Contents

Contents ........................................................................................................................................... ii
Introduction ..................................................................................................................................... 3
   Purpose ...................................................................................................................................... 3
   Prerequisites ............................................................................................................................. 3
   Lab Overview ............................................................................................................................. 3
   Getting Help .............................................................................................................................. 3
   Note ........................................................................................................................................... 3
Exercises ........................................................................................................................................ 4
   A. Securing Web Services ........................................................................................................ 4
      1. Getting Started ................................................................................................................ 4
      2. Creating User Accounts and Groups ............................................................................ 5
      3. Enabling WSE 2.0 ......................................................................................................... 7
      4. Sending Security Tokens ............................................................................................... 8
      5. Requiring Security Tokens ........................................................................................... 9
      6. Authorization .................................................................................................................. 11
      7. Implementing a UsernameTokenManager ................................................................... 11
   B. Web Services Policy ............................................................................................................ 12
      1. Getting Started ................................................................................................................ 13
      2. Writing a Policy ............................................................................................................... 13
      3. Configuring a Policy ....................................................................................................... 14
Summary ....................................................................................................................................... 15
Introduction

Purpose
This lab will help you gain an understanding of WSE 2.0's basic security features including the new support for declarative policy assertions.

Prerequisites
This lab was designed for developers already familiar with Microsoft .NET, XML and Web services technologies. Specifically, you should have some experience working with:

- C# and Visual Studio.NET
- XML 1.0, Namespaces, and SOAP Messages
- System.Xml APIs (e.g., DOM, XPath, etc.)
- HTTP Pipeline (e.g., IHttpHandler, web.config, etc.)

Lab Overview
The lab is organized into several directories, each of which contains information to assist you in completing the exercises.

HOL-WEB02

| Lab Module
| Manual
| Exercises

The Manual directory provides detailed procedures that guide you through the process of completing the exercises while the Exercises directory contains the "before" and "after" Visual Studio.NET projects.

Getting Help
If you get stuck on a particular exercise you can a) find a lab proctor to assist you (proctors are available during lab hours), b) self-diagnose the issue by peaking at the solution project in the Exercises\After directory.

Note
The working directory for this lab is C:\Labs\HOL-WEB02. Use this as the reference point for all instructions on this lab.

Exercises

Walk through each exercise following the detailed step-by-step instructions. You'll be working in the Exercises\Before directory.

A. Securing Web Services

WS-Security defines a message-oriented framework for securing Web services. A message-oriented framework makes it possible to secure Web services in a transport-neutral way. Although Web services can be secured using SSL, such a solution is tied to HTTP for practical purposes.

Many Web services desire message-level access control, integrity, and privacy. WS-Security facilitates achieving these goals by defining mechanisms for authenticating, signing, and encrypting SOAP messages respectively. Once a message has been authenticated, it's also possible to authorize access to specific operations based on the message's verified credentials.

WSE 2.0 provides an implementation of WS-Security in the Microsoft.Web.Services.Security namespace. The WSE 2.0 implementation makes it possible to authenticate, sign, and encrypt the SOAP messages used in your WebMethods. It also makes it possible to authorize access to specific functionality based on the incoming message's credentials.

Your first exercise is to secure a Web service. If you get stuck along the way, feel free to check out the solution found in Exercises\After.

1. Getting Started
   - Open the SecureInvoiceClient and SecureInvoiceService projects in Exercises\Before.
   - Familiarize yourself with the client and server code. These two projects constitute an invoice management system. Run SecureInvoiceClient and experiment with the different operations. Notice that any user (including no user) can perform any of the supported operations. If you run the Invoice Manager application and press View, you should see the following results:

   ![Invoice Manager Screen]

   - The SecureInvoiceService virtual directory is not performing any type of HTTP authentication when messages arrive. Open Internet Information Services to verify this (select Start | Control Panel | Administrative Tools | Internet Information Services). Right click on the SecureInvoiceService virtual directory and select Properties | Directory Security. Then press Edit in the Anonymous access and authentication control area.
You should see the following dialog:

- We're not going to use IIS's built-in authentication mechanisms because we're going to use WSE 2.0’s WS-Security implementation instead.
- Your goal is to add security features that control access to the various operations based on the credentials of the incoming message. Assume that the Invoice Manager application supports the following groups of users and corresponding rights:

<table>
<thead>
<tr>
<th>Group</th>
<th>Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>View invoices</td>
</tr>
<tr>
<td>Vendor</td>
<td>Submit invoices</td>
</tr>
<tr>
<td>Manager</td>
<td>Approve invoices</td>
</tr>
<tr>
<td>Accounting</td>
<td>Pay invoices</td>
</tr>
</tbody>
</table>

- Any user should be allowed to view invoices, but only vendors can submit invoices, only managers can approve invoices, and only accounting personnel can pay invoices. Your job in this exercise is to implement these features using the WSE 2.0 security APIs.

