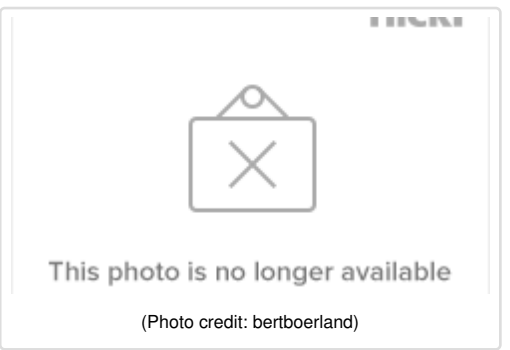
A Hitchhiker's Guide to Cross-Site Scripting (XSS) in PHP (Part 1): How Not To Use Htmlspecialchars() For Output Escaping



(Photo credit: bertboerland)

In recent weeks, I consulted with the second most intelligent species on the planet: Dolphins. Dolphins are renowned across the known Universe for their awesome programming skills. After

MY OPEN SOURCE PROJECTS

Mockery: The popular PHP mock object framework
Humbug: A Mutation Testing framework for PHP
phar-updater: Easy and secure self-update for PHARs
humbug_get_contents: An alternative to file_get_contents() preconfigured for TLS protection on remote HTTPS requests
For Zend Framework:
Zend\Feed: RSS/Atom Feed reader, writer and Pubsubhubbub support
Zend\Escaper: Escape HTML output for any context

RECENT POSTS

Self-Updating PHARs:

Up A Planet?". The answer to the last will be published on 01/01/2013 after the experiment is shut down and sent to a landfill site assuming the Supreme Spaghetti Monster signs off on the permit.

Dolphins think we are really dumb and theorise that this level of stupidity has one obvious cause: self-imposed ignorance. We are, after all, only the third most intelligent species on Earth and appear to have aspirations to lower our IQ just a bit more.

While it's no harm poking fun at ourselves, in PHP we do have a serious problem. Cross-Site Scripting (XSS) remains one of the most significant classes of security problems afflicting PHP applications. Despite years of education, community awareness and the development of frameworks which can offer a huge boost in consistent practices – things are not getting any better.

So, I finally figured out what the core problem is: PHP programmers are completely clueless about XSS. It's that simple. Instead of going out and studying the topic, we blindly follow some preferred herd of people offering advice with heartfelt conviction despite the fact that they are probably just as ignorant as the rest of us. Does that sound like the behaviour of something which allegedly evolved into an intelligent species? The result is a mix of ignorance and stagnant knowledge that leaves PHP in an unenviable position beset by wrongheaded zealots.

To get the ball rolling, this two-part article series is a tour of how NOT to use the htmlspecialchars() function that is typically pressed ganged into service as PHP's universal output escaper. By offering an example based guide, I hope it will illustrate just how many ways a prospective attacker

stable versions of my phar-updater package. So there. Announced. Can I go back to playing Witcher 3 now? […]

5 MONTHS AGO

PHP's "Magic Hash" Vulnerability (Or Beware Of Type Juggling) (6)
A while back, I noticed a flurry of activity around a somewhat obvious outcome of PHP's type juggling antics. As the snowball gathered pace […]

5 MONTHS AGO

TLS/SSL Security In PHP: Avoiding The Lowest Common Insecure Denominator Trap (1)
A few weeks back I wrote a piece about updating PHARs in-situ, what we've taken to calling "self-updating". In that article, I touched on […]

6 MONTHS AGO

Introduction to Humbug: A Mutation Testing Framework for PHP (5)
On 1 January 2015, I first pushed Humbug onto Github and three months later it is reaching a state where I can prep for the release of […]

6 MONTHS AGO

and/or lodge an official complaint with somebody who looks like they keep a complaints box handy (your local fast food restaurant is a good start).

This example led approach has another motive. Simple examples can be translated into unit tests. Ideally, many of the current crop of frameworks can use this article as a guide to what their unit tests should be looking for. This also makes it far easier for everyday programmers to consume the article and run around the place, drunk with ungodly power, identifying issues in the libraries, frameworks and other projects that they rely on.

To help us on the path of enlightenment before it's too late (I'd lodge an appeal with the Supreme Spaghetti Monster but apparently the Mayans already tried and failed), I also invite other PHP programmers to blog about a security topic over the next month or two. Give programmers one last chance to get it right before the Planet is demolished by the Vogon destructor fleet. Just pick a topic that drives you up the walls in defiance of gravity and spend an hour writing something useful and (optionally) expletive filled. Every little bit helps.

What Is Htmlspecialchars()?

