Software Engineering Institute
Carnegie Mellon University

Understanding Malicious Content Mitigation for Web Developers

CERT Division

http://www.sei.cmu.edu

REV-03.18.2016.0

Table of Contents

1 Problem Summary 1

2 Mitigation Summary 2

3 Explicitly Setting the Character Encoding 3

4 ISO 8859-1 (Latin-1) Character Set 8

1 Problem Summary

Web pages contain both text and HTML markup that is generated by the server and interpreted by the client browser. Servers that generate static pages have full control over how the client will interpret the pages sent by the server. However, servers that generate dynamic pages do not have complete control over how their output is interpreted by the client. The heart of the issue is that if untrusted content can be introduced into a dynamic page, neither the server nor the client has enough information to recognize that this has happened and take protective actions.

In HTML, to distinguish text from markup, some characters are treated specially. The grammar of HTML determines the significance of "special" characters -- different characters are special at different points in the document. For example, the less-than sign "<" typically indicates the beginning of an HTML tag. Tags can either affect the formatting of the page or introduce a program that the browser executes (e.g., the <SCRIPT> tag introduces code from a variety of scripting languages).

Many web servers generate web pages dynamically. For example, a search engine may perform a database search and then construct a web page that contains the result of the search. Any server that creates web pages by inserting dynamic data into a template should check to make sure that the data to be inserted does not contain any special characters (e.g., "<"). If the inserted data contains special characters, the user's web browser will mistake them for HTML markup. Because HTML markup can introduce programs, the browser could interpret some data values as HTML tags or script rather than displaying them as text.

The risk of a web server not doing a check for special characters in dynamically generated web pages is that in some cases an attacker can choose the data that the web server inserts into the generated page. Then the attacker can trick the user's browser into running a program of the attacker's choice. This program will execute in the browser's security context for communicating with the *legitimate web server*, not the browser's security context for communicating with the attacker. Thus, the program will execute in an inappropriate security context with inappropriate privileges.

2 Mitigation Summary

Any data inserted into an output stream originating from a server is presented as originating from that server, even if it does not include malicious tags. Web developers must evaluate whether their sites will send untrusted data as part of an output stream.

Untrusted input can come from, but is not limited to,

- URL parameters
- Form elements
- Cookies
- Databases queries

A combination of steps must be taken to mitigate this vulnerability. These steps include

1. Explicitly setting the character set encoding for each page generated by the web server
2. Identifying special characters
3. Encoding dynamic output elements
4. Filtering specific characters in dynamic elements
5. Examine cookies

The following sections discuss details of each of these steps.

3 Explicitly Setting the Character Encoding

Many web pages leave the character encoding ("charset" parameter in HTTP) undefined. In earlier versions of HTML and HTTP, the character encoding was supposed to default to ISO-8859-1 if it wasn't defined. In fact, many browsers had a different default, so it was not possible to rely on the default being ISO-8859-1. HTML version 4 legitimizes this - if the character encoding isn't specified, any character encoding can be used.

If the web server doesn't specify which character encoding is in use, it can't tell which characters are special. Web pages with unspecified character encoding work most of the time because most character sets assign the same characters to byte values below 128. But which of the values above 128 are special? Some 16-bit character-encoding schemes have additional multi-byte representations for special characters such as "<". Some browsers recognize this alternative encoding and act on it. This is "correct" behavior, but it makes attacks using malicious scripts much harder to prevent. The server simply doesn't know which byte sequences represent the special characters.

For example, UTF-7 provides alternative encoding for "<" and ">", and several popular browsers recognize these as the start and end of a tag. This is not a bug in those browsers. If the character encoding really is UTF-7, then this is correct behavior. The problem is that it is possible to get into a situation in which the browser and the server disagree on the encoding. Web servers should set the character set, then make sure that the data they insert is free from byte sequences that are special in the specified encoding. For example:

```
<HTML>

<HEAD>

<META http-equiv="Content-Type"

content="text/html; charset=ISO-8859-1">

<TITLE>HTML SAMPLE</TITLE>

</HEAD>

<BODY>

<P>This is a sample HTML page

</BODY>

</HTML>
```

The META tag in the HEAD section of this sample HTML forces the page to use the ISO-8859-1 character set encoding.