2. Creating User Accounts and Groups
You need to setup up some local user accounts and groups to use in this part of the lab. You're going to create one group for each of the user types described above (e.g., User, Vendor, Manager, and Accounting) along with some user accounts assigned to the different groups.

- Open your local Computer Management utility (select Start | Control Panel | Administrative Tools | Computer Management).
- Navigate to System Tools | Local Users and Groups | Users.

- Create four new user accounts named 'admin', 'vick', 'mike', and 'aaron'. You can use the same password for all of them to make things easier. Follow these steps for creating each account:
  a. Select Action | New User.
  b. Enter the user name (e.g., 'admin') and password (e.g., 'TechEd2003!')
  c. Deselect **User must change password at next logon**.
  d. Press Create. The dialog should look like this for the 'admin' account:

![New User Dialog](image)

- Navigate to **System Tools | Local Users and Groups | Groups**. You're going to create the following new groups with the corresponding members:

<table>
<thead>
<tr>
<th>Group</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>admin, vick, mike, aaron</td>
</tr>
<tr>
<td>Vendor</td>
<td>admin, vick</td>
</tr>
<tr>
<td>Manager</td>
<td>admin, mike</td>
</tr>
<tr>
<td>Accounting</td>
<td>admin, aaron</td>
</tr>
</tbody>
</table>

- Create four new groups named 'User', 'Vendor', 'Manager', and 'Accounting'. Follow these steps for creating each group:
  a. Select **Action | New Group**.
  b. Type in the group name (e.g., 'User').
  c. Press **Add** to select the group members. The following dialog will appear. Simply type the name of the user (e.g., 'admin') and press **OK**.
d. Repeat this for each member of the group. For the 'User' group, the New Group dialog should look like this:

e. Press Create.

- Now you're ready to start writing code that takes advantage of these user accounts and groups using WSE 2.0.

3. Enabling WSE 2.0

- We've already enabled WSE 2.0 in both projects. Here's a summary of what's been done:
  a. We added a reference to the Microsoft.Web.Services assembly in both projects.
  b. In SecureInvoiceService, we added Microsoft.Web.Service.WebServicesExtension to the soapExtensionTypes section of web.config. Doing this hooks the WSE 2.0 framework into ASP.NET's WebMethod processing pipeline. This is essentially what "turns WSE 2.0 on".

Your web.config should contain the following section (Note: This is already completed for you):

```xml
<configuration>
  <system.web>
    <webServices>
      <soapExtensionTypes>
        <add type="Microsoft.Web.Services.WebServicesExtension, Microsoft.Web.Services, Version=2.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" priority="1" group="0"/>
      </soapExtensionTypes>
    </webServices>
  </system.web>
</configuration>
```

c. In the SoapInvoiceClient project, we changed the base class of each Web Reference to WebServicesClientProtocol instead of SoapHttpClientProtocol. If you have the WSE 2.0 Settings Tool installed, it will automatically do this when creating a Web Reference.

4. Sending Security Tokens

Since the SecureInvoiceService project is already configured to use WSE 2.0, it's ready to process security tokens sent by the client application. Hence, the first step is to add code to SoapInvoiceClient to start sending security tokens to the service.

- We already have the code in place to retrieve a username and password from the user.
- Open login.cs and inspect the code behind the Login button. Notice that it creates a new Microsoft.Web.Services.Security.UsernameToken object with the supplied values and saves it in a public field called Token.

```csharp
public class LoginForm : System.Windows.Forms.Form
{
  public UsernameToken Token = null;
  private void button1_Click(object sender, System.EventArgs e)
  {
    this.Token = new UsernameToken(this.textBox1.Text, this.textBox2.Text, PasswordOption.SendPlainText);
    this.Close();
  }
}
```

- Inspect the code behind the Set User button (on the InvoiceManagerForm). It simply displays the form found in login.cs and displays the specified username on the form.

```csharp
private void btnLogin_Click(object sender, System.EventArgs e)
{
  login.ShowDialog();
  labelLogin.Text = String.Format("Username: {0}", login.Token.Username);
}
```

