Using WS-Security with Crystal Reports 2008

This document demonstrates how Crystal Reports 2008 and its XML and Web Services data access driver can make use of a Web Service that uses WS-Security. It demonstrates how to create a sample WS-Security-enabled Web Service and then the steps required to connect Crystal Reports 2008 to it using the XML and Web Services data access driver included in Crystal Reports 2008.

Contents

Introduction

Crystal Reports 2008 introduces many new features, including a much-improved XML and Web Services driver. Support for more types of parameters has been included in this new driver. Compatibility with Web Services has been increased with the use of the newer Apache Axis 2 Web Services stack. Covered in this document is the ability to connect to Web Services using WS-Security.

Recall that Crystal Reports is not just made up of the Crystal Reports designer. Reports are often deployed to servers so that users can easily view them. A report that is displayed which consumes a WS-Security enabled Web Service will need to supply authentication credentials. In the Crystal Reports designer, this could be done interactively. However, in a server environment, the user can't always supply credentials to the server. Report developers might not want to force users to enter credentials. More importantly, the type of authentication used by the Web Service may not match up with your Business Objects Enterprise security or your corporate security. As a result, you may need to supply special credentials for that Web Service and need to supply it in a seamless way. This is done using a credential callback handler.

The XML and Web Services driver in Crystal Reports 2008 uses the Apache Rampart WS-Security module, which works with Apache Axis 2 to provide support for WS-Security. Apache Rampart uses a credential callback handler so that authentication can be controlled by the developer. The handler's design only requires following a common interface, leaving the implementation details to the developer. Crystal Reports 2008's XML and Web Services driver accepts a configuration file for the Web Service connection so that the callback handler can be located and associated with calls to the service.

Note: Vendors of Web Services that use WS-Security to secure their valuable information may want to provide a credential handler and associated WS-Security configuration file for their service to help customers using Crystal Reports to be able to more easily use their services. This callback handler should work for other customers using the Rampart WS-Security module.

This document explains how to create a WS-Security enabled Web Service and then how to configure Crystal Reports 2008 to be able to retrieve data from it. This sample uses username token authentication to secure access to the Web Service. A sample Rampart authentication callback handler is described. This document provides the essential information to enable Crystal Reports 2008 users to connect to WS-Security enabled Web Services with various types of authentication.

A Microsoft IIS Web Service is used as the sample Web Service that is developed, using ASP.NET (ASMX) with WSE 3.0. This helps to demonstrate the interoperability between the separate stacks: the Java-based Apache Axis stack and the Microsoft .NET-based ASP.NET stack.

The development tool used for the walk-through steps is Visual Studio 2005.

Creating the Web Service

After installing WSE 3.0 with support for Visual Studio 2005, create a Web Service (C# is used in the examples). Follow the instructions for adding WS-Security and click Username Token authentication. If you take the defaults, a Service1.asmx file will be created with a few other support files. The result should include code similar to this in the Service1.asmx.cs file that is produced:

```
[WebService(Namespace = "http://www.CrystalReports.com/")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]

[Policy("myPolicy")]

  public class Service1 : System.Web.Services.WebService
```

***** For initial testing, you should comment-out the Policy attribute to turn off WS-Security.**

Return Some Data from your Service

You need to return something from your service. Crystal Reports 2008 supports returning arrays of objects from a service and treats it like a data set.

*Note: At time of writing, Crystal Reports 2008 does not support serialized ADO.NET **DataSet** or **DataTable** objects. This limitation is expected to be removed in future service packs.*

A full-featured service would return some valuable data. For now, create a simple array of objects and return them through a service method similar to this:

```
public class ProcessInfo
{
    public ProcessInfo()
    {
    }

    public int processID;
    public string processName;
    public int numberOfThreads;
}

[WebMethod]
public ProcessInfo[] GetProcesses()
{
    Process[] processes = Process.GetProcesses();
    ProcessInfo[] processInfoArray = new
           ProcessInfo[processes.GetLength(0)];
```

