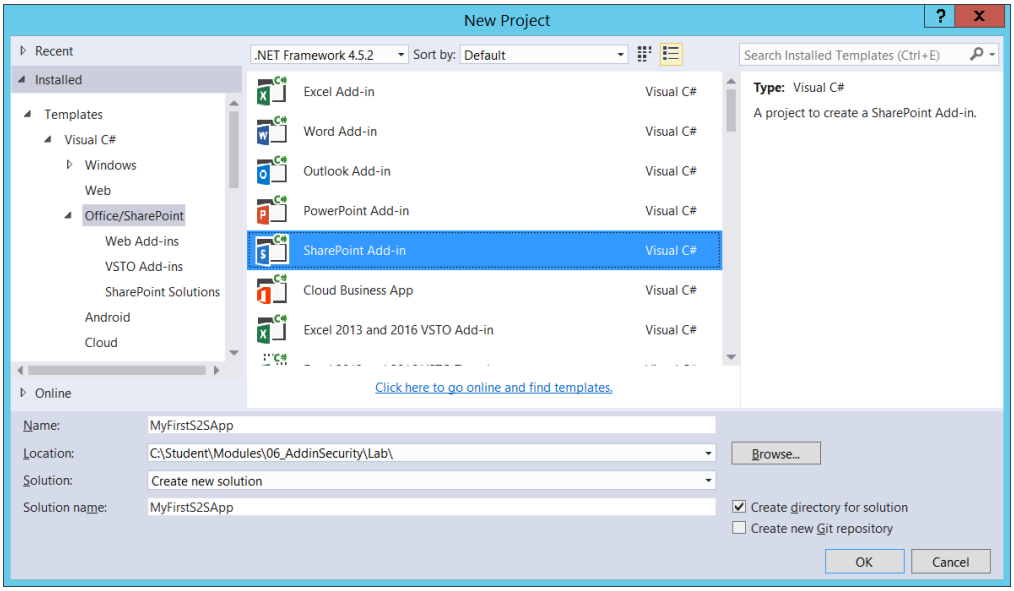
1. Launch **Visual Studio 2015** as administrator (if it is not already open).
   a) Press the Windows key to display the Windows Start page
   b) Right click on the **Visual Studio 2015** tile and select **Run as administrator**
2. Create a new project in Visual Studio 2015 for hosting your app:
   a) Select **File → New → Project**.
   b) In the **New Project** dialog:

i) Find the **SharePoint Add-in** template under the **Templates → Visual C# → Office / SharePoint** section.

ii) **Name**: MyFirstS2SApp

iii) **Location**: C:\Student\Modules\AppSecurity\Lab\

iv) When the **New Project** dialog looks like the screenshot below, click the **OK** button.

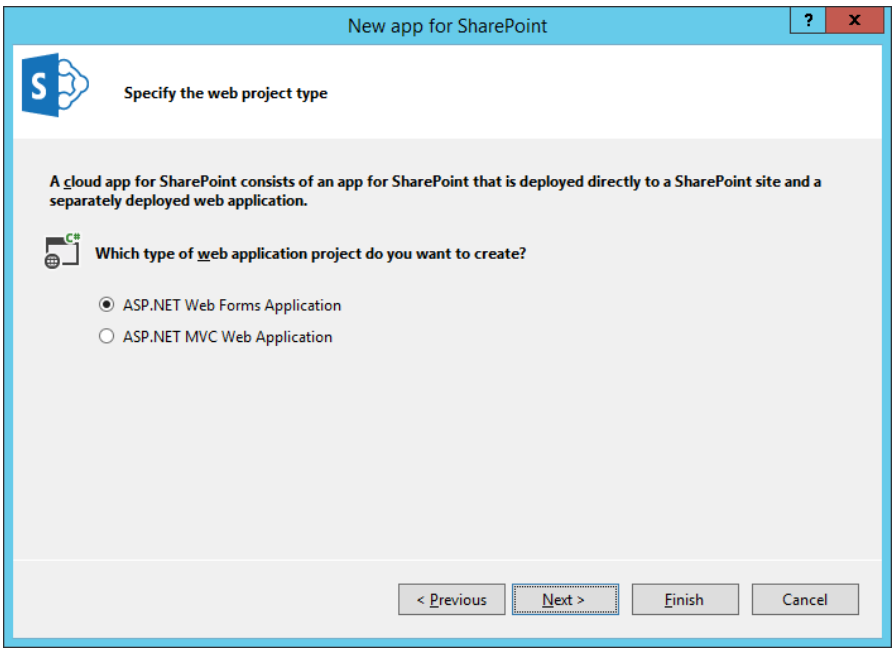


c) In the **New SharePoint Add-in** wizard, use the following values to complete the wizard and click **Next**.

i) **What site do you want to use for debugging?** http://security.wingtip.com

ii) **How do you want to host your app for SharePoint?** Provider-hosted.

d) On the **Specify the web project type** screen select the **ASP.NET Web Forms Application** choice and click the **Next** button to move to the next screen. **Do not** click the Finish button.



