# Securing Web Services with Visual Basic .NET

Zoran Zaev

This month, Zoran Zaev shows you how to secure your Web Services by implementing transport-level authentication over SSL with Visual Basic .NET. Then he calls the Web Services from a Visual Basic .NET Windows Form.

As you start thinking about actually deploying your Web Services in production, an essential question comes to your attention: How can I make sure that my Web Services are secure? Even more so, perhaps you've already deployed your Web Service and have only glanced over the topic of security. Or maybe your Web Service was deployed in an internal environment where you didn't think much security was necessary. In any of these situations, you'll likely want to look into how you can apply enough security for your Web Service, such that misuse or malicious use is prevented.

When you're implementing security within a system or an application, you have to take an integrated approach and examine all of the components in your system, all of your applications and modules, but also your business policies and processes. Systems or application security itself consists of distinctive areas of concern, such as authentication, authorization, auditing and logging, integrity via encryption, privacy via encryption, non-repudiation (not rejecting that an agreement was actually signed) and digital signatures, and the often left-out topic of availability with tasks such as load balancing, failover, and backup. Web security, in particular, can be addressed at two different levels: at the transport level, such as the level of HTTP, and at the application level, such as at the level of SOAP messages.

As you can see, security is a large topic. Therefore, for the purposes of this article, I'll particularly focus on authentication with Basic Authentication, and briefly touch on encryption via Secured Sockets Layer (SSL). Furthermore, I'll focus on transport-level security, and HTTP in particular. In last month's article, I showed you some ways that you can use to address Web Services security using Visual Basic 6 and MS SOAP Toolkit 2.0. This time, I'll use Visual Basic .NET.

## Getting started with a sample Web Service

I'll use a sample Web Service, which is a job submittal service, part of a sample Job Bank system (this is the same Web Service that I did for last month's article, but now it's implemented in VB.NET). Partners will be allowed to submit new job postings to this Web Service. You can easily imagine that companies that host job banks could use this kind of a service, to which other companies could submit new job postings.

Before we get started, you want to make sure that you have the necessary software. If you don't have the Microsoft .NET platform, you can do one of the following:

- Get the .NET redistributable package in order to be able to run .NET programs that you've downloaded (http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/829/msdncompositedoc.xml).
- Obtain the .NET Framework SDK that allows you to write, build, test, and deploy applications (http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/000/976/msdncompositedoc.xml).
- Use Visual Studio .NET to do the same thing that the .NET Framework SDK allows you to do, just visually (for the examples in this article, you'd only need Visual Basic .NET).

Finally, you'll need to have an operational Web server. IIS 4 or IIS 5 would be fine. Both the version included with the Windows server, as well as the versions with the Windows Professional/ Workstation operating system, would work. Of course, you'll need to have one of the mentioned .NET packages installed on it.

Getting started with this sample Web Service is very easy, especially if you're using Visual Studio .NET. Open Visual Studio .NET and choose to create a new project. Then select "Visual Basic .NET Projects" as the project type, and select "ASP.NET Web Service" as the template. Give the Web Service a name "JobBank" by modifying the URL for the Web site to http://localhost/JobBank. Click OK. The Visual Studio .NET wizard will build the template project for you. Within the Solutions Explorer (typically on the right-hand side of the screen), right-click on the Service1.asmx file and rename it to JobBankSvc.asmx. This Web Service project template has several files within it (see Figure 1).

First, you have the JobBankSvc.asmx class module

that will contain the Web Service code (the server side of the code). The JobBank.vsdisco file is a discovery file with information about the site. The Global.asax file contains common site-wide information similar to the old Global.asa file under classical ASP. The AssemblyInfo.vb file has information about the assembly that contains this Web Service, and Web.cofig contains Web site configuration information. Finally, you have the references to the namespaces that will be used in this Web Service implementation.