- Now you need to configure the proxy class with the UsernameToken created for the user. Add a new method to InvoiceManagerForm called ConfigureProxy that looks like this:

```csharp
private void ConfigureProxy(WebServicesClientProtocol proxy)
{
  proxy.RequestSoapContext.Security.Tokens.Add(login.Token);
}
```

- This method adds the UsernameToken to the SOAP request and signs the message with the token. The UsernameToken allows the service to perform authentication while the signature allows it to perform integrity checks.
- Locate the ViewInvoices method in InvoiceManagerForm and call ConfigureProxy before invoking the View method as illustrated here:

```csharp
private void ViewInvoices()
{
    ViewInvoicesWse viewProxy = new ViewInvoicesWse();
    ConfigureProxy(viewProxy);
    DataSet ds = viewProxy.View();
    ...
}
```

- Build and run SecureInvoiceClient.
- Press Set User. Specify the 'admin' account's credentials (that you created earlier) and press OK. Then, press View and verify that it works. WSE 2.0 automatically authenticates the supplied username token against the machine accounts.
- Try changing the user account to something else (e.g., username: bob, password: bob), press View again, and verify that it doesn't work. You should get an authentication error.
- Now select an invoice in the list and press Approve. Notice that it works even though we didn't send a username token to the service for this operation. This illustrates that the service doesn't require a username token even though it authenticates the token when supplied.

Note: in addition to sending UsernameTokens and signatures, it's also possible to encrypt portions of the message using similar techniques.

5. Requiring Security Tokens

You can require clients to supply a security token by adding some code to your WebMethods.

- In the SecureInvoiceService project, create a new class called WseSecurityHelpers that has a single method called GetUsernameToken that looks like this (Note: Create this class in the WseSecurityHelpers file. It already includes the correct using references. Also be sure to create a constructor for this class.):

```csharp
public class WseSecurityHelpers
{
    public WseSecurityHelpers(){}

    public static UsernameToken GetUsernameToken(SoapContext context)
    {
        ...
    }
}
```

- In GetUsernameToken, verify that context is not null and that it contains a UsernameToken object, which needs to be returned. Here's an example of how you can accomplish this:

```csharp
public class WseSecurityHelpers
{
```
public static UsernameToken GetUsernameToken(SoapContext context) {
    if (context == null)
        throw new Exception("Only SOAP requests are permitted.");

    // Make sure there's a token
    if (context.Security.Tokens.Count == 0)
    {
        throw new SoapException("
            Missing security token",
                SoapException.ClientFaultCode);
    }

    foreach (UsernameToken tok in context.Security.Tokens)
        return tok;

    throw new Exception("UsernameToken not supplied");
}

• Then, open ViewInvoices.asmx.cs and update the View WebMethod to call GetUsernameToken before doing anything else as shown here:

    [WebMethod]
    public DataSet View()
    {
        WseSecurityHelpers.GetUsernameToken(RequestSoapContext.Current);
        ... // remainder of method as before

    • Repeat this by adding the same call to the WebMethods found in SubmitInvoice.asmx.cs, ApproveInvoice.asmx.cs, and PayInvoice.asmx.cs.

    • Build SecureInvoiceService.

    • Run the client and try invoking all operations again. The only one that should work is View (assuming you supply a valid user account), since it's the only one sending a username token to the service.

    • Go back to the client project and update all of the button handlers to call ConfigureProxy before invoking the operation (like you did earlier for View).

    • You may also want to add the following code fragment to the beginning of each button handler to ensure the user has provided a user account before calling ConfigureProxy:

        ... private void btnPay_Click(object sender, System.EventArgs e)
        {
            try
            {
                if (login.Token == null)
                {
                    MessageBox.Show("You must specify user credentials
                        before invoking this operation", "Missing Credentials",
                        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                    return;
                }

                PayInvoiceWse proxy = new PayInvoiceWse();
                ConfigureProxy(proxy);
                ...

    • Build SecureInvoiceClient.

    • Run the client again and verify that you must provide a valid user account before you can invoke any operation.
**Note**: in addition to requiring **UsernameTokens**, you can also require signatures and encryption using similar techniques.

6. Authorization

At this point, **SecureInvoiceService** is performing (and requiring) message authentication but it isn't distinguishing between different users and what they're authorized to do. Authorizing messages based on the supplied token is made possible by the token's **Principal** property. WSE 2.0 populates **Principal** with the Windows account information mapped to the supplied token.