According to many programmers from Earth, htmlspecialchars() is a function used to escape output to prevent XSS. This is however a completely wrong definition. The function was actually co-opted by programmers to combat XSS because it was either that or create slow userland functions for which the internals developers might get around to creating, when the full moon coincided with the right planetary alignment in another 314 years, a speedier C alternative to. The actual definition (along with a half-hearted

> *Certain characters have special significance in HTML, and should be represented by HTML entities if they are to preserve their meanings. This function returns a string with some of these conversions made; the translations made are those most useful for everyday web programming. If you require all HTML character entities to be translated, use htmlentities() instead. This function is useful in preventing user-supplied text from containing HTML markup, such as in a message board or guest book application.*

Note that this hints at, but does not explicitly use, the terms Cross-Site Scripting, XSS or even Security. Then again, it does refer to guest book applications so it was probably written in 1790 by the Dolphin who created PHP v86 and who then got around to backporting version 1.0 for Humans in the late 20th Century out of extreme pity for our reliance on CGI. No, not the let's take an action movie and turn it into a plotless eyesore with computer generated fake stuff style CGI – though memories of both are comparably bad.

Does this make htmlspecialchars() terrible at preventing XSS? No. As part of a comprehensive well-understood strategy to prevent XSS, the function is very useful. However, in PHP it is frequently overused, misused, abused, confused and…. Darn it, ran out of rhyming words again. Suffice it to say that a good description of htmlspecialchars() is that it's an unsuitable tool for preventing XSS that has slowly evolved into a better suited tool over the years. I keep telling myself that, at least.

The function, htmlspecialchars(), accepts four parameters. Here is its function prototype as of PHP 5.4:

The first parameter accepts a string whose special HTML characters will be converted to HTML entities. The second accepts one or more flags which defaults to using ENT_COMPAT (does not convert single quotes to entities) but should be set to use ENT_QUOTES (does convert single quotes to entities). You can include another flag, in PHP 5.4, called ENT_SUBSTITUTE which is not a bad idea for UTF-8, i.e. ENT_QUOTES | ENT_SUBSTITUTE. You can pretend that all the other constants don't exist. The third parameter accepts a string indicating the character encoding of the string being processed and defaults to ISO-8859-1 for PHP 5.3, and UTF-8 for PHP 5.4. Don't ever set the fourth parameter to FALSE when escaping unless your filtering logic was written by an Über Dolphin – always keep filtering and escaping separate from each other to avoid confusing the two and then having to pointlessly argue why your way is better in defiance of all logic.

The function, if correctly configured using this super simple article for guidance, will now convert the following characters to entities: <, >, ‘, ” and &. These characters make sense to escape since they are used to construct HTML tags, delineate attribute values or reference HTML entities – none of which we want users to be able to do!

If you want some very good advice before your brain implodes from too much reading, a good way to potentially make yourself vulnerable to XSS is to not explicitly set the first two optional parameters ($flags and $encoding) to an appropriate value. In fact, if you see htmlspecialchars() missing any of those two parameters in someone's source code, you should request that they fix it or, at the very least, curse their name and pray for the Supreme Spaghetti Monster to label them as biohazardous waste in need of

information. I'm told that this part is like being sucked into the Total Perspective Vortex machine on Frogstar World B.

To Quote Or Not To Quote: How Is That A Question?

As it turns out, HTML is not simply a popular markup language, it is a popular markup language designed by a bureaucratic species of transdimensional beings seeking to drive Humanity insane by inventing the most impossible-to-secure markup language known in 172 Universes which is then interpreted by "browsers" written by Mice to test the patience of security professionals and keep the really intelligent Humans distracted from the truth of their soon-to-end existence as they search out ever more ludicrous examples of parsing weirdness. Excuse me, I held my breath writing that and need to fetch my Oxygen tank…

Consider the following example. If you want to see whether they work without copy pasting, you can clone all examples from my ominously titled xss repository on Github into a webroot somewhere to read or execute them.

Single Quoted Attributes

```
1. <?php header('Content-Type: text/html; charset=UTF-8'); ?>
2. <!DOCTYPE html>
3. <?php
4. $input = <<<INPUT
5. ' onmouseover='alert(/Meow!/);
6. INPUT;
7. /**
8.  * NOTE: This is equivalent to using htmlspecialchars($input, ENT_COMPAT)
```

```
2. <html>
3. <head>
4.     <title>Single Quoted Attribute</title>
5.     <meta http-equiv="Content-Type"
   content="text/html; charset=UTF-8">
6. </head>
7. <body>
8.     <div>
9.         <span title='<?php echo $output
   ?>'>
0.             What's that latin placeholder
   text again?
1.         </span>
2.     </div>
3. </body>
4. </html>
```

If you run the example from a browser and pass your mouse pointer over the text, you will get a popup saying "/Meow!/". Granted, this is hardly the most impressive XSS ever but remember that the Javascript executed could be a lot more ingenious and damaging. The reason you see alert() used everywhere in XSS examples is to prove that Javascript was executable – a real attacker will hardly advertise his success like this.

In this case, the htmlspecialchars() function call omits the second parameter which defaults to using the ENT_COMPAT flag. With this setting, the function does not convert single quotes to entities, allowing us to inject an unescaped single quote (to close the title attribute value) and another to start a new attribute and value which will be closed by the final single quote used in the template.

We can fix this problem in one of two ways:

1. Use double quotes which will prevent user input from

2. Set the second parameter to htmlspecialchars() to use the ENT_QUOTES flag which will escape any single quotes a user tries to inject.