Identifying the Special Characters

The next two steps, encoding and filtering, first require an understanding of "special characters". The HTML specification determines which characters are "special", because they have an effect on how the page is displayed. However, many web browsers try to correct common errors in HTML. As a result, they sometimes treat characters as special when, according to the specification, they aren't. In addition, the set of special characters depends on the context:

- In the content of a block-level element (in the middle of a paragraph of text)
  - "<" is special because it introduces a tag.
  - "&" is special because it introduces a character entity.
  - ">" is special because some browsers treat it as special, on the assumption that the author of the page really meant to put in an opening "<", but omitted it in error.
- Attribute values
  - In attribute values enclosed with double quotes, the double quotes are special because they mark the end of the attribute value.
  - In attribute values enclosed with single quote, the single quotes are special because they mark the end of the attribute value.
  - Attribute values without any quotes make the white-space characters such as space and tab special.
  - "&" is special when used in conjunction with some attributes because it introduces a character entity.
- In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL. When this is done, it introduces additional special characters:
  - Space, tab, and new line are special because they mark the end of the URL.
  - "&" is special because it introduces a character entity or separates CGI parameters.
  - Non-ASCII characters (that is, everything above 128 in the ISO-8859-1 encoding) aren't allowed in URLs, so they are all special here.
  - The "%" must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server-side code. The percent must be filtered if input such as "%68%65%6C%6C%6F" becomes "hello" when it appears on the web page in question.
- Within the body of a <SCRIPT> </SCRIPT>
  - The semicolon, parenthesis, curly braces, and new line should be filtered in situations where text could be inserted directly into a preexisting script tag.
- Server-side scripts
  - Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.
- Other possibilities
  - No current exploits rely on the ampersand. This character may be useful in future exploits. Conservative web page authors should filter this character out if possible.

It is important to note that individual situations may warrant including additional characters in the list of special characters. Web developers must examine their applications and determine which characters can affect their web applications.

Encoding Dynamic Output Elements

Each character in the ISO-8859-1 specification can be encoded using its numeric entry value. A complete description of the ISO-8859-1 specification can be found in the appendix of this document.

The following example uses the copyright mark in an HTML document:

```
<p>&#169 2000 Some Co., Inc.
```

The copyright character is 169 and using the &# syntax allows the author to insert encoded characters that will be interpreted by the browser.

In addition, many of the ISO-8859-1 characters include an entity name encoding. The copyright can also be done using this method:

```
<p>&copy; 2000 Some Co., Inc.
```

Encoding untrusted data has benefits over filtering untrusted data, including the preservation of visual appearance in the browser. This is important when special characters are considered acceptable.

Unfortunately, encoding all untrusted data can be resource intensive. Web developers must select a balance between encoding and the other option of data filtering.

Filtering Dynamic Content

Unfortunately, it is unclear whether there are any other characters or character combinations that can be used to expose other vulnerabilities. The recommended method is to select the set of characters that is known to be safe rather than excluding the set of characters that might be bad. For example, a form element that is expecting a person's age can be limited to the set of digits 0 through 9. There is no reason for this age element to accept any letters or other special characters. Using this positive approach of selecting the characters that are acceptable will help to reduce the ability to exploit other yet unknown vulnerabilities.

The filtering process can be done as part of the data input process, the data output process, or both. Filtering the data during the output process, just before it is rendered as part of the dynamic page, is recommended. Done correctly, this approach ensures that all dynamic content is filtered. Filtering on the input side is less effective because dynamic content can be entered into a web sites database(s) via methods other than HTTP. In this case, the web server may never see the data as part of the input process. Unless the filtering is implemented in all places where dynamic data is entered, the data elements may still be remain tainted.

Examine Cookies

One method to exploit this vulnerability involves inserting malicious content into a cookie. Web developers should carefully examine cookies that they accept and use the filtering techniques describe above to verify that they are not storing malicious content.

Sample Filtering Code

C++ Example

```
BYTE IsBadChar[] = {
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xFF,0xFF,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0xFF,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00
};
DWORD FilterBuffer(BYTE  * pString,DWORD cChLen){
        BYTE * pBad  = pString;
        BYTE * pGood = pString;
        DWORD i=0;
        if (!pString) return 0;
        for (i=0;pBad[i];i++){
                if (!IsBadChar[pBad[i]]) *pGood++ = pBad[i];
        };
        return pGood-pString;
}
```

JavaScript Example

```
function RemoveBad(InStr){
    InStr = InStr.replace(/\</g,"");
    InStr = InStr.replace(/\>/g,"");
    InStr = InStr.replace(/\"/g,"");
    InStr = InStr.replace(/\'/g,"");
    InStr = InStr.replace(/\%/g,"");
    InStr = InStr.replace(/\;/g,"");
    InStr = InStr.replace(/\(/g,"");
    InStr = InStr.replace(/\)/g,"");
    InStr = InStr.replace(/\&/g,"");
    InStr = InStr.replace(/\+/g,"");
    return InStr;
}
```