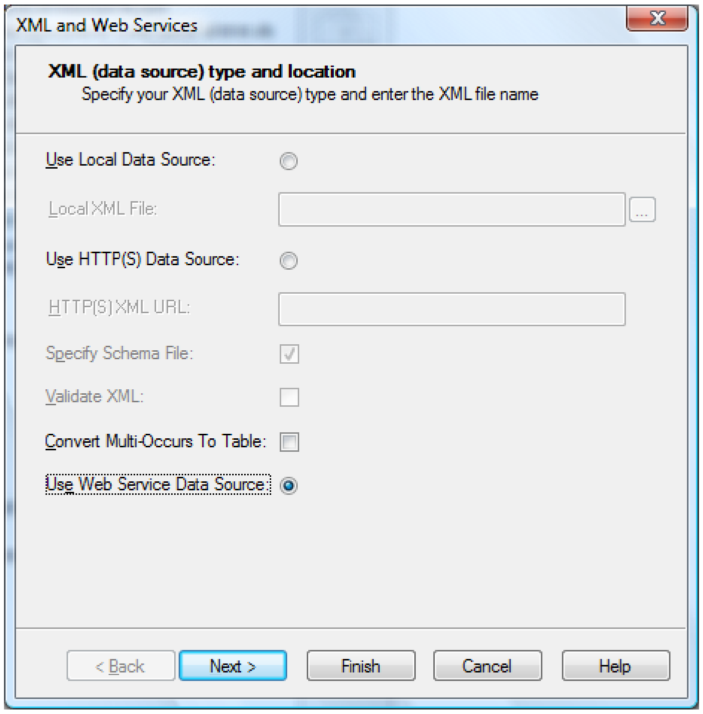
```
        for (int i = 0; i < processes.GetLength(0); i++)
        {
            processInfoArray[i] = new ProcessInfo();
            processInfoArray[i].processID = processes[i].Id;
            processInfoArray[i].processName = processes[i].ProcessName;
            processInfoArray[i].numberOfThreads = processes[i].Threads.Count;
        }

        return processInfoArray;
    }
```

The new Web Service method is thus, **GetProcesses**. It returns a list of the processes running on the system and a few items of information about them. This is simple enough as a data set to get things up and running without additional setup.

At this point, you should be able to run your service, get the URL to the WSDL and use that URL in Crystal Reports. The URL will be similar to this: http://localhost:2937/WSEServiceSample/Service1.asmx?WSDL

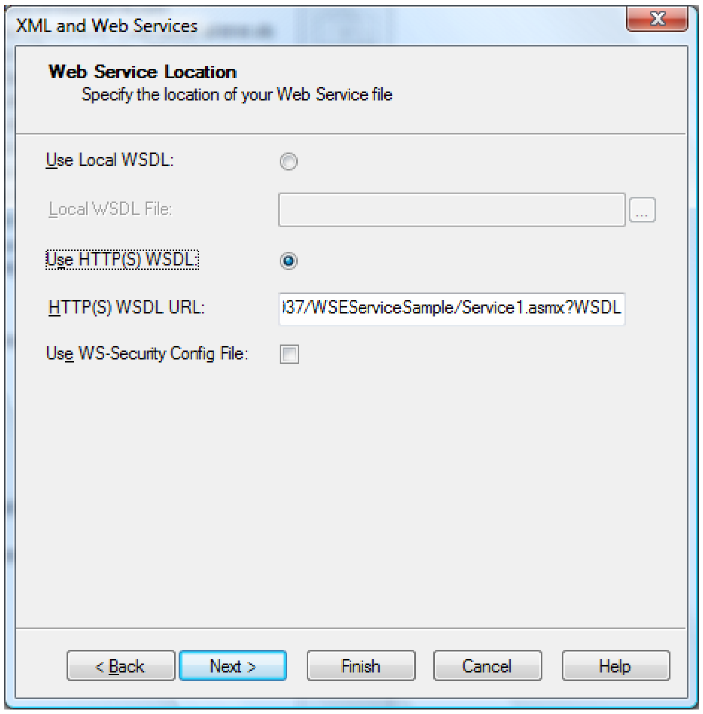
You may need to click on Service1.asmx in your browser depending on how Visual Studio launches the browser to show the service.

In the Crystal Reports XML and Web Services data source wizard, select Web Service as shown below:


XML and Web Services
XML (data source) type and location
Specify your XML (data source) type and enter the XML file name
< Back
Next >
Finish
Cancel
Help

Use Local Data Source: ○
Local XML File:
Use HTTP(S) Data Source: ○
HTTP(S) XML URL:
Specify Schema File: ☑
Validate XML: ☐
Convert Multi-Occurs To Table: ☐
Use Web Service Data Source: ●

Then paste the URL to the WSDL in the HTTP WSDL box as shown below:



Use Local WSDL: ○

Local WSDL File:

Use HTTP(S) WSDL: ●

HTTP(S) WSDL URL: i37/WSEServiceSample/Service1.asmx?WSDL

Use WS-Security Config File: ☐

At this point, you should be able to complete the wizard, add fields to the report and view the results, as long as you commented-out the Policy attribute in the Web Service.

