OWASP Encoding Project

The Reform Library
Implementation Guide

Michael Eddington
(meddington@phed.org)

Revision History

Date	*Author*	*Comment*
04/11/04	Michael Eddington	Initial document creation
06/30/04	Michael Eddington	Added in ASP section
07/01/04	Michael Eddington	Added in Java section
04/28/05	Michael Eddington	Adding .NET/COM version
12/21/05	Michael Eddington	Misc updates
11/05/06	Michael Eddington	Clarify returns from JsString and VbsString
11/12/06	Michael Eddington	Namespace change to Owasp

Table of Contents

1 The Problem

Web applications face any number of threats; one of them is cross-site scripting and related injection attacks. 90% of all web applications contain cross-site scripting attacks because they are easy to introduce, and time consuming (but not always hard) to prevent. In addition, there is no good single library that provides all the functions required by developers to incorporate a fix into there code that will stand up to the test of time and continual research in the field. The Reform library attempts to provide a solid set of functions for encoding output for the most common context targets in web applications (e.g. HTML, XML, JavaScript, etc). The library also attempts to take a conservative view of what are allowable characters based on historical vulnerabilities, and current injection techniques.

1.1 Support Options and Reporting Bugs

This is free software and as such there is no official support line. However, you may try and bug the author if all else fails. Additionally, I am always eager to hear feed back, or even a quick note to say you are successfully using Reform. Bug reports are especially good to get so Reform can grow into a more useful library.

If interest is sufficient a mailing list or forum may be created to ask questions and possible notify people of any updates.

Email address: meddington@phed.org

2 Reform Specs

The Reform library will encode any character that does not fall into a normal alphabet of characters. In addition *all* Unicode characters are encoded.

2.1 Unicode

The introduction of internationalized application (both web and not) which take advantage of the widely accepted Unicode standard generally use UCS-2 or UTF-8 encoding standards. Due to implementation problems, and problems in the Unicode specification, it is not possible to predict how malformed characters will be interpreted by differing Unicode implementations, or versions of the implementation. This has created large vulnerabilities in the past. Because of this the Reform library takes the safe path of always encoding Unicode characters so they will not (in theory) cause injection type attacks.

2.2 Importing Methods

Unless noted all implementations of the Reform library are done by implementing a class Reform in the Owasp namespace. In addition all methods are implemented as "static" if such a concept exists in the language.

2.2.1 Perl

Implemented as Owasp::Reform. Usage should follow standard non-class package method calling. For example:

```
Owasp::Reform::HtmlEncode("mystring", "default")
```

In addition, methods can be imported into current namespace using the following code:

```
use Owasp::Reform qw(HtmlEncode HtmlAttributeEncode);.
```

Additional methods can be imported in the same manor.

2.2.2 Python

Reform is implemented as a Python class, and also a set of functions. Depending on the form/import used you can import all the functions into the current namespace or use the class interface.

2.2.3 PHP

The PHP implementation is a class called `Reform` with all member functions marked as `static`. Class can be imported using a standard include directive:

```
<?php include('Reform.php'); ?>
```

Accessing member functions using standard static calling syntax of:

```
Reform::HtmlEncode( … );.
```

2.2.4 Javascript

Currently the Javascript implementation does not make use of classes, methods are included into the current namespace. If requested a future version may include a class version to provide for a clean namespace separation. The Javascript functions can be imported using a standard script include line such as:

```
<script src="Reform.js"></script>.
```

2.2.5 ASP

The Reform library is available as a standard ASP include file. The include file declares an instance of the Reform class as the `reform` variable. Reform can be included into any ASP page using the following line:

```
<!-- #include file="Reform.inc.asp" -->
```

Access to the Reform methods is done ala:

```
<%= Reform.HtmlEncode( Request("Foo") ) %>
```

2.2.6 Java

To use the Java version of Reform you will need to include org.owasp.reform namespace. Access to Reform is done through a static class called Reform. Here is an example call:

```
import org.owasp.reform.Reform;

Reform.HtmlEncode("<script> alert('meow') </script>");
```

2.2.7 .NET

The .NET version of Reform is available as both web controls and as an API exposed via the Reform class. The namespace is Owasp.Reform. All of the Reform class methods are marked as static, and are best called directly. Here is an example call:

```
use Owasp.Reform;

…

string s = Reform.HtmlEncode("<sript> alert('meow') </sript>");
```

3 API Reference

The following are the core functions provided by the Reform library. The functions are defined and operate the same between all implementations unless noted.