Perl Example

```
#! The first function takes the negative approach.
#! Use a list of bad characters to filter the data
sub FilterNeg {
    local( $fd ) = @_;
    $fd =~ s/[\<\>\"\'\%\;\)\(\&\+]//g;
    return( $fd ) ;
}
#! The second function takes the positive approach.
#! Use a list of good characters to filter the data
sub FilterPos {
    local( $fd ) = @_;
    $fd =~ tr/A-Za-z0-9\ //dc;
    return( $fd ) ;
}
$Data = "This is a test string&LT;script&GT;";
$Data = &FilterNeg( $Data );
print "$Data\n";
$Data = "This is a test string&LT;script&GT;";
$Data = &FilterPos( $Data );
print "$Data\n";
```

4 ISO 8859-1 (Latin-1) Character Set

Number	**Name**	**Description**	**Appearance**
&#00;-&#08;	-	Unused	-
&#09;	-	HorizontalTab	space
&#10;	-	Linefeed	space
&#11;-&#31;	-	Unused	-
&#32;	-	Space	space
&#33;	-	Exclamationmark	!
&#34;	&quot;	Quotationmark	"
&#35;	-	Numbersign	#
&#36;	-	Dollarsign	$
&#37;	-	Percentsign	%
&#38;	&amp;	Ampersand	&
&#39;	-	Apostrophe	'
&#40;	-	Leftparenthesis	(
&#41;	-	Rightparenthesis	)
&#42;	-	Asterisk	*
&#43;	-	Plussign	+
&#44;	-	Comma	,
&#45;	-	Hyphen	-
&#46;	-	Period(fullstop)	.
&#47;	-	Solidus(slash)	/
&#48;-&#57;	-	Digits(0-9)	0-9
&#58;	-	Colon	:
&#59;	-	Semi-colon	;
&#60;	&lt;	Lessthan	<
&#61;	-	Equalssign	=
&#62;	&gt;	Greaterthan	>
&#63;	-	Questionmark	?
&#64;	-	Commercialat	@
&#65;-&#90;	-	UppercaseA-Z	A-Z
&#91;	-	Leftsquarebracket	[
&#92;	-	Reversesolidus(backslash)	\
&#93;	-	Rightsquarebracket	]
&#94;	-	Caret	^

Number	**Name**	**Description**	**Appearance**
&#95;	-	Horizontalbar	_
&#96;	-	Acuteaccent	`
&#97;-&#122;	-	Lowercasea-z	a-z
&#123;	-	Leftcurlybrace	{
&#124;	-	Verticalbar	|
&#125;	-	Rightcurlybrace	}
&#126;	-	Tilde	~
&#127;-&#159;	-	Unused	-
&#160;	&nbsp;	Non-breakingspace	
&#161;	&iexcl;	Invertedexclamation	¡
&#162;	&cent;	Centsign	¢
&#163;	&pound;	Poundsterlingsign	£
&#164;	&curren;	Generalcurrencysign	¤
&#165;	&yen;	Yensign	¥
&#166;	&brvbar;	Brokenverticalbar	¦
&#167;	&sect;	Sectionsign	§
&#168;	&uml;	Umlaut(dierisis)	¨
&#169;	&copy;	Copyright	©
&#170;	&ordf;	Feminineordinal	ª
&#171;	&laquo;	Leftanglequote,guillemotleft	«
&#172;	&not;	Notsign	¬
&#173;	&shy;	Softhyphen	
&#174;	&reg;	Registeredtrademark	®
&#175;	&macr;	Macronaccent	¯
&#176;	&deg;	Degreesign	°
&#177;	&plusmn;	Plusorminus	±
&#178;	&sup2;	Superscripttwo	²
&#179;	&sup3;	Superscriptthree	³
&#180;	&acute;	Acuteaccent	´
&#181;	&micro;	Microsign	µ
&#182;	&para;	Paragraphsign	¶
&#183;	&middot;	Middledot	·
&#184;	&cedil;	Cedilla	¸
&#185;	&sup1;	Superscriptone	¹
&#186;	&ordm;	Masculineordinal	º
&#187;	&raquo;	Rightanglequote,guillemotright	»

