Calling secured Web services methods from PHP
Processing PHP structured parameters into SOAP arguments

Skill Level: Intermediate

Vladimir E. Shcherbina (shcherb@il.ibm.com)
Research Staff Member
IBM

Yoav Ossia (YOSSIA@il.ibm.com)
Research Staff Member
IBM

05 May 2006

Learn how to satisfy the requirements for security and method definition in PHP:Hypertext Processor (PHP) scripts which implement Web Services clients. Using the new SOAP extension in PHP 5, you'll see how to implement WS-Security basic authentication and how to pass complex objects as parameters for SOAP calls.

Section 1. Before you start

Objectives

This tutorial will provide you:

- An overview of SOAP in PHP
- A demonstration of how to create security headers with an actual implementation example
- A description of complex structures that may be passed as parameters to SOAP methods and how to transform the PHP representation of such parameters into the required format of a SOAP call argument
Prerequisites

This tutorial assumes basic knowledge of the PHP5 scripting language. For installation of PHP and PHP SOAP extension, please refer to the article "Access an enterprise application from a PHP script", listed in the Resources.

System requirements

To run the examples in this tutorial, you need:

- Any Web browser, preferably MS IE version 4.0 or higher or Mozilla Firefox version 1.0 or higher
- Front end machine with any HTTP server and PHP 5 extension installed
- Any server that support Web services interface (WSDL, SOAP over HTTP) and implement Basic WS-Security protocol

Section 2. Getting started

Open source applications and open standards are becoming the preferred method for building applications on the Web. The most effective way to create such applications is to utilize existing software components and services across the Web. A common implementation of such a tiered application includes server-side PHP scripts using Web services. With this type of architecture, there is an increased need for secured communication.

SOAP and PHP - Overview and references

Before the introduction of PHP 5, it was hard to call Web services in pure PHP. In PHP 5, the application developer has a number of options for implementing PHP Web services clients: PEAR:SOAP, NuSOAP, and the new SOAP extension. This tutorial focuses on the use of the latter.

The SOAP extension has improved capabilities over previous PHP solutions, including the SoapVar type and several OO mechanisms that can be used to construct virtually any complex SOAP type.
According to Rosenberg and Remy's 2005 book (see Resources), "WS-Security is an overarching conceptual model that abstracts different security technologies into "claims" and "tokens"...SOAP headers are used for directive information. This is essentially the place where SOAP security lives. System-level information used to manage and secure the message is placed here as well. SOAP runtimes and intermediaries process SOAP header directives. Headers are intended to add new features and functionality and WS-Security headers will be located here. A sender can require that the receiver understand the header. Headers speak directly to the SOAP processors and can require that a processor reject the entire SOAP message if it does not understand the header. If a header contains critical security information that the SOAP processor does not understand, you may not want it to process this SOAP message at all."

The developerWorks article Access an enterprise application from PHP script (see Resources) describes the usage of SOAP in PHP 5 for accessing a J2EE application using Web services. It also describes the SOAP PHP installation procedure, and relates to the security issues we are discussing here. The article states: "...there's no first-class support in ext/soap for WS-Security. Therefore, if we're going to send and receive WS-Security headers in PHP, we'll have to drop down into a more detailed interface where we can explicitly create the SOAP headers. ...You build up the message elements using the SoapHeader, SoapParam and SoapVar classes, and then use SoapClient::__call to send the SOAP request and get the response."

Creating a WS-Security header

As explained above, WS-Security works by adding security headers to the SOAP messages. The box below lists an example of the required security header for WS-Security basic authentication (for the user myUserName and the password myPass):

```
Listing 1. Security header for WS-Security basic authentication

<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <wsse:UsernameToken>
      <wsse:Username>myUserName</wsse:Username>
      <wsse:Password>myPass</wsse:Password>
    </wsse:UsernameToken>
  </wsse:Security>
</soapenv:Header>
```

The challenge here is to utilize generic SOAP extension object constructions in order to create the required headers and integrate them into the SOAP call. The instrument for this task is the SOAP extension's SoapVar data structure, which is defined in the PHP online manual (see the Resources).:
SoapVar is a special low-level class for encoding parameters and returning values in non-WSDL mode. It is just a data holder and does not have any special methods except the constructor. It’s useful when you want to set the type property in SOAP request or response. The constructor takes data to pass or return, encoding ID to encode it (SOAP_VAR_ENC in our case) and, optionally, type name, type namespace, node name and node name namespace.