Enabling Security in the Web Service

This is the first step to turning on WS-Security in your Web Service.

Start by enabling security: re-establish the Policy attribute at the top of the Web Service implementation, as shown here:

```
    [Policy("myPolicy")]

      public class Service1 : System.Web.Services.WebService
```

Next, open the wse3policyCache.config file that should have been generated for your project. In that file, you may need to change the name of the allowed user to be a user account on your server or in the Windows network domain that the user account belongs to.

You also need to comment-out the **requiredActionHeader** elements. The result should be similar to this:

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <extensions>
    <extension name="authorization"
type="Microsoft.Web.Services3.Design.AuthorizationAssertion,
Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
```

```
    <extension name="usernameOverTransportSecurity"
type="Microsoft.Web.Services3.Design.UsernameOverTransportAssertion,
Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
    <!--<extension name="requireActionHeader"
type="Microsoft.Web.Services3.Design.RequireActionHeaderAssertion,
Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />-->
  </extensions>
  <policy name="myPolicy">
    <authorization>
      <allow user="YOUR-DOMAIN\Your-Account" />
      <deny user="*" />
    </authorization>
    <usernameOverTransportSecurity />
    <!--<requireActionHeader />-->
  </policy>
</policies>
```

Be sure to change the `<allow user="YOUR-DOMAIN\Your-Account" />` element to include your domain and account.

Custom Token Management

The Web Service needs a custom token manager in order to evaluate user names and passwords. This allows you to override the token authentication as appropriate to your service.

Right-click the **App_Code** node in the Visual Studio Solution Explorer, click **Add New Item**… and select a Class.

Add the following code to the implementation of the class:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using Microsoft.Web.Services3.Security.Tokens;

/// <summary>
/// Summary description for SampleUsernameTokenManager
/// </summary>
public class SampleUsernameTokenManager : UsernameTokenManager
{
  protected override string AuthenticateToken(UsernameToken token)
  {
    bool Valid = (("YOUR-DOMAIN\\Your-Account" == token.Username));
    if (Valid)
    {
      return token.Password;
    }
    else
    {
      throw new UnauthorizedAccessException("User validation failed");
    }
  }
}
```

The Web Service's Web.config file controls which token manager is used. Add the following items to the bottom of the Web.config for the service:

```
  <microsoft.web.services3>
    <security>
      <securityTokenManager>
        <add type="SampleUsernameTokenManager, App_Code"
namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" localName="UsernameToken" />
      </securityTokenManager>
    </security>
    <policy fileName="wse3policyCache.config"/>

  </microsoft.web.services3>
</configuration>
```

Compile the project to check that it still builds correctly.

More Web.config Tune-ups

To help make sure the Web Service WSDL is as easily understood as possible, the version of SOAP is restricted to SOAP 1.1. Other WSE 3.0 extensions need to be added as well. Make sure the top of Web.config has these lines of XML:

```
<?xml version="1.0"?>
<configuration>
  <configSections>
  <section name="microsoft.web.services3"
type="Microsoft.Web.Services3.Configuration.WebServicesConfiguration,
Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
  </configSections>
  <system.web>
    <webServices>
      <soapExtensionImporterTypes>
        <add type="Microsoft.Web.Services3.Description.WseExtensionImporter,
Microsoft.Web.Services3, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
      </soapExtensionImporterTypes>
      <soapServerProtocolFactory
type="Microsoft.Web.Services3.WseProtocolFactory, Microsoft.Web.Services3,
Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
      <protocols>
        <remove name="HttpSoap12"/>
      </protocols>
    </webServices>
```

If you run the project at this point and inspect the WSDL, you should see the SOAP 1.2 namespace declared at the top but no use of the namespace (using the soap12 prefix) within the file.

```
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
```

The Web Service itself is now ready to go. You can leave it running.

Axis 2 and Rampart WS-Security Components

Crystal Reports 2008's XML and Web Services data driver is implemented using Java. It uses the Apache Axis 2 Web Services stack and the Apache Rampart extensions to support WS-Security.

A key requirement for Crystal Reports data access is that no prompts go to the user. This is necessary for any type of server deployment in custom Web applications or management in Business Objects Enterprise.