Next, let's make some modifications to this Web Service template. First, go ahead and right-click on the JobBankSvc.asmx file and select "View Code." You'd probably want to modify the default namespace provided by Visual Studio—that is, the "tempuri" namespace—into something that will be unique to your implementation. I modified this to Namespace := "http://myCompany.org/JobBank." The final code of JobBankSvc.asmx looks like this:

```
Imports System.Web.Services

<WebService(Namespace:= _
    "http://myCompany.org/JobBank", _
Description:="Job Bank Service")> _
Public Class JobBankSvc
    Inherits System.Web.Services.WebService

    <WebMethod( _
    Description:="This method expects to " & _
    "receive companyID, expireDate, " & _
    "jobTitle, jobDescription, salaryAmount." & _
    "It will return a confirmation message")> _
 _
    Public Function JobAdd( _
        ByVal companyID As Short, _
        ByVal expireDate As Date, _
        ByVal jobTitle As String, _
        ByVal jobDesc As String, _
        ByVal salaryAmount As Double) As String

        If companyID = CDbl("1") Then
            JobAdd = "Your job with title: " & _
            jobTitle & " and description: " & _
            jobDesc & " was received " & _
            "successfully on " & Now & _
            ". The salary requested is: " & _
            salaryAmount & _
            " and the expiration" & _
            "date of this posting is: " & _
            expireDate & "."
        Else
            JobAdd = _
            "Your company is not allowed" & _
            " to post jobs to our Job Bank."
        End If
    End Function

End Class
```

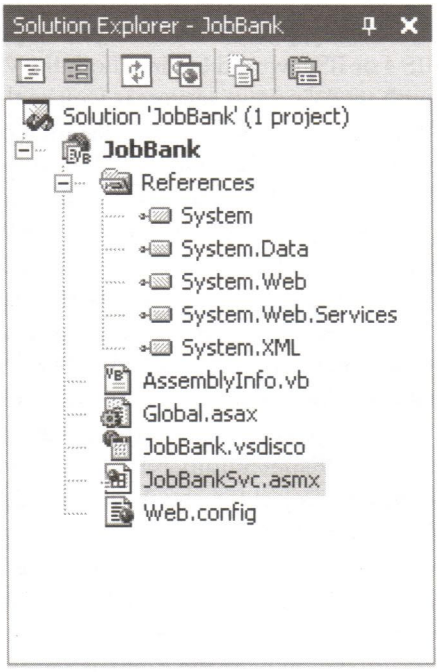


**Figure 1.** Web Service template project created in Visual Studio.

In summary, I added one method to this Web Service, the JobAdd method. This method expects few parameters from the application that will be calling this Web Service. In your own case, you can make this Web Service as complicated as you like. Once you're done, go ahead and build this solution (you can use the Build command from the Build menu option). Finally, you can test the Web Service by pressing F5 (before you do this, make sure your project has the JobBankSvc.asmx file as the default file). If by some chance you get an error "CS2001..." make sure you see the Microsoft support article Q313105 at http://support.microsoft.com/default.aspx?scid=kb;en-us;Q313105. One of the ways you can resolve this problem is by increasing the permissions to your directory that .NET uses for temporary files (such as the C:\Windows\Temp folder). The Microsoft.NET approach of testing Web Services couldn't be any easier, I have to admit. What happened when you pressed F5 is that you were simply redirected to the URL of the Web Service, where you can select the particular operation out of the Web Service that you'd like to test, such as JobAdd. In this case, the URL for testing the operation of JobAdd within the JobBank Web Service is http://localhost/JobBank/JobBankSvc.asmx?op=JobAdd. Next, let's see how you can add security to your VB.NET Web Service.

## Adding security to the server side of the sample Web Service

Since you're using the HTTP transport-level security, the difference on the server side comes within the Web server configuration. Let's see what that means. You'd want your users to authenticate before they can access your Web Service. Of course, you can require them to submit username/password credentials within the SOAP message itself. This would be an example of implementing security at the application level, and, in such a case, you'd have to change your server-side Web Service implementation in order for this model to be supported. However, if you're implementing your authentication at the transport protocol level (in this case HTTP), you'll want to leverage the typical HTTP authentication capabilities, such as Basic Authentication.