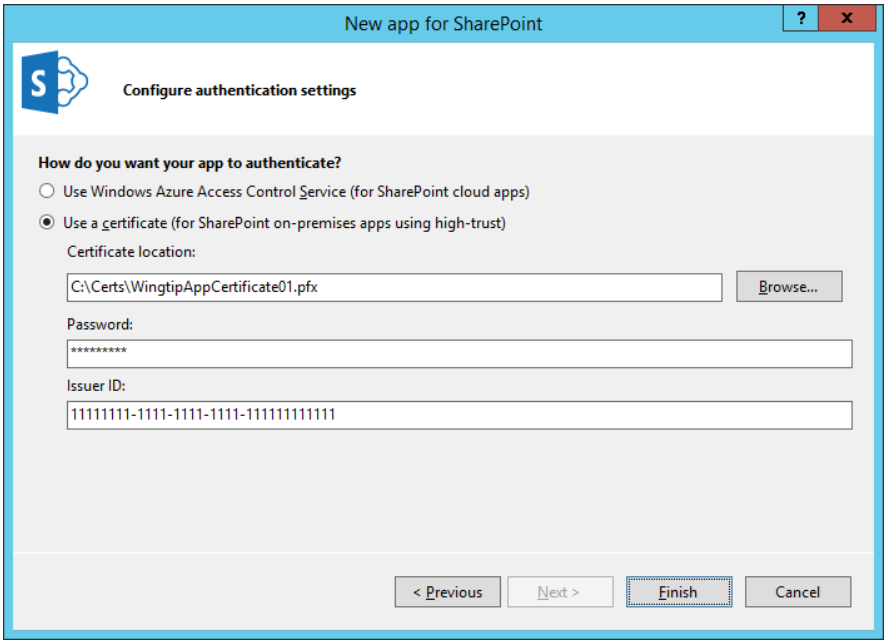
e) You should now be at the Configure authentication settings page. Fill this page out as follows.

i) Select **Use a certificate (for SharePoint on-premise apps using high-trust)**

ii) **Certificate location**: C:\Certs\WingtipAppCertificate01.pfx

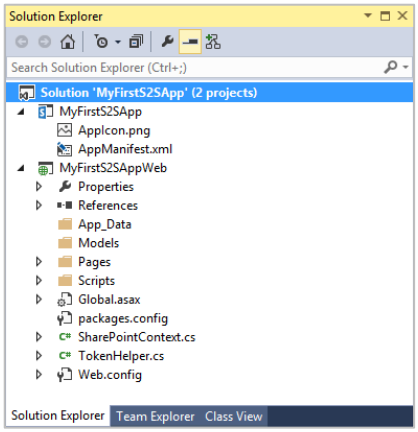
iii) **Password**: Password1

iv) **Issuer ID**: 11111111-1111-1111-1111-111111111111



f) Click **Finish** to complete the wizard and create the new provider-hosted app project.

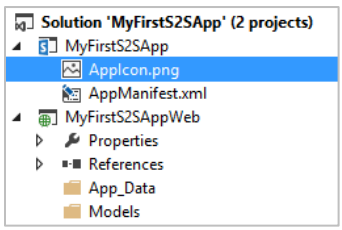
g) Once the Visual Studio solution has been created, you should see it contains two projects named **MyFirstS2SApp** and **MyFirstS2SAppWeb**.



Make some modification to the App project named MyFirstS2SApp

3. Replace the **AppIcon.png** file with a custom image.

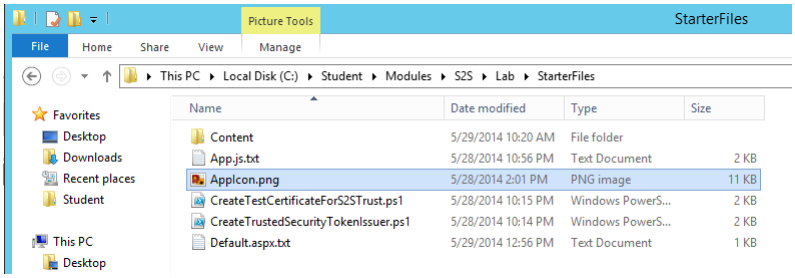
a) In the Solution Explorer, look inside the **MyFirstS2SApp** project and locate the image file **AppIcon.png**.



b) Using Windows Explorer, look inside the **StarterFiles** folder for this lab at the following path.

```
C:\Student\Modules\06_AddinSecurity\Lab\StarterFiles
```

c) Locate the custom image file named **AppIcon.png**.

