PINNACLE

XML Developer

XML Solutions for Client/Server, Web-Based, and Business-to-Business Systems

Securing Web Services with Visual Basic 6.0 and MS SOAP Toolkit 2.0

Zoran Zaev

DOWNLOAD

This month, Zoran Zaev shows you how to secure your Web Services by implementing transport-level authentication with Visual Basic. Then, he calls the Web Service from Visual Basic, VBScript/ASP, and Perl. Finally, he implements transport-level SSL encryption to the Web Service used in his examples.

BY now, you've probably heard quite a bit about Web Services and how to build Web Services in your favorite environment. My previous articles in *XML Developer* have described how to create and access Web Services in various programming languages and on a variety of platforms.

As you start thinking of 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? The question is even more critical if you've already deployed your Web Service and have only glanced over the topic of security (or if your Web Service was deployed in an internal environment where you didn't think much security was necessary). In either of these situations, you'll likely want to look into how you can apply enough security for your Web Service to prevent both misuse and malicious intent.

Continues on page 3

June 2002

Volume 3, Number 6

DOWNLOAD

Accompanying files available online at www.xmldevelopernewsletter.com

Securing Web Services...

Continued from page 1

In this article, I'll talk a lot about Web Services security, but let me first clearly state what I will and won't cover. Security is a complex subject, and I'd need much more than one article to cover all of the important security aspects as it relates to Web Services. When you're implementing security for a system or an application, you've got to take an integrated approach. This includes examining all of the components in your operating system, plus reviewing all of your applications and modules. You must also look at your business policies and processes.

Systems or application security itself consists of distinctive areas of concern, such as authentication, authorization, auditing and logging, integrity and encryption, privacy and encryption, non-repudiation (not rejecting that an agreement was actually signed), and digital signatures. It also includes the often-left-out topic of availability, which includes tasks such as load balancing, failover, and backup. Web security in particular can be addressed at two different levels: the transport level (for example, at the HTTP level) and the application level (for example, at the level of SOAP messages). As you can see, security is a broad topic. For the purposes of this article, I'll particularly focus on authentication, and briefly touch upon encryption. Furthermore, I'll cover transport-level security, and HTTP in particular. I'll use Visual Basic 6.0 and MS SOAP Toolkit 2.0 in the examples shown in this article. In an upcoming issue, I'll look at implementing Web Services security with Microsoft's new .NET platform. While the Microsoft future belongs to .NET, Visual Basic 6 and COM+ systems are numerous and they'll very likely be out there for quite some time.

A sample Web Service

For this article, I'll use a sample Web Service that I created for the October 2001 issue of *XML Developer* and show you how to add security to it. This Web Service was created with Visual Basic 6.0 using the MS SOAP Toolkit 2.0. This sample Web Service is a job submittal service, part of a sample Job Bank system. Partners will be allowed to submit new job postings to this Web Service. You can imagine that companies that host job banks could easily use this kind of a service, which other companies could submit new job postings. I said in the October 2001 issue that, in a real-world scenario, this Web Service will likely be more complex, and security will be applied, so that not everybody can submit a new job posting.

Before we get started, make sure that you have the necessary software. If you don't have the MS SOAP Toolkit 2.0, you need to obtain it from http://msdn.microsoft.com/webservices/ and install it on your computer. I built this Web Service on a Windows 2000 Server machine with IIS 5, but you should be able to use *Windows 2000 Professional with IIS on it. Additionally, I* used Visual Basic 6.0 with SP4 to create the server component to this Web Service. The client components of the Web Service are built in a few different languages: Visual Basic 6, VBScript (under ASP), JScript (under ASP), and (just to show you that you can call this Web Service from a language environment that's substantially different and that's not COM+-based) I created a Web Service client in Perl.

You'd create the server side of this Web Service as if you didn't have to worry about security at all. I'll create a COM object using Visual Basic 6. The COM object will consist of one class and one function that can be called to add jobs to the Job Bank system. The function would look like this:

```
Public Function JobAdd( _
   ByVal companyID As Integer, _
   ByVal expireDate As Date, _
   ByVal jobTitle As String, _
   ByVal jobDesc As String, _
   salaryAmount As Double) As String
   If companyID = "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 isn't allowed" & _
       " to post jobs to our Job Bank."
   End If
End Function
```

If this COM object wasn't compiled on your server, you'd have to register it on the server by typing regsvr32 wsJobBank.dll at the command prompt within the directory where you've placed the DLL (you can obtain this file from the Download file that's available at www.xmldevelopernewsletter.com, or you can create it by using the code provided and compile it using Visual Basic 6). I'm not going to go into further detail on this code or some of the other files needed for the server side of the Web Service implementation (for example, the WSDL file, the WSML file, and so on).

The second part of the server-side Web Service setup is to set up your Web server so that it receives the requests for your Web Service. For this, you'll need to create a virtual directory under IIS. I created a virtual directory within the Web server (IIS, or Internet Information Server) with the name of "wsJobBankSec" pointing to the physical location where the WSDL and WSML files are located.