Basic Authentication is a simple authentication

protocol that's part of the HTTP protocol. Virtually all Web servers, browsers, and Web Services implementations support Basic Authentication. Basic Authentication works well across most firewalls and proxy servers, and within most SOAP/Web Services toolkits. This widespread availability and support is one of the strongest reasons to use Basic Authentication. However, the negative aspect of Basic Authentication is that it sends the password in Base64-encoding format, which is so easy to decode that you might as well consider it clear text. The way to overcome this serious drawback of Basic Authentication is to use SSL.

Enabling Basic Authentication on your server for your Web Service is quite easy. Under IIS, open your Web server management console of Internet Services Manager (MMC or Microsoft Management Console), typically found under your Administrative Tools. Within the MMC, locate the name of your Web server and expand its branch. This will show you the various directories and virtual directories on your Web server. Right-click on your Web Service virtual directory (in this case, that's JobBank—Visual Basic .NET created this automatically when you created the Web Services project) and select Properties. Next, go to the Directory Security tab and click Edit under the "Anonymous access and authentication control" section. Within the Authentication Methods window that opens up, select Basic Authentication and unselect all other options. If you like, you can select a default domain for your client Web Service requests by clicking on the Edit button within the Authentication Access section. For this example, this IIS screen looks like Figure 2 (this was taken from IIS within Windows XP; it's slightly different from what it looks like under Windows 2000 or Windows NT, but it essentially has the same information).

Now if you try to activate your Web Service by going to http://localhost/JobBank/JobBankSvc.asmx?op=JobAdd, you'll get the standard Basic Authentication box. After you've entered your credentials, you'll receive your Web Service test page (see Figure 3).

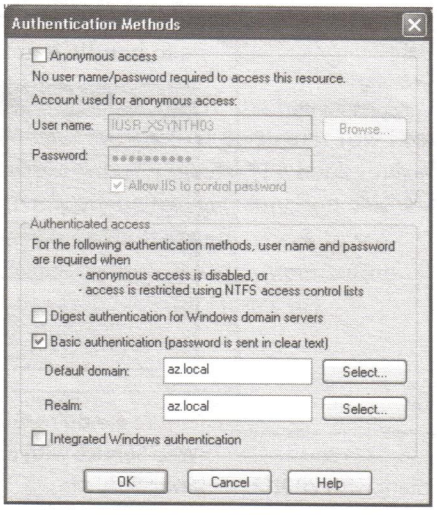


Figure 2. Setting up Basic Authentication under IIS.

## How about encrypting those credentials?

As I mentioned earlier, Basic Authentication isn't secure, because the username and password are exchanged in plain-text format. One way to address this issue is to use SSL and encrypt the data exchange. This would encrypt both the authentication credential exchange and the data exchange. SSL is much slower than HTTP traffic without SSL, so that's something you'd want to keep in mind. An appropriate approach may be to encrypt the initial authentication credentials exchange, and then switch to an unencrypted communication. This hybrid approach, as well as other application-level Web Services approaches, I'll have to leave for future articles.

Adding SSL to your Web Service implementation doesn't require that you change anything in your server-side implementation of your Web Service. It requires that you go to a Certificate Authority (CA), such as VeriSign at www.verisign.com, Thawte at www.thawte.com, Entrust at www.entrust.com, or your own company CA if you have one, and request a Web server certificate for IIS. You can even request a test certificate—if you just want to use one for testing purposes (for example, VeriSign offers test server certificates). I'm not going to go through the details of obtaining a certificate and installing it, as that's a somewhat lengthy discussion. The CAs' Web sites tend to have pretty good instructions for how to do this, and I recommend that you follow their directions. Once you install your certificate on your Web server, you'll need to enable SSL. To do this, open the Internet Information Server MMC, go to your Web server icon, right-click on it and select Properties, select the Directory Security tab, and click on Edit under the Secure Communications section. The Secure Communications window will open up. Here, you can select to "Require secure channel (SSL)" for the Web site (see Figure 4). You can also set it up to require SSL only to a particular directory. Figure 4 shows what that would look like.