The procedure for creating the required nested tag is: wrap the Username and Password simple (i.e., without nesting) tags into SoapVar. The result should then be wrapped into another SoapVar, tagged as UsernameToken, which is placed inside the <Security> tag. Finally, this tag is placed inside the SOAP header.

In this case the required wrapping operation is two levels deep. In general the approach described here may also be applied to creating complex SOAP objects of arbitrary nesting levels.

In the next section, we describe the actual implementation.

Coding basic security in PHP

The Security tag of the SOAP header is assembled using the bottom-up approach. First, the UserName and Password tags are expressed as XSD strings. Listing 2 assumes that the $username and $password variables are properly assigned:

Listing 2. Creating XSD strings from credentials

```php
$userT = new SoapVar($username, XSD_STRING, NULL, $nameSpace, NULL, $nameSpace);
$passwT = new SoapVar($password, XSD_STRING, NULL, $nameSpace, NULL, $nameSpace);
```

In order to express <UserNameToken> tag with nested <Username> and <Password> tags inside, we need to define the intermediate class with private data members: $Username and $Password. Note that while the defined class may have an arbitrary name, the data members must have the same names as the corresponding XML tags. Our implementation creates a class named UsernameT1. The data members are assigned by the constructor.

Listing 3. The UsernameT1 class definition

```php
class UsernameT1 {
    private $Username; // Name must be identical to corresponding XML tag in SOAP header
    private $Password; // Name must be identical to corresponding XML tag in SOAP header

    function __construct($username, $password) {
        $this->Username=$username;
        $this->Password=$password;
    }
}
```
Now, we can create the content of the complex XML tag `<UsernameToken>` as the SoapVar, whose type is not an XSD_STRING, but SOAP_ENC_OBJECT. In this case (unlike when creating an XSD string) the name of the created XML tag is also passed to the SoapVar constructor.

**Listing 4. Creating a UserNameT1 instance and wrapping it into SoapVar**

```php
$tmp = new UsernameT1($userT, $passwT);
$uuT = new SoapVar($tmp, SOAP_ENC_OBJECT, NULL, $nameSpace, 'UsernameToken', $nameSpace);
```

This SoapVar will be wrapped into a UserNameT2 class with the private data member `$UsernameToken`. Again, the defined class may have an arbitrary name, but the data member must have the same name as the corresponding XML tag.

**Listing 5. The UserNameT2 class definition**

```php
class UserNameT2 {
    private $UsernameToken; //Name must be identical to corresponding XML tag in SOAP header
    function __construct ($innerVal){
        $this->UsernameToken = $innerVal;
    }
}
```

A UserNameT2 instance is created and wrapped into a SoapVar:

**Listing 6. Creating a UserNameT2 instance and wrapping it into SoapVar**

```php
$tmp = new UsernameT2($uuT);
$userToken = new SoapVar($tmp, SOAP_ENC_OBJECT, NULL, $nameSpace, 'UsernameToken', $nameSpace);
```

The UsernameToken object is attached to the parent `<Security>` XML tag, using the same method again, and the SoapHeader `<Security>` is now constructed:

**Listing 7. Constricting the security header**

```php
$secHeaderValue=new SoapVar($userToken, SOAP_ENC_OBJECT, NULL, $nameSpace, 'Security', $nameSpace);
$secHeader = new SoapHeader($nameSpace, 'Security', $secHeaderValue);
```

This header will be passed to the method `__soapCall()` of the SoapClient class as an element in the `input_headers` array. For example, when the security is the only input header:

**Listing 8. Using the security header**

```php
$client->__soapCall($theMethodName, $theMethodSignature, null, $secHeader);
```
Section 3. Creating the method's signature

Calling __soapCall includes one additional task: the construction of the method's passed arguments, also known as the method's signature. In the case of a simple list of parameters, this is a trivial task which is documented in the PHP Online Manual (see the Resources). This section deals with the more complex case, where the Web service's method includes arrays and object parameters and these parameters may also have members that are themselves arrays or objects.

As with the security header, we need to create a nested tag by using the SOAP extension's SoapVar data structure. This is needed for each parameter that is an object, and for each object member of any passed array or object.

The next section introduces a few methods that are capable of packing a complex set of parameters into a valid __soapCall arguments parameter.