This time, however, I'll use the ASP listener instead of the ISAPI listener I used in the October 2001 issue. The ISAPI listener works as an ISAPI application that handles any WSDL requests. This works very well for many situations. However, when it comes to implementing basic authentication, there are some challenges. The ASP listener is slower, but it's more flexible and allows you to do additional processing (for instance, parsing or verifying input) as well as verifying security.

In order to implement the ASP listener, you'd want to create the file wsJobBankListener.asp and place it within your Web Service virtual directory. This file's security settings should be set to basic authentication. Anonymous access shouldn't be allowed (I'll explain later in this article how to specify these settings). The ASP file is very simple:

```
<%@ LANGUAGE = VBScript %>
<% Response.ContentType = "text/xml" %>
<%
set soapserver = server.CreateObject _
   ("MSSOAP.SoapServer")
wsdl = Server.MapPath("wsJobBank.WSDL")
wsml = Server.MapPath("wsJobBank.wsml")
call soapserver.init(wsdl, wsml)
call soapserver.SoapInvoke(request, response, "")
%>
```

The ASP listener simply takes the SOAP message from ASP's Request and passes it to the SOAPServer object for execution. The SOAPServer object will return the result of the Web Service as an XML document and write it to ASP's Response object, where it will flow back to the client.

If you created the WSDL application using the ISAPI listener, you'll need to modify the <soap:address> element "location" attribute of your WSDL file to use the ASP listener. The modified section of your WSDL file will look like this:

```
<soap:address location=
"http://localhost/wsJobBankSec/wsJobBankListener.asp"
/>
```

Adding security

So you may wonder, if the code on the server side of the Web server is no different from the "unsecured" code, how do you secure your Web Service? Of course, you can require your users to submit username/ password credentials within the SOAP message itself. This would be an example of implementing security at the application level, and, in that case, you'd have to change your server-side Web Service implementation in order for this model to be supported.

In my case, I'm going to use the HTTP transport-level security so that the difference on the server side is handled by your Web server configuration. My goal is have my users authenticate before they access the Web Service. If you're implementing your authentication at the transport protocol level (in this case, HTTP), you can 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. This widespread availability and support is one of the strongest reasons to use basic authentication. However, there's a negative aspect to basic authentication: It sends the user's password in Base64 encoding format, a format that's 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 Secured Sockets Layer (SSL) encryption. I'll cover this approach toward the end of this article.

Enabling basic authentication on your server for your Web Service is quite easy. Under IIS, you'd open the 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 "wsJobBankSec") and select Properties. Next, go to the Directory Security tab and click Edit under "Anonymous access and authentication control." Within the Authentication Methods window that opens up, select Basic Authentication and unselect all other options. If you like, you may also select a default domain for your client Web Service requests by clicking on the Edit button within the Authentication Access section.

There's one other configuration setting to be made, and that's to change the authentication of the WSDL file within this directory. This file needs to be accessible to everybody, so you'll want to enable Anonymous access to this file.

Enabling authentication

With basic authentication, most of the required code changes are all on the client side. For a Visual Basic 6 Web Services client, the changes are quite simple, requiring only a couple of extra lines of code added to the code that doesn't implement any authentication. Here it is:

```
Private Sub txtExecute_Click()

   Dim soapClient
   Set soapClient = CreateObject _
      ("MSSoap.SoapClient")

   soapClient.mssoapinit txtWSDL
   soapClient.ConnectorProperty _
      ("AuthUser") = "XMLDevUser"
   soapClient.ConnectorProperty _
      ("AuthPassword") = "1234"

   txtResult.Text = soapClient.JobAdd _
      (1, #12/31/2001 11:15:00 AM#, _
      "XML Project Manager", _
      "Must have 10 years of technical " & _
      "experience...", 80000)
   Set soapClient = Nothing

End Sub
```

The most important lines here are the ones that specify the authentication credentials. These are the ConnectorProperty lines that set the username and password needed to obtain access to the Web Service.

Here's an example of a client Web Service implementation using VBScript in an ASP environment (in the Download file, you'll also find a version done in JScript/ ASP):

```
<%@Language = "VBScript"%>
<%
   Set objSoapClient = _
      Server.CreateObject ("MSSOAP.soapClient")
   strWSDL = "http://localhost/wsJobBankSec/" & _
      "wsJobBank.WSDL"

   objSoapClient.ClientProperty _
      ("ServerHTTPRequest") = True
   objSoapClient.mssoapinit (strWSDL)
   objSoapClient.ConnectorProperty _
      ("AuthUser") = "XMLDevUser"
   objSoapClient.ConnectorProperty _
   ("AuthPassword") = "1234"

   Response.Write ("<HTML><HEAD><TITLE>" & _
      "JobBank: Add Jobs Via VBScript" & _
      "</TITLE></HEAD><BODY>The Response " & _
      "from the Web Service is: " & "<BR><BR>")
   Response.Write (objSoapClient.JobAdd (1, _
      "12/31/2001 11:15:00 AM", _
      "XML Project Manager", _
      "Must have 10 years of technical " & _
      "experience...", 80000))
   Response.Write ("</BODY></HTML>")

   Set objSoapClient = Nothing
%>
```

The key section here is the section that specifies the ConnectionProperty settings. In the script environment, this is a little different from the Visual Basic code because you have to do the following:

- Set the ServerHTTPRequest property to True.
- Load the WSDL file before setting the ConnectionProperty.