In order to pass the user ID and password without prompting, a small class is required to be built that passes the user ID and password to the Rampart WS-Security component for authentication between client and server on your user(s) behalf. The sample here uses a password call-back handler class.

Save the following code to a file named WSE_PWCBHandler.java:

```
package org.apache.rampart.samples.test;

import java.io.IOException;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.apache.ws.security.WSPasswordCallback;

public class WSE_PWCBHandler implements CallbackHandler {

    public void handle(Callback[] callbacks) throws IOException,
            UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            //When the server side need to authenticate the user
            WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];
            if(pwcb.getIdentifer().equals("YOUR-DOMAIN\\Your-Account")) {
                pwcb.setPassword("Your-Password"); // nice for testing but
you should create a better architecture if you want to support many users.
                return;
            } else {
                throw new UnsupportedCallbackException(callbacks[i], "check
failed");
            }
        }
    }
}
```

Be sure to change the account name and password listed in the code. It is expected that a 'real' implementation would have a more appropriate mechanism in place for managing user names and password retrieval.

In order to compile this code, you need the Java 1.5 SDK. Additionally, the Apache Web Service Security library, wss4j-1.5.1.jar, needs to be in the CLASSPATH for the Java compiler to use. Crystal Reports 2008 comes with this library and also comes with the required Java SDK.

The Java SDK is, by default, installed here: C:\Program Files\Business Objects\javasdk

The wss4j-1.5.1.jar file is located here by default:

C:\Program Files\Business Objects\Common\4.0\java\lib\external\wss4j-1.5.1.jar

Add that file to the Windows CLASSPATH environment variable, and then run the following at the command line:

C:\Program Files\Business Objects\javasdk\bin\javac WSE_PWCBHandler.java

This should produce a file called WSE_PWCBHandler.class. Note the location of this file. It will be copied to a new location in the following section.

WS-Security Policy File

The Apache Rampart components need to be configured to know what type of security to use and where to find the password call-back handler class compiled above. This is the client-side policy configuration for the Web Service. The Crystal Reports XML and Web Services driver is provided the path to this policy file.

The following is the contents of the policy file, which you can save with the name wse_policy.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy wsu:Id="UTOverTransport" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <wsp:ExactlyOne>
        <wsp:All>
            <sp:TransportBinding
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
              <wsp:Policy>
                  <sp:TransportToken>
                    <wsp:Policy>
```

```
                          <sp:HttpsToken RequireClientCertificate="false"/>
                      </wsp:Policy>
                    </sp:TransportToken>
                    <sp:AlgorithmSuite>
                      <wsp:Policy>
                          <sp:Basic256/>
                      </wsp:Policy>
                    </sp:AlgorithmSuite>
                    <sp:Layout>
                      <wsp:Policy>
                          <sp:Lax/>
                      </wsp:Policy>
                    </sp:Layout>
                    <sp:IncludeTimestamp/>
                </wsp:Policy>
              </sp:TransportBinding>
              <sp:SignedSupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
                    <wsp:Policy>
                          <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/Include
Token/AlwaysToRecipient" />
                </wsp:Policy>
              </sp:SignedSupportingTokens>

              <ramp:RampartConfig
xmlns:ramp="http://ws.apache.org/rampart/policy">
                    <ramp:user>YOUR-DOMAIN\Your-Account</ramp:user>

      <ramp:passwordCallbackClass>org.apache.rampart.samples.test.WSE_PWCBHan
dler</ramp:passwordCallbackClass>

      </ramp:RampartConfig>
        </wsp:All>
      </wsp:ExactlyOne>
</wsp:Policy>
```

To follow along easily, save the policy file to C:\Users\Public or some other public folder on the machine.

Be sure to change the `<ramp:user>` entry in the file to include your domain and user account.

Notice the `<ramp:passwordCallbackClass>` item. It identifies the password handler component for Rampart, and also forms the file system path appended to the Crystal Reports Java CLASSPATH (located in CRConfig.xml) in which to look for the password handler. Assuming you've built the password handler component as above and want to place it in the C:\Users\Public folder on the machine, the following actions are required:

Create the folder structure to create the password handler with the following path:

C:\Users\Public\WSE_PWCBHandler\org\apache\rampart\samples\test\WSE_PWCBHandler.class

Copy the **WSE_PWCBHandler.class** file to the folder.

The folder path, C:\Users\Public\WSE_PWCBHandler\, should be appended to the Crystal Reports CLASSPATH in the Crystal Reports Java configuration file, located by default here:

C:\Program Files\Business Objects\Common\4.0\java\CRConfig.xml

The Classpath item should end like this:

```
;C:\Users\Public\
WSE_PWCBHandler\;${CLASSPATH}</Classpath>
```

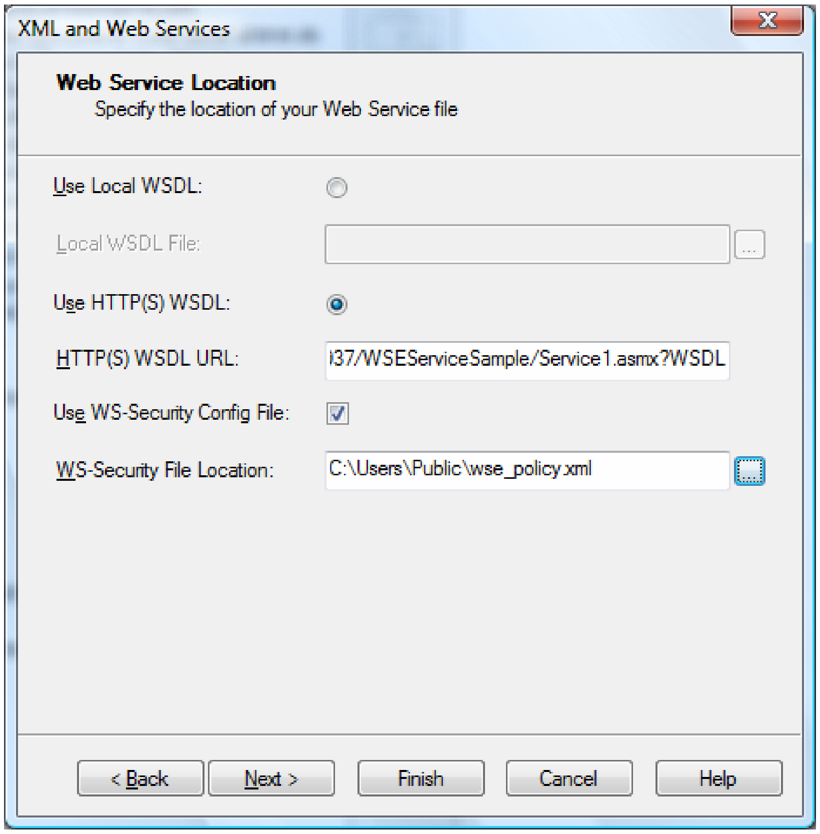
Note: in order to edit CRConfig.xml on Windows Vista or Windows Server 2008, you need to start your editor to run as Administrator. Right-click on the editor's shortcut, such as Visual Studio 2005, and click Run as Administrator.

Optional: You can eliminate the need to create a deep folder structure for the password call-back handler by eliminating part or all of the corresponding namespace declaration in the `<ramp:passwordCallbackClass>` element.

Create a Report

The final step is to create a Crystal report. In this step, you'll pass the policy file to the XML and Web Services data driver so that it can invoke the Rampart components correctly.

Follow the steps outlined above to create a report using the XML and Web Services data driver, only this time enter the path to the WS-Security policy configuration file as shown here:


XML and Web Services

Web Service Location

Specify the location of your Web Service file

Use Local WSDL: ○

Local WSDL File:

Use HTTP(S) WSDL: ●

HTTP(S) WSDL URL: 37/WSEServiceSample/Service1.asmx?WSDL

Use WS-Security Config File: ✓

WS-Security File Location: C:\Users\Public\wse_policy.xml

The report should refresh and display data as expected.

Conclusion

This document has demonstrated how to create a WS-Security enabled Web Service and then write a Crystal report that can consume it. The various ways in which Crystal reports can be deployed and the types of security supported by WS-Security combine to make the complexity of the connectivity problem non-trivial. However, the flexibility offered should allow you to connect to a WS-Security enabled Web Service employing almost any type of authentication.

Additional Notes

At the time of the Crystal Reports 2008 Service Pack 0 release, SOAP 1.2 was not supported. This limits connectivity to WS-Security enabled Microsoft Web Service stacks to ASP.NET (ASMX with WSE 3.0). Microsoft's Windows Communication Foundation (WCF) WS-Security implementation (the WSHttpBinding) requires SOAP 1.2. Future Crystal Reports 2008 service packs are likely to add support for SOAP 1.2. Refer to the release notes included with future Crystal Reports service packs and products.