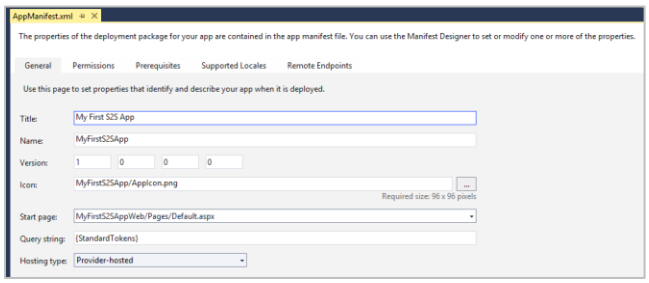


d) Use the **AppIcon.png file** in the **StarterFiles** folder to replace the **AppIcon.png file** in the **MyFirstS2SApp** project.

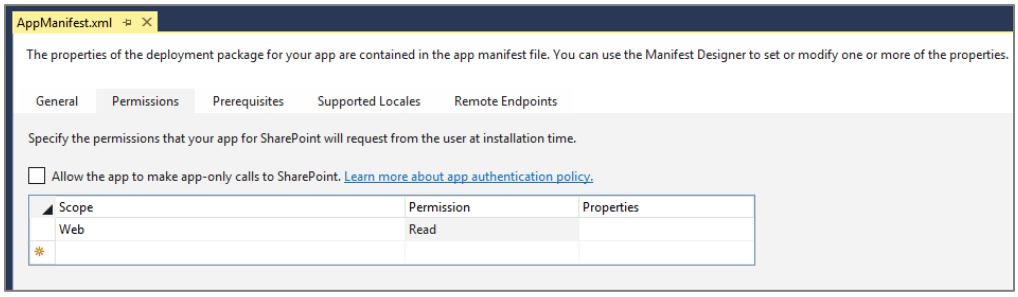
4. Modify the **AppManifest.xml** file.

a) In Solution Explorer, double-click on **AppManifest.xml** open it in Visual Studio's App Manifest Designer.

b) Update the Title property from **MyFirstS2SApp** to **My First S2S App**.



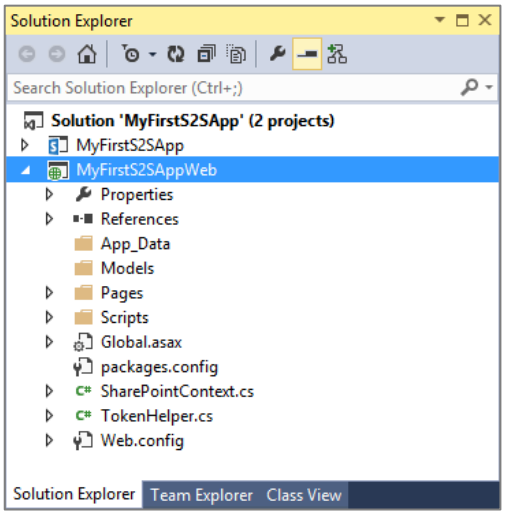
c) Click on the **Permissions** tab and then add a permission with a **Scope** of **Web** and a **Permission** level of **Read**.



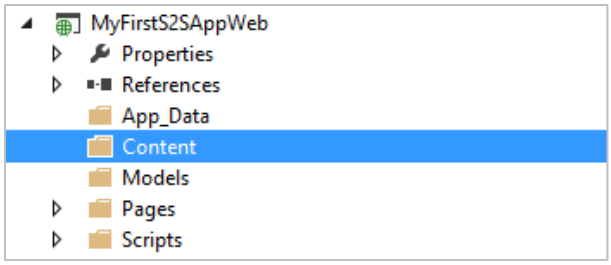
d) Save your changes and close **AppManifest.xml**.

