ECOLLEGE

Secured eCollege Web Services

Working with Web Services Security

VERSION 1.0

Revision History

Date	Author	Document Version	Sections Edited	Notes
12/12/2007	Chris Hatton	1.0	All	Initial Version

Introduction

Definition

This document is intended for development and IT staff for the purpose of adding the appropriate security header information to SOAP requests destined for eCollege web services. Resources and code samples are provided in a number of different programming languages, including: .Net, Java, and PHP.

Overview

eCollege leverages the Oasis Web Services Security Username Token Profile 1.0 for securing all web services. This entails some additions to the SOAP header that are used to authenticate the requestor, specifically a Username and PasswordDigest. This authentication means is intended to support the broadest array of client-side technologies by leveraging an open and vendor-neutral standard.

Authenticating SOAP Requests

All interaction with the secured eCollege web services will take place using SOAP 1.1 via the HTTP protocol. As most clients will consume these services via a SOAP toolkit specific to their platform, the specifics of calling the services will depend upon the documentation provided with your specific toolkit.

Every web service request to a secured eCollege web service will require authentication information that identifies the client. The services will utilize this information to both authenticate the client, and authorize their requested action to the specific Educational Partners campus data. As mentioned previously, these web services will leverage the WS-Security Username Token in the SOAP header for authentication.

The Username Token specification requires that the following information be inserted into the SOAP request header for the purposes of identifying the requester. According to the OASIS Web Services Security Username Token Profile and its standard usage for eCollege web services, the SOAP header requires the following elements:

Parameter	Description
`Username`	This element identifies the Username of the third-party system that is making the web service request. This parameter is assigned by eCollege to the partner.
`Password`	The Password element is also assigned by eCollege. However, eCollege utilizes a PasswordDigest and not a plaintext password. The specification outlines the procedure for appropriately hashing the Password. eCollege does not currently support plaintext passwords.
`Nonce`	The Nonce is a random value that is used by the framework to ensure against replay attacks. This value should be automatically supplied by the WS-Security toolkit. Note that not all third-party implementations include a Nonce in the Username Token by default. However, *eCollege requires this element for purposes of replay detection*. Consult your specific frameworks documentation should you discover that the Nonce is not being included.
`Created`	Indicates the date and time that a particular request was generated. This value should be automatically inserted by the toolkit in a pre-defined format.
`Timestamp`	The Timestamp element should be provided by the WS-Security toolkit for the purposes of ensuring that a message can expire.

The following XML snippet is an example SOAP header containing a UsernameToken from a third-party system identified as *PublicUniversity*. Note the usage of the Password Digest option to avoid including the password as raw text.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Header>
                <wsse:Security soapenv:mustUnderstand="1">
                        <wsse:UsernameToken>
                                <wsse:Username>PublicUniversity</wsse:Username>
                                <wsse:Password
                                        Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-username-token-profile-1.0#PasswordDigest">
                                        JikHwAalkuG9uwJJ/eNobRlePoY=</wsse:Password>
                                <wsse:Nonce>M3LjJFjoaD6Bl4aSLS5Okw==</wsse:Nonce>
                                <wsu:Created>2007-08-30T19:45:29Z</wsu:Created>
                        </wsse:UsernameToken>
                        <wsu:Timestamp>
                                <wsu:Created>2007-08-30T19:45:29Z</wsu:Created>
                                <wsu:Expires>2007-08-30T19:50:29Z</wsu:Expires>
                        </wsu:Timestamp>
                </wsse:Security>
        </soapenv:Header>
        <soapenv:Body>
                ...
        </soapenv:Body>
</soapenv:Envelope>
```

There are several freely available WS-Security implementations available. See Appendix A for a list of resources on WS-Security.

Securing the Payload

All interaction with eCollege web services will be encrypted at the Transport Layer with SSL. This allows the hardware to assist in the encryption/decryption process and prevents requiring software developers with concerns over ensuring message integrity. At this time, eCollege does not support payloads encrypted via means other than SSL.

Sample Client Code

.Net Framework

The following code sample uses the Microsoft WSE 2.0 framework for all WS-Security related functionality. In order to have Visual Studio.Net automatically generate a WS-enabled proxy class, make sure that your project is enabled for Web Services Enhancements.

```
using Microsoft.Web.Services2.Security.Tokens;

…

// The following three values are all dependent on your specific project.
string url = "https://custom.ws.ecollege.com";
string webServicesUser = "PublicUniversity";
string webServicesPassword = "shhh";

// A similar class is created by you, using Visual Studio.Net or wsdl.exe using the WSDL
// provided by eCollege for your respective service
SecureEcollegeServiceWse secureProxy = new SecureEcollegeServiceWse();
secureProxy.Url = url;

// Create Username Token with the required elements
UsernameToken unToken = new UsernameToken(webServicesUser, webServicesPassword,
   PasswordOption.SendHashed);

secureProxy.RequestSoapContext.Security.Tokens.Add(unToken);

// Call your method
// Something something = secureProxy.DoSomething();
```


Java & wss4j

The following sample code uses wss4j and the Web Services Deployment Descriptor (wsdd) syntax for adding the appropriate SOAP header information.