I also recommend that you enable "Require 128-bit encryption" as long as you know that your client Web Service applications can support it. This is all it takes to enable the server with Basic Authentication and SSL. Now

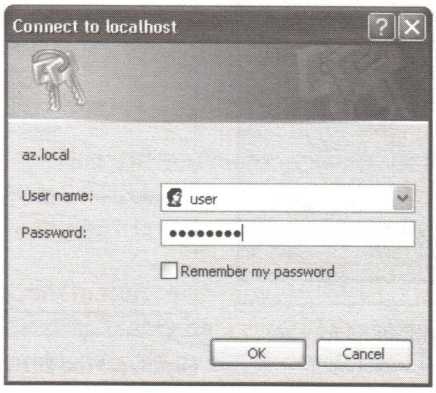


Figure 3. Basic Authentication challenge (as seen in the browser when testing the Web Service).

you can go to https://xsynth03/JobBank/JobBankSvc.asmx?op=JobAdd (note that I'm using "https" this time) and test your Web Service. Note that the server name in the URL should match the name to which the SSL certificate was issued and where your Web Service has been deployed. In my case, I used the server name, but more often you may use the full domain name, such as www.xmldevelopernewsletter.com. You specify this choice at the time when you're requesting your Web Service certificate.

## Enabling authentication within the Web Services client

By this time, you've had the chance to build your Web Service and test it with Basic Authentication over SSL. Now, let's explore invoking this Web Service from another application. First, let's try to do this from within a Visual Basic .NET application.

In Visual Studio .NET, create a new Visual Basic Project with the Windows Application template. This is a Windows Forms-type application, what used to be a "Standard EXE" application under Visual Basic 6. Within your Solution Explorer, you should get the screen shown in Figure 5.

I renamed the solution and the default VB form to WebServicesClient. Next, you'll want to add a Web Reference to the Web Service you created earlier. Go to the Project menu and select Add Web Reference. In the address for the Web Service, enter https://xsynth03/JobBank/JobBankSvc.asmx?op=JobAdd (that is, the address of your Web Service, by modifying the server name and path to match your own configuration). Since this Web Service requires authentication, you'll be prompted to enter your credentials, as shown in Figure 6. Remember to enter the username and password of an account that has access to this Web Service, and make sure that you enter the correct domain. In Windows environments, the domain will be a Windows 2000 or NT domain if the account is a domain account, or the machine name if the account is a local account to the Web server where this Web Service is being hosted.

Finally, you can click on Add Reference, which adds this Web Reference to your project. Now you're ready to create the VB.NET form. The code for the form is shown here:

```
Imports System.Net
Public Class WebServiceClient
    Inherits System.Windows.Forms.Form

    Private Sub btnCall_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnCall.Click

        Dim strResult As String
        Dim clsJobBank As _
            New xsynth03.JobBankSvc()
        Dim oCreCache As CredentialCache = _
            New CredentialCache()
        Dim oCre As NetworkCredential = New _
           NetworkCredential("xmlUser", _
           "test", "xsynth03")
        oCreCache.Add(New _
            Uri(clsJobBank.Url), "Basic", oCre)
        clsJobBank.Credentials = oCreCache

        strResult = clsJobBank.JobAdd( _
            txtCompanyID.Text, _
            txtExpireDate.Text, _
            txtJobTitle.Text, _
            txtJobDesc.Text, _
            txtSalaryAmount.Text)
        txtReply.Text = strResult
    End Sub
End Class
```

The important part within this code that handles the authentication is the part that creates the CredentialCache object and adds the NetworkCredential object, containing your credentials to it. Also, you have to import the System.Net in order to keep your code a bit shorter. Everything else is the same as when placing the Web Service call unauthenticated. When you run the form, it will look like Figure 7.