Implement the Web Project named MyFirstS2SAppWeb

5. In Solution Explorer, move down to the Web Project named **MyFirstS2SAppWeb**.



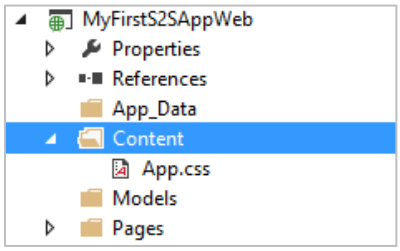
6. Add a CSS file and an image file for the start page.
   a) Inside the **MyFirstS2SAppWeb** project, add a new top-level folder named **Content**.



   b) Using Windows Explorer, locate the CCS file in the **StarterFiles** folder at the following path.

```
C:\Student\Modules\AppSecurity\Lab\StarterFiles\Content\App.css
```

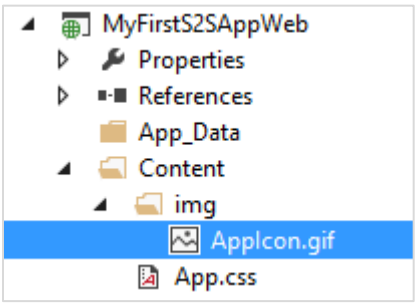
   c) Add the **App.css** file from the **StarterFiles** folder into the new **Content** folder in the **MyFirstS2sApp** project.



   d) Create a child folder named **img** inside the **Content** folder.
   e) Using Windows Explorer, locate the CCS file in the **StarterFiles** folder at the following path.

```
C:\Student\Modules\06_AddinSecurity\Lab\StarterFiles\Content\img\AppIcon.gif
```

   f) Add the **AppIcon.gif** file from the **StarterFiles** folder into the new **img** folder in the **MyFirstS2sApp** project.



g) Open the CSS file named **App.css** and have a quick look at the set of CSS rules inside.

h) When you are done, close the **App.css** file without saving any changes.

7. Modify the app's start page named **Default.aspx**.

a) Using the Solution Explorer, look inside the **Pages** folder of the **MyFirstS2SAppWeb** project and locate **Default.aspx**.

b) Double click on **Default.aspx** to open it in the Web Forms Editor of Visual Studio.

c) Delete all contents from **Default.aspx** except for the first line with the **<@Page>** directive.

d) Using Windows Explorer, locate the file named **Default.aspx.txt** in the **StarterFiles** folder at the following path.

```
C:\Student\Modules\06_AddinSecurity\Lab\StarterFiles\Default.aspx.txt
```

e) Open the file named **Default.aspx.txt** in **Notepad.exe** and copy its entire contents to the Windows clipboard.

f) Return to Visual Studio.

g) Navigate to **Default.aspx** which should still be open in the Web Forms Editor.

h) Position you cursor in **Default.aspx** right below the first line with the **<@Page>** directive

i) Paste the content of the Windows clipboard into **Default.aspx**.

j) Save your changes to **Default.aspx**.

8. Examine the layout of the HTML you just pasted into **Default.aspx**.

a) Make sure you are looking at **Default.aspx** in Code View.

b) Note in the **head** section that that there is already a link to the CSS file named **App.css**.

c) The **body** section contains a **div** with an **id** of **page_width** which contains child elements that make up the user interface.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" ... >

<!DOCTYPE html>

<html>

<head runat="server">
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>My First S2S App</title>
  <link href="../Content/App.css" rel="stylesheet" />
</head>

<body>
  <form id="form1" runat="server">

    <div id="page_width">
      <!-- more inside -->
```

```
    </div>

  </form>
</body>
</html>
```

9. Examine the server-side controls inside the **div** with the **id** of **page_width**.
   a) First, you should see there is an ASP.NET **HyperLink** control with an **ID** of **HostWebLink**.

```
<div id="nav_bar">
  <asp:HyperLink ID="HostWebLink" runat="server" />
</div>
```

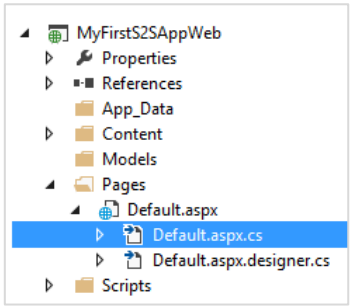
   b) Next, you should see two ASP.NET **Button** controls with **ID** values of **cmdGetTitleCSOM** and **cmdGetTitleREST**.

```
<nav id="toolbar">
  <asp:Button ID="cmdGetTitleCSOM" runat="server" Text="Get Title using CSOM" />
  <asp:Button ID="cmdGetTitleREST" runat="server" Text="Get Title using REST" />
</nav>
```

   c) Finally, there is an ASP.NET **Literal** control with an **ID** of **placeholderMainContent**.