- Within each WebMethod add a call to **Principal.IsInRole** to verify that the authenticated user is in the appropriate group for the given operation (Refer to Page 5 for roles).
- The following code illustrates how to check for the **Accounting** group before executing **Pay**:

```csharp
[WebMethod]
public void Pay(string id)
{
    UsernameToken tok =
        WseSecurityHelpers.GetUsernameToken(RequestSoapContext.Current);
    if (!tok.Principal.IsInRole(string.Format("{0}\Accounting", 
        Dns.GetHostName())))
        throw new Exception("access denied");
    InvoiceManager.Pay(id);
}
```

- Repeat this for each WebMethod specifying the appropriate group (as discussed above).
- Build **SecureInvoiceService**.
- Run the client again and verify that vick can submit invoices, mike can approve invoices, aaron can pay invoices, and admin can do everything. Try to do something that isn't allowed for the current user and verify that you get "access denied".

7. Implementing a UsernameTokenManager

WSE 2.0 allows custom **UsernameToken** authentication through a **UsernameTokenManager** class. This allows you to specify a custom **Principal** object at runtime.

- In **SecureInvoiceService**, define a new class called **MyUsernameTokenManager**.
- Override the **AuthenticateToken** method as illustrated here (Create this in the WseSecurityHelpers file):

```csharp
public class MyUsernameTokenManager : UsernameTokenManager
{
    protected override string AuthenticateToken(UsernameToken token)
    {
        ...
    }
}
```

- Within **AuthenticateToken**, you can inspect the supplied token to perform authentication. You can also explicitly set its **Principal** property. Check the token's **Username** property and if its value is 'superman', create a new **GenericPrincipal** object and assign it to all of the roles that we defined above. For other usernames, delegate to the base class's **AuthenticateToken** method. Here's one way to do this:

```csharp
...
```

public class MyUsernameTokenManager : UsernameTokenManager
{
    protected override string AuthenticateToken(UsernameToken token)
    {
        ArrayList roles = new ArrayList();
        switch (token.Username)
        {
            case "superman":
                roles.Add(string.Format("{0}\User", Dns.GetHostName()));
                roles.Add(string.Format("{0}\Vendor", Dns.GetHostName()));
                roles.Add(string.Format("{0}\Manager", Dns.GetHostName()));
                roles.Add(string.Format("{0}\Accounting", Dns.GetHostName()));
                token.Principal = new GenericPrincipal(
                    new GenericIdentity(token.Username),
                    roles.ToArray(typeof(string)) as string[]);
                break;
            default:
                base.AuthenticateToken(token);
                break;
        }
        return token.Password;
    }
}

• To use this UsernameTokenManager, you need to configure it in the project's web.config file as illustrated here:

<configuration>
    
    ...<configSections>
        <section name="microsoft.web.services"
            type="Microsoft.Web.Services.Configuration.WebServicesConfiguration,
            Microsoft.Web.Services, Version=2.0.0.0, Culture=neutral,
            PublicKeyToken=31bf3856ad364e35" />
    </configSections>

    <microsoft.web.services>
        <security>
            <securityTokenManager qname="wsse:UsernameToken"
                xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
                type="SecureInvoiceService.MyUsernameTokenManager,
                SecureInvoiceService" />
        </security>
    </microsoft.web.services>

    ...
</configuration>

• Build SecureInvoiceService.

• Run the client again and verify that 'superman' can do everything that 'admin' can do even though there isn't a 'superman' account configured on the machine. Use the same password for superman as the other accounts you created.

B. Web Services Policy
In the last exercise you were able to secure the SecureInvoiceService application by requiring UsernameToken authentication and implementing authorization checks in your WebMethod code. Implementing security features this way is tedious and error-prone because it requires too much code from the developer. A better approach would be one that allows developers to enable such security features in a declarative manner, letting the infrastructure take care of the details.
WS-Policy is a new Web Services Architecture (WSA) specification that makes it possible for Web services to describe policies (e.g., requirements, preferences, and capabilities, etc.). WS-Policy allows you to define policy statements that mean anything you want. A policy statement consists of one or more policy assertions (see WS-PolicyAssertions). Other developers can then read policy statements and, assuming semantic understanding, deal with the service appropriately.

WS-SecurityPolicy is another WSA specification that defines a set of security-related policy assertions that can be used in your policy statements. Using WS-SecurityPolicy assertions, you can secure the SecureInvoiceService application without adding a single line of code to the various WebMethods. WSE 2.0 provides support for all of these specifications as you'll see below.