3.1 HtmlEncode

The HtmlEncode function takes in a string and performs HTML encoding using the &#NN; style encoding where NN is the decimal number of the character. The encoding style is the same for both Unicode and non Unicode characters.

```
encodedString = HtmlEncode ( string, default = '' )
```

3.1.1 Parameters

string

[in] String to encode. If *string* is null then *default* is used.

default

[in, optional] A default value to use (will be encoded) if *string* is null. If *default* is not specified then an empty string will be used instead.

3.1.2 Return Values

Returns an HTML encoded string. Encoded characters are encoded using the &#NN; style of encoding where NN is the decimal number of the encoded character.

3.1.3 Example Code

Sample PHP Page

```
<?php
include('Reform.php');
$reform = new Reform;
php>
<body>
Hello, your name is <?php echo $reform->HtmlEncode(
"<script>alert('meow')</script>"); ?>.
</body>
```

Output from Sample PHP Page

```
<body>
Hello, your name is
&#60;script&#62;alert&#40;&#39;meow&#39;&#41;&#60&#47;script&#62;.
</body>
```

3.2 HtmlAttributeEncode

The HtmlAttributeEncode function takes in a string and performs HTML encoding using the &#NN; style encoding where NN is the decimal number of the character. The encoding style is the same for both Unicode and non Unicode characters.

The HtmlAttributeEncode function differs from HtmlEncode function due to a more restrictive encoding policy which includes the SPACE character. This is to prevent vulnerabilities caused by failure to wrap the HTML attribute in quotation marks.

```
encodedString = HtmlAttributeEncode ( string, default = '' )
```

3.2.1 Parameters

string

[in] String to encode. If *string* is null then *default* is used.

default

[in, optional] A default value to use (will be encoded) if *string* is null. If *default* is not specified then an empty string will be used instead.

3.2.2 Return Values

Returns an HTML encoded string. Encoded characters are encoded using the &#NN; style of encoding where NN is the decimal number of the encoded character.

3.2.3 Example Code

Sample PHP Page

```
<?php
include('Reform.php');
$reform = new Reform;
php>
<body>
<input type=hidden value="<?php echo $reform->HtmlEncode(
"<script>alert('meow')</script>"); ?>">
</body>
```

Output from Sample PHP Page

```
<body>
<input type=hidden
value="&#60;script&#62;alert&#40;&#39;meow&#39;&#41;&#60&#47;script&#62
;">
</body>
```

3.3 XmlEncode

The XmlEncode function takes in a string and performs XML encoding using the &#NN; style encoding where NN is the decimal number of the character. The encoding style is the same for both Unicode and non Unicode characters.

```
encodedString = XmlEncode ( string, default = '' )
```

3.3.1 Parameters

string

[in] String to encode. If *string* is null then *default* is used.

default

[in, optional] A default value to use (will be encoded) if *string* is null. If *default* is not specified then an empty string will be used instead.

3.3.2 Return Values

Returns an XML encoded string. Encoded characters are encoded using the &#NN; style of encoding where NN is the decimal number of the encoded character.

3.4 XmlAttributeEncode

The XmlAttributeEncode function takes in a string and performs XML encoding using the &#NN; style encoding where NN is the decimal number of the character. The encoding style is the same for both Unicode and non Unicode characters.

The XmlAttributeEncode function differs from XmlEncode function due to a more restrictive encoding policy which includes the SPACE character. This is to prevent vulnerabilities caused by failure to wrap the XML attribute in quotation marks.

```
encodedString = XmlAttributeEncode ( string, default = '' )
```

3.4.1 Parameters

string

[in] String to encode. If *string* is null then *default* is used.

default

[in, optional] A default value to use (will be encoded) if *string* is null. If *default* is not specified then an empty string will be used instead.