```
<div id="content_box">
  <asp:Literal ID="placeholderMainContent" runat="server" ></asp:Literal>
</div>
```

10. Modify the server-side C# code behind the start page.
    a) In Solution Explorer, locate the code-behind file for **Default.aspx** named **Default.aspx.cs**.



    b) Open **Default.aspx.cs** and inspect the code inside that was generate by Visual Studio. You should be able to see that there is a code-behind class named **Default** for the page named **Default.aspx**. This Default code-behind class contains two methods in the **Default** code-behind class named **Page_PreInit** and **Page_Load**.

```
public partial class Default : System.Web.UI.Page {

  protected void Page_PreInit(object sender, EventArgs e) {
    // implementation details
  }

  protected void Page_Load(object sender, EventArgs e) {
    // implementation details
  }
}
```

    c) Delete the entire method named **Page_PreInit**.
    d) Leave the **Page_Load** method but delete all to code inside it so that it is an empty method.

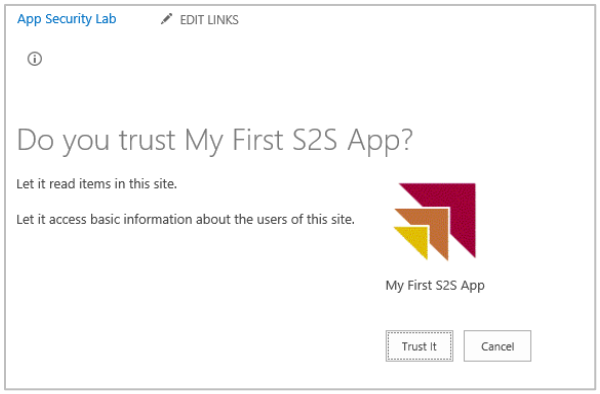
```
public partial class Default : System.Web.UI.Page {

  protected void Page_Load(object sender, EventArgs e) {

  }
}
```

    e) Implement the **Page_Load** method using the following code.

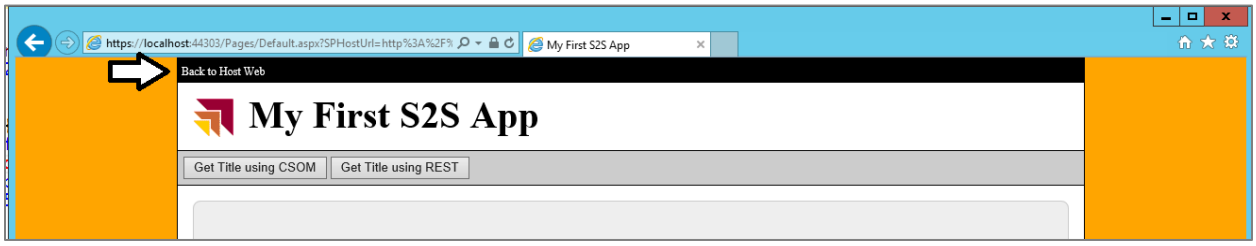
```
protected void Page_Load(object sender, EventArgs e) {
```

```
  SharePointContext spContext =
    SharePointContextProvider.Current.GetSharePointContext(Context);

  this.HostWebLink.NavigateUrl = spContext.SPHostUrl.AbsoluteUri;
  this.HostWebLink.Text = "Back to Host Web";

}
```

11. Test out your work by running the app in the Visual Studio Debugger.
    a) Press the **{F5}** key in Visual Studio to begin a debugging session for the app project.
    b) As Visual Studio installs the app in the test site, you will be prompted whether you trust the app. Click **Trust It**.

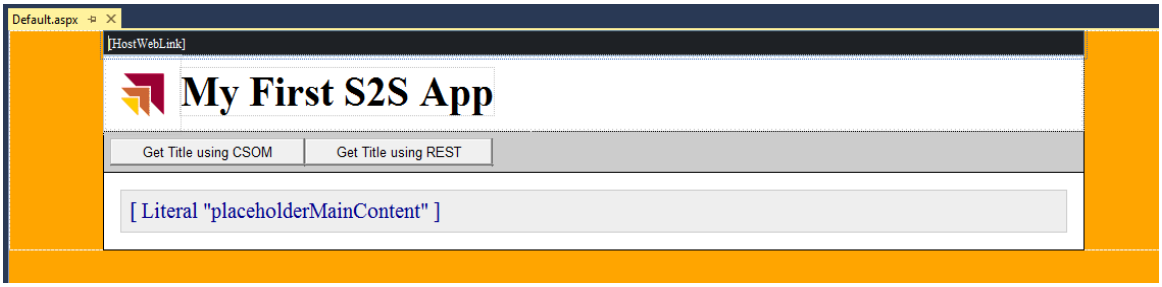


    c) Here we are



    d) Test it out. Make sure the link works.
    e) Close the browser and quit the debugging session.

12. Add event handlers behind the buttons
    a) Return to Visual Studio and the Web Project named **MyFirstS2SAppWeb**.
    b) Open the page named Default.aspx in the Web Forms Editor.
    c) Switch Default.aspx into Design View.



    d) When in Design View, double-click on the **Get Title with CSOM** button to generate an event handler in **Default.aspx.cs**.
    e) Return to Design View in **Default.aspx** and double-click on the **Get Title with REST** button to generate another event handler.
    f) When you're done, there should be two new event handlers named **cmdGetTitleCSOM_Click** and **cmdGetTitleREST_Click**.