That's it for this month. Keep in mind that another possible way to address the security of Web Services is to handle it at the application level.

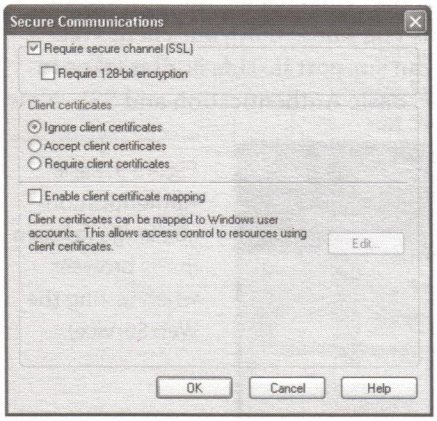


Figure 4. Enabling SSL under IIS for the Web Service directory.

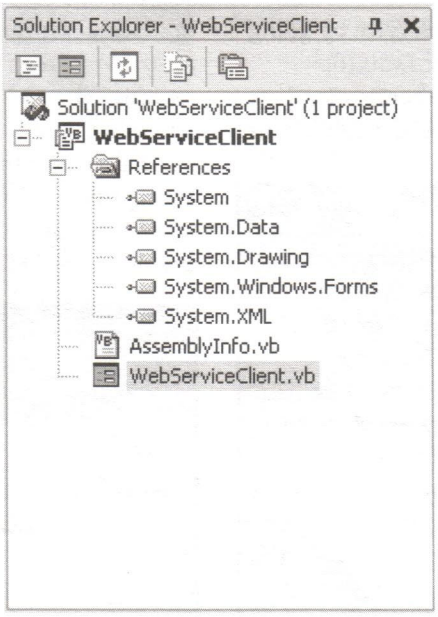


Figure 5. Web Services client with VB.NET.

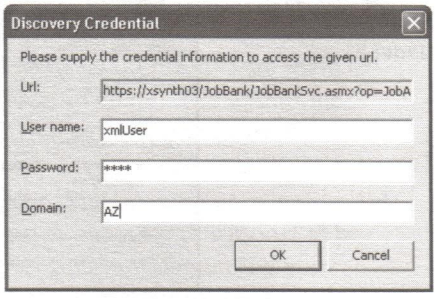


Figure 6. Adding a Web Reference to a Web Service with Basic Authentication and SSL.

Application-level security brings its own benefits and drawbacks as it compares to transport-level security, but I'll leave that discussion for another time. ▲

Zoran Zaev works as the principal software architect for xSynthesis LLC, a software and technology services company in the Washington, DC, area. He enjoys helping others realize the potential of technology, and when he isn't working, he spends considerable time writing articles such as this one and books (for example, he co-authored *Professional XML Web Services* and *Professional XML 2nd Edition* with Wrox Press). Zoran's research interests include complex systems that often involve XML, highly distributed architectures, systems integration, Web systems, and Web system usability, as well as the application of these concepts in newer areas such as biotechnology. When he's not programming or thinking of exciting system architectures, Zoran can be found traveling, reading, and exploring various learning opportunities. zzaev@yahoo.com.

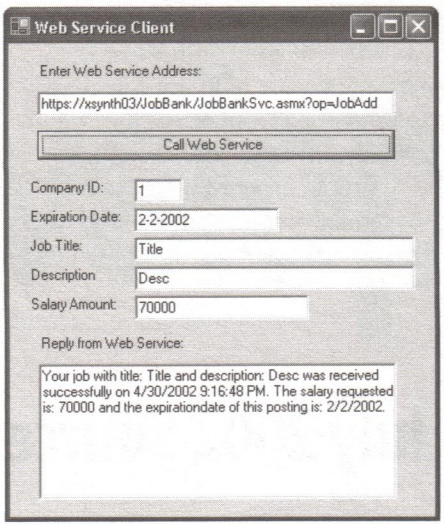


**Figure 7.** VB.NET form calling a secured Web Service over SSL.