3.4.2 Return Values

Returns an XML encoded string. Encoded characters are encoded using the &#NN; style of encoding where NN is the decimal number of the encoded character.

3.5 JsString

The JsString function creates a literal JavaScript string. This means the output from this function is actually wrapped in quotation marks. Encoded characters are encoded using the \xNN and \uNNNN syntax where the 'u' indicates a UCS-2 encoded Unicode character and NN/NNNN is the hex number of the encoded character.

```
encodedString = JsString ( string, default = '' )
```

3.5.1 Parameters

string

[in] String to encode. If *string* is null then *default* is used.

default

[in, optional] A default value to use (will be encoded) if *string* is null. If *default* is not specified then an empty string literal ('') will be used instead.

3.5.2 Return Values

Returns a JavaScript string literal with encoded characters. For example the input of `Hello World` would produce the output of `'Hello World'` (including the apostrophes).

3.5.3 Example Code

Sample PHP Page

```
<?php
include('Reform.php');
$reform = new Reform;
php>
<body>
<script language="JavaScript">
var myString = <?php echo $reform->JsString("breaking out';
alert('meow');var foo='"); ?>;
</script>
</body>
```

Output from Sample PHP Page

```
<body>
<script language="JavaScript">
var myString = 'breaking out\x27\x3B alert\x40\x27meow\x27\x41\x3Bvar
foo=\x27';
</script>
</body>
```

3.6 VbsString

The VbsString function returns a value that can be treated as a string literal in VBScript. In addition, characters are encoded using the wchr function which is able to handle both Unicode and non Unicode characters. Encoding VBScript strings is not as straight forward as with others since VBScript provides no way to actually escape characters from inside of the string (see 3.6.3 for more information).

```
encodedString = VbsString ( string, default = '' )
```

3.6.1 Parameters

string

[in] String to encode. If *string* is null then *default* is used.

default

[in, optional] A default value to use (will be encoded) if *string* is null. If *default* is not specified then an empty string literal ("") will be used instead.

3.6.2 Return Values

Returns a VBScript string literal. Encoded characters are encoded using the wchr function. For example, input of `Hello World` would result in `"Hello World"`.

3.6.3 Remarks

The VBScript language provides no method for escaping values inside of strings with the exception of the quotation mark ("). The only way then to encode values for VBScript is to close out the string and call the wchr function. This results in a much longer and harder to read string, but such in the limitations of VBScript.

3.6.4 Example Code

Sample PHP Page

```
<?php
include('Reform.php');
$reform = new Reform;
php>
<body>
<script language=”VBScript”>
dim myString = <?php echo $reform->VbsString(“Hello <> World”); ?>
</script>
</body>
```

Output from Sample PHP Page

```
<body>
<script language=”VBString”>
dim myString = “Hello “&wchr(60)&wchr(62)&” World”
</script>
</body>
```

4 Common Problems and Solutions

This section lists common usages of the Reform library to solve injection related issues.

4.1 Creating HTML from Client Side Script

```
<script>
// .. parse some XML or something
// .. usercontroled data in userData

document.write( '<form><input type=hidden name=foo value="' +
HtmlAttributeEncode( userData ) + '">');

</script>
```

4.2 Creating Javascript from Server Side Code

```
<?php echo "<script> var myString = " + $reform->JsString( … ) + "; …
</script> "; ?>
```

4.3 Accessing Form Variables Based on User Input from Server Side Code

Original Vulnerable Code:

```
<script>
document.<?php echo form ?>.<?php echo element ?>.value = '<?php echo
str ?>';
</script>
```

Fixed up Code:

```
<script>
document.forms[<?php echo $reform->JsString( form ); ?>].elements[<?php
echo $reform->JsString( element ); ?>].value = <?php echo $reform-
>JsString( str ); ?>;
</script>
```

5 APPENDIX A – ASP.NET Web Controls

In addition to making the standard Reform API available to the .NET framework the stock web controls have been extended to perform proper encodings as needed. They can be used via the web designer by adding the Owasp.Reform assembly into your toolbox in Visual Studio .NET.