1. Getting Started
You can write policies for receiving messages as well as for sending messages. In this exercise you're going to write a policy that describes messages received by the Web service.

- Open the PolicyInvoiceClient and PolicyInvoiceService projects and familiarize yourself with the code.
- The code in PolicyInvoiceClient is nearly identical to the solution you wrote for SecureInvoiceClient. It sends UsernameTokens along with each invocation.
- The code for PolicyInvoiceService, on the other hand, is nearly identical to the SecureInvoiceService project you started with before adding security features.
- Build both projects.
- Run the client and verify the following:
  - The operations will authenticate a UsernameToken if supplied.
  - The operations don't require a UsernameToken (try commenting out one of the calls to ConfigureProxy in the client application).
  - Authorization is not being performed (authenticated users can invoke all operations).
- Now your goal is to provide all the same security features you implemented in the previous exercise without writing a single line of code – you'll accomplish this with a policy.

2. Writing a Policy
- Open policy.xml in the PolicyInvoiceService project and familiarize yourself with it. This illustrates what a policy statement looks like.
- Within <policyDocument> there are two child elements: <policies> and <mappings>.
- The <policies> element contains policy assertions. The policy states that a UsernameToken signature is required. It also states that the token must map to the MACHINE_NAME\User role for access (note: you can specify encryption requirements in a similar manner).

```xml
<policies
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
 xmlns:wse="http://schemas.microsoft.com/wse/2003/06/Policy">
 <wsp:Policy wsu:Id="ViewInvoices.asmx">
  <Integrity wsp:Usage="wsp:Required"
   xmlns="http://schemas.xmlsoap.org/ws/2002/12/secext">
   <TokenInfo>
    <SecurityToken>
     <TokenType>UsernameToken</TokenType>
    </SecurityToken>
   </TokenInfo>
   <MessageParts
    xmlns:rp="http://schemas.xmlsoap.org/rp"/>
  </Integrity>
 </wsp:Policy>
</policies>
```
• Search through the entire file and replace `MACHINE_NAME` with the actual name of the machine you're working on (if you don't know the name of your machine, go to Start Control Panel System Computer Name).

• The `<mappings>` element maps a resource to a policy assertion by policy ID. For example, the following maps `http://localhost/PolicyInvoiceService/ViewInvoices.asmx` to the policy defined above (with an ID of `#ViewInvoices.asmx`).

```
<policyDocument
    xmlns="http://schemas.microsoft.com/wse/2003/06/Policy">
  <mappings>
    <map to="http://localhost/PolicyInvoiceService/ViewInvoices.asmx">
      <default policy="#ViewInvoices.asmx" />
    </map>
    ...
  </mappings>
</policyDocument>
```

3. Configuring a Policy

• Now you need to configure WSE 2.0 to use your policy statement in this virtual directory. You do this by adding the following section to `PolicyInvoiceService`'s `web.config` file.

```
<configuration>
  <microsoft.web.services>
    <policy>
      <receive>
        <cache name="policy.xml" />
      </receive>
    </policy>
    ...
  </microsoft.web.services>
</configuration>
```

• Save `web.config` and test invoking the different operations.

• Verify that you must provide a `UsernameToken` now for all of the operations. Also verify that vick can submit invoices, mike can approve invoices, aaron can pay invoices, and that admin and superman can do everything. Try to do something that isn't allowed for a given user and verify that you get "access denied".

• Notice that you achieve the same functionality as the previous exercise but without writing any code in your WebMethods. Clients also know what behavior to expect by reading the policy.

**Note:** WSE 2.0 makes it possible to specify policy assertions on messages that are sent from a Web service as well. In this case, when a policy assertion fails, the WSE 2.0 infrastructure may attempt to modify the message to conform to the policy (e.g., by signing or encrypting, etc.). WSE 2.0 supports several WS-SecurityPolicy assertions out-of-the-box. To implement a custom policy, you must implement a policy assertion handler that plugs into the WSE 2.0 framework.
Summary
WSE 2.0 provides a framework for securing Web services with authentication, integrity, and privacy. You can accomplish these goals by sending tokens, signatures, and encrypted elements in SOAP messages. The WSE 2.0 WS-Security API makes it easy to get these features up and running within your client-side proxy and server-side WebMethod code. Fully implementing these features, however, requires you write quite a bit of security code within your WebMethods. WSE 2.0 also supports WS-Policy, which makes it possible to accomplish the exact same security goals declaratively.