Number	Name	Description	Appearance
&#188;	&frac14;	Fraction(onequarter)	¼
&#189;	&frac12;	Fraction(onehalf)	½
&#190;	&frac34;	Fraction(threequarters)	¾
&#191;	&iquest;	Invertedquestionmark	¿
&#192;	&Agrave;	CapitalA,graveaccent	À
&#193;	&Aacute;	CapitalA,acuteaccent	Á
&#194;	&Acirc;	CapitalA,circumflexaccent	Â
&#195;	&Atilde;	CapitalA,tilde	Ã
&#196;	&Auml;	CapitalA,umlaut(dierisis)	Ä
&#197;	&Aring;	CapitalA,ring	Å
&#198;	&AElig;	CapitalAEdipthong(ligature)	Æ
&#199;	&Ccedil;	CapitalC,cedilla	Ç
&#200;	&Egrave;	CapitalE,graveaccent	È
&#201;	&Eacute;	CapitaE,acuteaccent	É
&#202;	&Ecirc;	CapitalE,circumflexaccent	Ê
&#203;	&Euml;	CapitalE,umlaut(dierisis)	Ë
&#204;	&Igrave;	CapitalI,graveaccent	Ì
&#205;	&Iacute;	CapitalI,acuteaccent	Í
&#206;	&Icirc;	CapitalI,circumflexaccent	Î
&#207;	&Iuml;	CapitalI,umlaut(dierisis)	Ï
&#208;	&ETH;	CapitalEth,Icelandic	Ð
&#209;	&Ntilde;	CapitalN,tilde	Ñ
&#210;	&Ograve;	CapitalO,graveaccent	Ò
&#211;	&Oacute	CapitalO,acuteaccent	Ó
&#212;	&Ocirc;	CapitalO,circumflexaccent	Ô
&#213;	&Otilde;	CapitalO,tilde	Õ
&#214;	&Ouml;	CapitalO,umlaut(dierisis)	Ö
&#215;	&times;	Multiplysign	×
&#216;	&Oslash;	CapitalO,slash	Ø
&#217;	&Ugrave	CapitalU,graveaccent	Ù
&#218;	&Uacute;	CapitalU,acuteaccent	Ú
&#219;	&Ucirc;	CapitalU,circumflexaccent	Û
&#220;	&Uuml;	CapitalU,umlaut(dierisis)	Ü
&#221;	&Yacute;	CapitalY,acuteaccent	Ý
&#222;	&THORN;	CapitalThorn,Icelandic	Þ
&#223;	&szlig;	Smallsharps,German(szligature)	ß

Number	**Name**	**Description**	**Appearance**
&#224;	&agrave;	Smalla,graveaccent	à
&#225;	&aacute;	Smalla,acuteaccent	á
&#226;	&acirc;	Smalla,circumflexaccent	â
&#227;	&atilde;	Smalla,tilde	ã
&#228;	&auml	Smalla,umlaut(dierisis)	ä
&#229;	&aring;	Smalla,ring	å
&#230;	&aelig;	Smallaedipthong(ligature)	æ
&#231;	&ccedil;	Smallc,cedilla	ç
&#232;	&egrave;	Smalle,graveaccent	è
&#233;	&eacute;	Smalle,acuteaccent	é
&#234;	&ecirc;	Smalle,circumflexaccent	ê
&#235;	&euml;	Smalle,umlaut(dierisis)	ë
&#236;	&igrave;	Smalli,graveaccent	ì
&#237;	&iacute;	Smalli,acuteaccent	í
&#238;	&icirc;	Smalli,circumflexaccent	î
&#239;	&iuml;	Smalli,umlaut(dierisis)	ï
&#240;	&eth;	Smalleth,Icelandic	ð
&#241;	&ntilde;	Smalln,tilde	ñ
&#242;	&ograve;	Smallo,graveaccent	òò
&#243;	&oacute;	Smallo,acuteaccent	ó
&#244;	&ocirc;	Smallo,circumflexaccent	ô
&#245;	&otilde;	Smallo,tilde	õ
&#246;	&ouml;	Smallo,umlaut(dierisis)	ö
&#247;	&divide;	Divisionsign	÷
&#248;	&oslash;	Smallo,slash	ø
&#249;	&ugrave;	Smallu,graveaccent	ù
&#250;	&uacute;	Smallu,acuteaccent	ú
&#251;	&ucirc;	Smallu,circumflexaccent	û
&#252;	&uuml;	Smallu,umlaut(dierisis)	ü
&#253;	&yacute;	Smally,acuteaccent	ý
&#254;	&thorn;	Smallthorn,Icelandic	þ
&#255;	&yuml;	Smally,umlaut(dierisis)	ÿ