```
public partial class Default : System.Web.UI.Page {
```

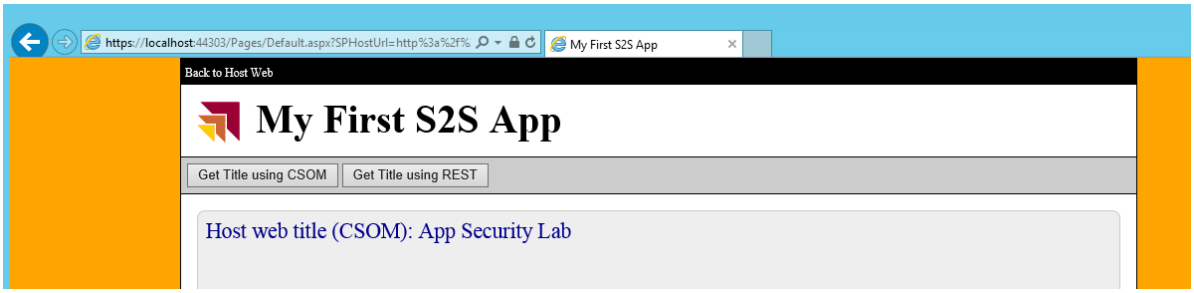
```
  protected void Page_Load(object sender, EventArgs e) {
    // ...
  }

  protected void cmdGetTitleCSOM_Click(object sender, EventArgs e) {

  }

  protected void cmdGetTitleREST_Click(object sender, EventArgs e) {

  }
}
```

Make a server-side CSOM Call using C# and S2S Authentication

13. Add the following code to **cmdGetTitleCSOM_Click** to execute a CSOM call using S2S authentication.

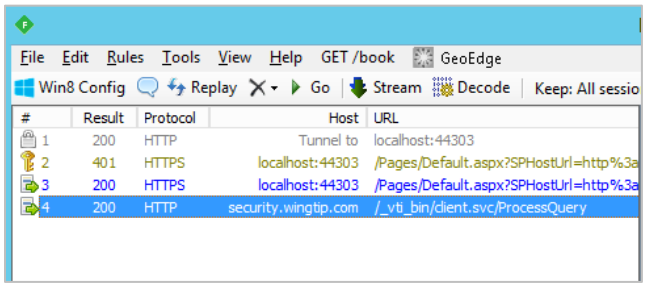
```
protected void cmdGetTitleCSOM_Click(object sender, EventArgs e) {

  SharePointContext spContext =
    SharePointContextProvider.Current.GetSharePointContext(Context);

  using (var clientContext = spContext.CreateUserClientContextForSPHost()) {
    clientContext.Load(clientContext.Web);
    clientContext.ExecuteQuery();
    placeholderMainContent.Text = "Host web title (CSOM): " + clientContext.Web.Title;
  }