```
// Point to wsdd file for run-time security binding
String wsddFilePath = "./config/usernametoken-deploy.wsdd";
EngineConfiguration config = new FileProvider(wsddFilePath);

//      Get the proxy
CourseEnrollmentServiceLocator locator =
       new CourseEnrollmentServiceLocator(config);

locator.setCourseEnrollmentServiceSoapEndpointAddress(this.webServiceUrl);
```

The code above points to a file named “usernametoken-deploy.wsdd” which instructs wss4j on the specific means that it should take to provide the appropriate security information. The following sample XML snippet from such a file, instructs wss4j to include a *Timestamp* and *UsernameToken* with a password type of *PasswordDigest*.

```
  <globalConfiguration>
   <requestFlow>
    <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
        <parameter name="action" value="Timestamp UsernameToken"/>
     <parameter name="user" value="PublicUniversity"/>
     <parameter name="passwordCallbackClass"
       value="com.eCollege.api.tc.common.UsernameCallbackHandler"/>
     <parameter name="passwordType" value="PasswordDigest"/>
     <parameter name="precisionInMilliseconds" value="false"/>
     <parameter name="signaturePropFile" value="wss-usernamesign.properties"/>
    </handler>
   </requestFlow>
  </globalConfiguration>
```

Note: in some cases, eCollege web services suffer from a known Microsoft compatibility factor that requires that timestamp precision does not include milliseconds (specified in the wsdd sample above as the *precisionInMilliseconds* parameter). Please consult your respective web service documentation to determine whether or not this is applicable to you.

The last piece of required code is identified in the .wsdd as the “passwordCallbackClass”. This class implements javax.security.auth.callback.CallbackHandler in order to insert the appropriate password, like this:

```
import java.io.*;
import java.util.*;
import javax.security.auth.callback.*;
import org.apache.ws.security.*;

public class UsernameCallbackHandler implements CallbackHandler {

   public void handle(Callback[] callbacks) throws IOException,
     UnsupportedCallbackException {

        for (int i = 0; i < callbacks.length; i++) {
             if (callbacks[i] instanceof WSPasswordCallback) {
                 WSPasswordCallback pc = (WSPasswordCallback)callbacks[i];

                 // Insert the password here
                 if (pc.getIdentifier().equalsIgnoreCase(“PublicUniversity”)) {
                        pc.setPassword(“shhh”));
                 }

             }
              else {
                 throw new UnsupportedCallbackException(callbacks[i], "Unrecognized Callback");
              }
         }
     }
…
```


PHP

In order to develop PHP SOAP clients, the following 2 libraries are required:

- PHP 5 or greater
- PHP SOAP Extension

Additionally, you will need a library that can perform the WS-Security Username Token Profile for you. The following sample references 2 files provided by the University of Toronto: soap -wsse.php and xmlseclibs.php. Alternatively, you could use an open source framework like WSO2 Web Services Framework for PHP.

```
<?php
require('soap-wsse.php');

class mySoap extends SoapClient {

   function __doRequest($request, $location, $saction, $version) {

   echo $location . "<br>" ;

    $doc = new DOMDocument('1.0');
    $doc->loadXML($request);

    $objWSSE = new WSSESoap($doc);

    $wsUser = 'PublicUniversity';
    $wsPassword = 'shhh';

    $objWSSE->addUserToken($wsUser, $wsPassword, TRUE);

    echo htmlspecialchars($objWSSE->saveXML()) . "<br>\n";

    return parent::__doRequest($objWSSE->saveXML(), $location, $saction, $version);
   }
}


//Use your specific URL
$wsdl = 'http://api.ecollege.com';

$sClient = new mySoap($wsdl, array('trace'=>1));

// Add specific information to call the web service
//$wrapper->something->someParameter = new SoapVar("This is something", XSD_STRING);

$result = $sClient->DoSomething(new SoapParam($wrapper,"something"));

echo htmlspecialchars($sClient->__getLastRequest()) .
    "<br><br>\n" .
    htmlspecialchars($sClient->__getLastResponse());

//var_dump($result);

?>
```


Appendix A—WS-Security Resources

At the original time of the authoring of this document, the following online resources were helpful in beginning to work with the WS-Security specification.

General

- Introduction to Web Services Security
- OASIS WS-Security Specification
- Web Services Security UsernameToken Profile 1.0 (PDF)

.Net Framework

- Microsoft Web Services Enhancements (WSE)
- Securing .NET Web Services with the WS-Security Protocol

Java

- Apache WSS4J
- Implementing WS-Security with Java and WSS4J

PHP

- Calling secured Web Service methods from PHP—*IBM Developer Works*
- PHP 5 Extension
- WSO2 Web Services Framework for PHP