The Perl Web Service client is quite similar to what we'd have if we didn't have to authenticate. Here, I'm using the SOAP::Lite Perl library (for more on Perl Web Services implementation, see the article "Connecting Web Services: Perl and Java" in the January 2002 issue). The difference between the version that supports authentication and the one that doesn't is in the very first subroutine. That subroutine overwrites the default basic authentication implementation and provides the username/ password credentials to be used in authenticating this request:

```
use SOAP::Lite;
sub
SOAP::Transport::HTTP::Client::get_basic_credentials {
    return 'XMLDevUser' => '1234';
}
print "HTTP/1.0 201 Ok \n";
print "Content-type:text/html\n\n";
print "<HTML><HEAD><TITLE>JobBank:".
   "Add Jobs Via PERL";
print "</TITLE></HEAD><BODY>The Response ".
   "from the Web Service is: ";
print "<BR><BR>";
print SOAP::Lite
   ->uri ("http://tempuri.org/message/")
   ->on_action(sub { return
   '"http://tempuri.org/action/clsJobs.JobAdd"'})
   ->proxy ("http://localhost/wsJobBankSec/".
   "wsJobBankListener.asp")
   ->JobAdd (SOAP::Data->type(short => 1)->
        name('companyID'),
      SOAP::Data->type(dateTime =>
        '2001-12-31T16:15:00Z')->
        name('expireDate'),
      SOAP::Data->type(string =>
        'XML Project Manager')->
        name('jobTitle'),
      SOAP::Data->type(string =>
        'Must have 10 years of technical '.
        'experience...')->name('jobDesc'),
      SOAP::Data->type(double => 80000)->
        name('salaryAmount'))
   ->result;
print "</BODY></HTML>";
```

In all of these examples, it's critical that you remove the application mapping for the ISAPI listener from your WSDL file. If you don't remove the ISAPI mapping, the client can use the ISAPI listener to access your Web Service. If that listener isn't secured (and I've only discussed securing the ASP listener), you'll have a security hole.

Encrypting credentials

As I said 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. SSL will encrypt both the authentication credential exchange and the actual data exchange. Not only will your credential be secured from eavesdropping, but the data that's being moved will also be safer.

SSL is much slower than HTTP traffic without SSL, so that's something you'll want to consider in a highly scalable design. An appropriate approach may be to encrypt the initial authentication credentials exchange, and then switch to an unencrypted communication for the remainder of the conversation. This hybrid approach, as well as other application-level Web Services approaches, I'll have to leave for future issues.

Adding SSL to your Web Service implementation doesn't require that you change anything in your server-side implementation of your Web Service. It does requires you to 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 (VeriSign offers test server certificates). I'm not going to go through the details of

Continues on page 9

Securing Web Services...

Continued from page 5

obtaining a certificate and installing it, as that's a somewhat lengthy discussion. The CAs' Web sites tend to have good instructions for how to do this, and I'd recommend that you follow their directions. Once you install your certificate to your Web server, you'll need to enable SSL. You can do this by following these steps:

1. Open the Internet Information Server MMC.
2. Find your Web Server in the left-hand panel.
3. Right-click on it and select Properties.
4. Select the Directory Security tab.
5. Click on Edit under the Secure Communications section. The Secure Communications window will open up.
6. Select "Require secure channel (SSL)" for the Web site.

You can also require SSL-only for a particular directory (for instance, your Web Service directory), or you can choose to be even more specific and set SSL at the level of the ASP listener page.

Finally, you have to modify the WSDL file to reflect the fact that the ASP listener is now only accessible by SSL. Also, this line has to have the correct name that's stated on the certificate. In my case, that's the name of my server "titan." This name can be a full domain name such as www.xmldevelopernewsletter.com. Here's a WSDL file modified for SLL:

```
<soap:address location=
"https://titan/wsJobBankSec/wsJobBankListener.asp"
/>
```

With these changes, your clients now will have access to the Web Service listener via SSL. No modification to the client-side Web Services code is necessary.

Conclusion

That wraps up this introduction to securing your Web Service. By leveraging the security built into IIS, you can provide a first-line defense against misuse or malicious use. Before I finish, though, I want to repeat my warning from the start of this article: Tackling one part of the security puzzle isn't sufficient. You must consider all aspects of your system, application, and business processes. ▲

DOWNLOAD SECURE.ZIP at www.xmldevelopernewsletter.com

Zoran Zaev works as the principal software architect for xSynthesis LLC, a software and technology services company in the Washington, D.C. 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. zoran@xsynthesis.com.