}
```

14. Test your work to ensure the CSOM call executes successful without any errors.
    a) Press the **{F5}** key to start another debugging session.
    b) If you are prompted to trust the app, click the **Trust It** button to complete the app installation.
    c) After the app has been installed, the Visual Studio debugger should redirect you to the app's start page in the remote web.
    d) Click the **Get Title using CSOM** button. When you click this button, the page should display a message with the title of the host web as shown in the following screenshot.

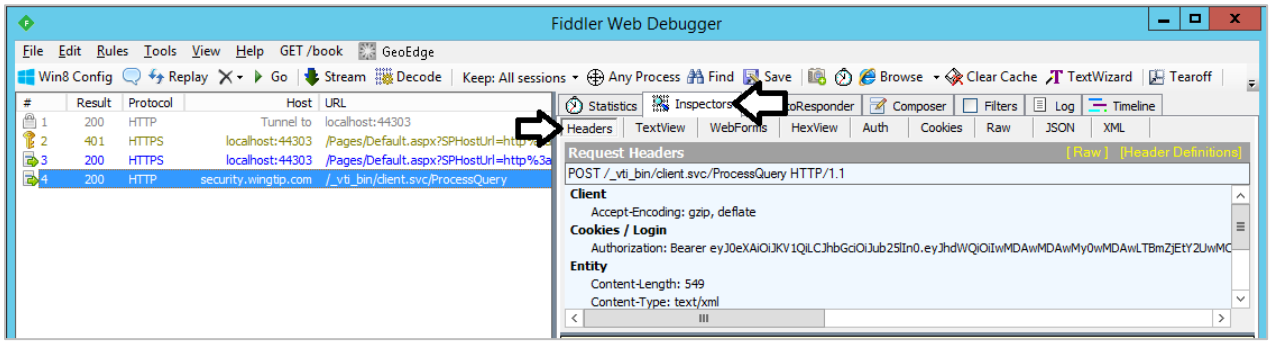


    e) Keep the browser window open because you will use it in the next step.

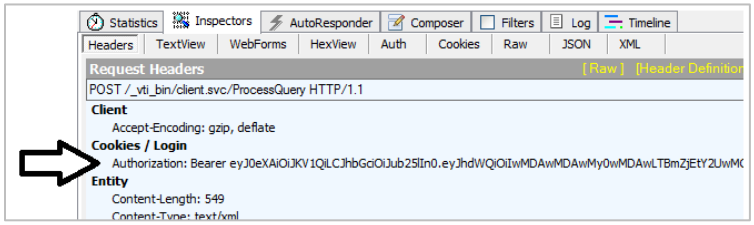
15. Inspect an S2S CSOM call using the Fiddler utility.
    a) Launch Fiddler (Press the **Windows** Key then Type **Fiddler** and click on the **Fiddler2** tile)
    b) Return to the app's Start page in the browser.
    c) Click the **Get Title using CSOM** button to execute a new CSOM.
    d) Return to Fiddler and inspect the requests.
    e) Select the CSOM request with the URL of **/_vti_bin/client.svc/ProcessQuery**.



f) On the right-hand side of the Fiddler window, click the **Inspectors** tab and then the **Headers** tab in the row directly below.



g) You should be able to verify that there is a request header named **Authorization**. The value of the **Authorization** header is the string value of *"Bearer "* parsed together with the S2S access token.



h) When you have finished examining the call in Fiddler, close Internet Explorer and stop your Visual Studio debugging session.

Make a server-side REST Call using C# and S2S Authentication

16. Return to Visual Studio and the Web Project named **MyFirstS2SAppWeb**.
17. Open the C# file named of **default.aspx.cs** if it is not already open.
18. Add the following using statements at the top of **default.aspx.cs**.

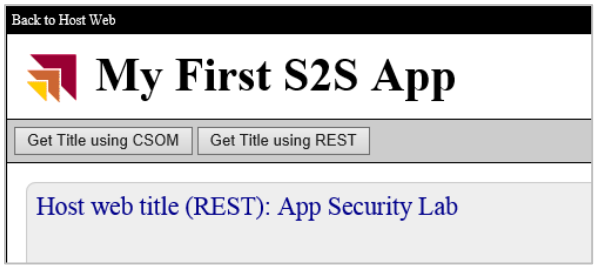
```
using System.Net;
using System.Xml.Linq;
```

19. Add the following code to **cmdGetTitleREST_Click** to execute a REST call using S2S authentication.

```
protected void cmdGetTitleREST_Click(object sender, EventArgs e) {

  SharePointContext spContext =
    SharePointContextProvider.Current.GetSharePointContext(Context);

  string restUri = spContext.SPHostUrl + "_api/web/title";

  HttpWebRequest request = WebRequest.Create(restUri) as HttpWebRequest;
  request.Accept = "application/atom+xml";

  string spAccessToken = spContext.UserAccessTokenForSPHost;
  request.Headers["Authorization"] = "Bearer " + spAccessToken;