Coding a method signature in PHP

This section lists code for packing PHP parameters into a SOAP call. In order to do that, a generic definition of input is required. We have defined the input as a PHP array of parameters. The array may include simple types, objects, arrays, nested objects, and nested arrays.

For example, consider the case of a Web services method storeProblemReport() that gets three parameters: severity (integer), errors (an array of strings), and owner (a record with a member that is also a record). The WSDL part of such a method may contain the following lines:

Listing 9. Example method and parameters - WSDL definition

```xml
<xsd:complexType name="fileInfo">
  <xsd:sequence>
    <xsd:element name="fname" type="xsd:string"/>
    <xsd:element name="line" type="xsd:int"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ownerInfo">
  <xsd:sequence>
    <xsd:element name="component" type="xsd:string"/>
    <xsd:element name="location" type="intf:fileInfo"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="storeProblemReport">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="severity" type="xsd:int"/>
      <!-- other parameters -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```
An example of possible values and a value input may be:

**Listing 10. Assigning the example parameters**

```php
$fileInfo->fname = '/usr/src/myDir/getToken.php'
$fileInfo->line = 7;
$theOwner->component = 'Parser';
$theOwner->location = $fileInfo;
$argumentsList = array ('severity' => 7,
                        'errors' => array ('empty token', "read error", "File open error"),
                        'owner' => $theOwner));
```

Creating the required arguments for `soapCall` is done by the `CreateMethodSignature()` function below:

**Listing 11. Creating the required arguments**

```php
function createMethodSignature($theMethod, $paramAr) {
    if (null == $paramAr)
        return array($theMethod =>null);
    $used = null;
    foreach ($paramAr as $name => $value) {
        if (is_array($value) || is_object($value)) {
            $used[$name] = createMixedValueSignature($value);
        } else {
            $used[$name] = $value;
        }
    }
    return array($theMethod =>$used);
}
```
Note that arrays of simple data types or arrays (arrays that do not include objects) need no special treatment. The only reason for processing arrays with the recursive function `createMixedValueSignature()` is the possibility of inner PHP objects. `CreateMethodSignature()`, with the above input, is used for producing a valid SOAP argument:

**Listing 12. Calling the example method**

```php
$theMethodSignature = CreateMethodSignature('storeProblemReport', $argumentsList);
$client->__soapCall('storeProblemReport', $theMethodSignature, null, $secHeader);
```

**Summary**

This tutorial has shown how to implement a WS-Security basic authentication scheme and how to pass complex parameters, using the SOAP extension of the PHP 5. We've covered only a small subset of the WS-Security specification. There are more complex -- and more serious -- authentication schemes, for example X.509, and many other elements in WS-Security, such as XML Encryption and XML Signatures, that one should implement in order to provide full scale WS-Security functionality. As for the processing of PHP structured parameter into SOAP arguments, the example here is generic and intended for a list of complex parameters. This list may include nested objects and arrays. This example may be easily modified to fit any specific and simpler use.
Resources

Learn

• The PHP Online Manual provides reference information about SOAP APIs in PHP 5.


• The article Access an enterprise application from a PHP script (developerWorks, February 2005) describes how to use the SOAP extension in PHP 5 to access a J2EE application using Web services. It also describes the SOAP PHP installation procedure, and relates to the security issues we are discussing here.

• Visit the SOA and Web services zone to expand your webservices skills.

• Stay current with developerWorks technical events and Webcasts.

Get products and technologies

• Build your next development project with IBM trial software, available for download directly from developerWorks.

Discuss

• Participate in developerWorks blogs and get involved in the developerWorks community.

About the authors

Vladimir E. Shcherbina
Vladimir Shcherbina is a research staff member at IBM Haifa Research Lab. He studied Informational Systems and earned a master's degree from the Minsk Radio Technical Institute. He joined IBM in 1992 and has worked in areas ranging from GUI to Business Process Management. Since 2000, he has been involved in the development of AMiT (Complex Event Processing (CEP) engine) and CEP based solutions. A main focus of his work has been the Event Driven Architecture (EDA), evaluation of customer architectural solutions in IBM IGS engagements and PHP integration with WS stack. Currently, he is working on Event Based Solutions.

Yoav Ossia
Yoav Ossia is a research staff member at IBM Haifa Research Lab. He has more than 15 years of software development experience, and has been working in IBM for the last six years. In IBM he worked on automatic memory management in the IBM JVM, and is recently working on infrastructure for Software as Services (SaS) applications.