  HttpWebResponse response = request.GetResponse() as HttpWebResponse;
```

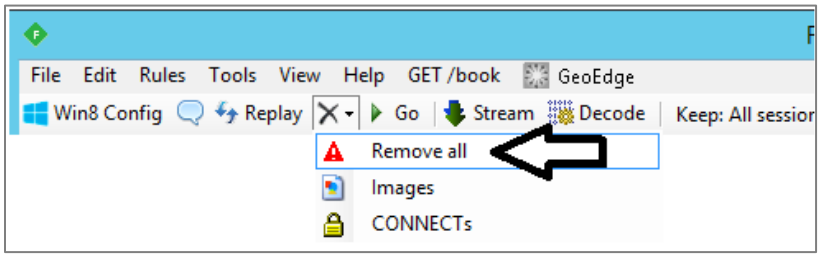
```
  XDocument responseBody = XDocument.Load(response.GetResponseStream());

  XNamespace nsDataService = "http://schemas.microsoft.com/ado/2007/08/dataservices";
  string hostWebTitle = responseBody.Descendants(nsDataService + "Title").First().Value;

  placeholderMainContent.Text = "Host web title (REST): " + hostWebTitle;

}
```

20. Test your work to ensure the REST call executes successful without any errors.
    a) Press the **{F5}** key to start another debugging session.
    b) If you are prompted to trust the app, click the **Trust It** button to complete the app installation.
    c) After the app has been installed, the Visual Studio debugger should redirect you to the app's start page in the remote web.
    d) Click the **Get Title using REST** button. When you click this button, the page should display a message with the title of the host web as shown in the following screenshot.

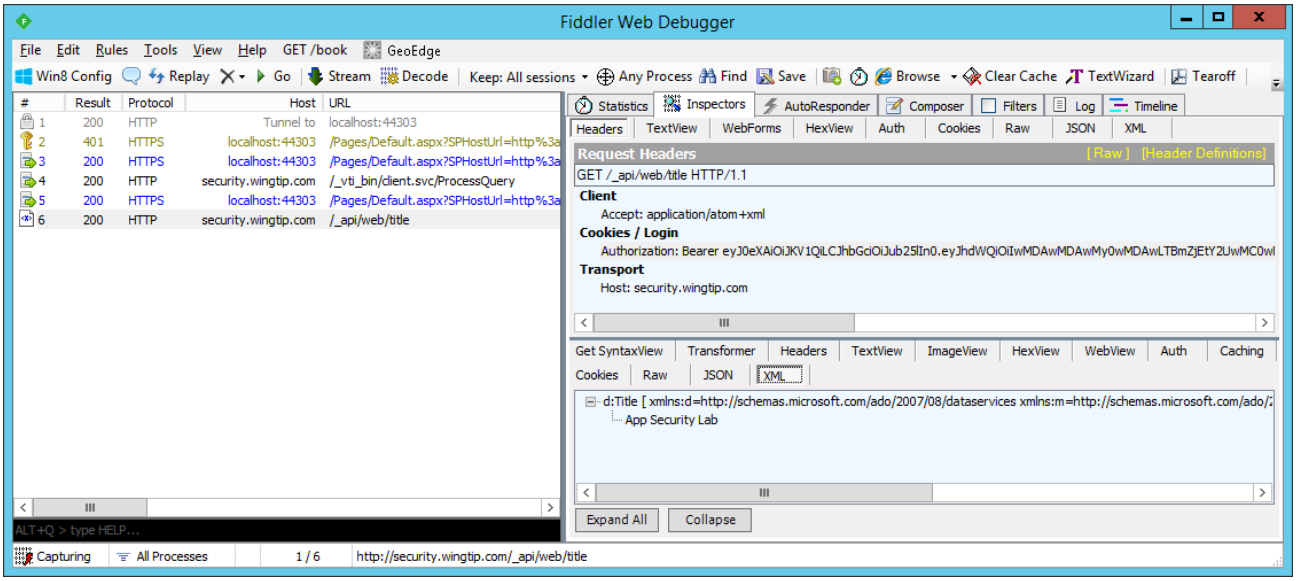


    e) Keep the browser window open because you will use it in the next step.

21. Inspect the REST call using the Fiddler utility.
    a) Return to Fiddler (or launch Fiddler if it's not already running).
    b) On the Fiddler toolbar, drop down the menu with X on it and select the Remove all command. This will clear the Fiddler window of all existing requests.



    c) Return to the app's Start page in the browser.
    d) Click the **Get Title using REST** button to execute a new REST call.
    e) Return to Fiddler and inspect the requests.
    f) Select the REST request with the URL of **/_api/web/title**.
    g) Verify that the request carries the Authorization header and a value just like the CSOM call did.



22. You are now done with the lab. You can close Internet Explorer and stop the debugging session.

In this exercise you created a new SharePoint Provider-Hosted App that uses S2S security. Using S2S security is both easy and fun!