The moral of the story can be made even clearer by another example. In this case we use another perfectly validating means of delineating attribute values in HTML5 – we just don't bother using quotes at all!

Quoteless Attributes

```
1. <?php header('Content-Type: text/html; charset=UTF-8'); ?>
2. <!DOCTYPE html>
3. <?php
4. $input = <<<INPUT
5. faketitle onmouseover=alert(/Meow!/);
6. INPUT;
7. $output = htmlspecialchars($input, ENT_QUOTES);
8. ?>
9. <html>
0. <head>
1.     <title>Quoteless Attribute</title>
2.     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
3. </head>
4. <body>
5.     <div>
6.         <span title=<?php echo $output ?>>
7.             What's that latin placeholder text again?
8.         </span>
9.     </div>
0. </body>
```

character (including any character a browser might interpret as a space – there are a lot!) allows the user to inject new attributes and values. As from the above, converting all quotes to entities is pointless if there are no quotes to start with! Our escaping doesn't convert spaces or other space-interpreted characters into entities at all.

By now, you should see the obvious. All HTML attribute values MUST be quoted, and preferably DOUBLE quoted, in any scenario where you suspect untrusted input will be injected into an attribute value, or where htmlspecialchars() calls do not set the second parameter to use ENT_QUOTES. Believe it or not, using single quotes or no quotes remains popular and is perfectly valid under the new HTML5 spec. Some people even celebrate this new insanity. Keep an eye on any designers who look a bit wild eyed or spend too much time smiling while staring into empty space.

Excuse Me, Sir, But Someone Ate My Quotes

One of the great mysteries in escaping output is a common myth known as the Great ASCII Delusion (GAD). Those under the influence of this delusion, besides hearing voices in their head, have arrived at a belief that many character encodings are equivalent for the purposes of escaping those characters which have a special meaning for HTML, e.g ISO-8859-1 and UTF-8. Alas, this is untrue because the Mice created something called Internet Explorer 6 – a thoroughly shameful (but still commonly used) browser which corporations across the Planet continue to insist on using because buying new computers and upgrading operating systems just to use some fancy new Microsoft Office version is seen as a waste of shareholder funds.

Internet Explorer 6 is the bad boy of the XSS world since it's

this example using IE6 and PHP 5.3. If you need a testing version of all IE browsers since IE 5.5, you can download IETester from http://www.my-debugbar.com/ietester/index_all.php and use it from Windows. Try hard, I know Windows is bad and the new Tablet makeover for Windows 8 makes you feel ill, but it's important to see these examples in action.

Source code

```
<?php header('Content-Type: text/html; charset=UTF-8'); ?>
<!DOCTYPE html>
<?php
/**
 * You could also subsititute \xC0 or any other impacted character
 * above ASCII number 192
 */
$input1 = 'fakeimage'.chr(192);
$input2 = <<<INPUT2
onerror=alert(/Meow!/)//
INPUT2;
$output1 = htmlspecialchars($input1, ENT_QUOTES);
$output2 = htmlspecialchars($input2, ENT_QUOTES);
?>
<html>
<head>
    <title>Swallowed Quotes</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
```

```
    /images/<?php echo $output1 ?>"
3.          title="<?php echo $output2 ?>">
4.      </div>
5. </body>
6. </html>
```

With the above example, something very weird happens. Using ASCII character number 192 just before a double quote in a document being interpreted as UTF-8 results in the double quote…vanishing in IE6. Seriously, it's there but not there. Obviously the Mice are behind it – no Human could possibly defy Physics like this!

This allows an attacker to once again break out of the HTML attribute they can inject values into. Using a coincidental opportunity to inject a second free text string nearby which a browser will concatenate to the broken out attribute value of the first, you get an effective XSS combo attack.

This IE6 quirk even bypasses the call to htmlspecialchars() which, as explained above, defaults to the ISO-8859-1 character encoding for PHP 5.3 or less. If the Great ASCII Delusion were not a fabrication of someone's imaginative wishful thinking, this should not be possible. Not to be too harsh though, this weirdness is due primarily to a bug in IE6's treatment of the various character encodings where you can trick the browser into thinking something like \xC0 (in hex) is the start of a multi-byte character thus swallowing the next ASCII character (the double quote).

To fix the above weirdness, you must make sure that escaping is done using the same character encoding that the document is being served as. The above HTML document is identifying itself as being UTF-8 but the default htmlspecialchars() encoding is ISO-8859-1 in PHP 5.3 – there's obviously something not agreeing there! This brings

attackers:

Always set the third parameter to htmlspecialchars(), set it correctly, and make sure your document is never served with a mismatched or invalid character encoding! Don't expect some theoretically perfect world to magically appear – browsers are filthily efficient at doing weird things you don't expect.

I suppose I have to mention that most versions of IE have similar issues with other character encodings such as BIG5 and Shift-JIS. You can test your IE versions using http://ha.ckers.org/weird/variable-width-encoding.cgi to see what characters can be used across different character encodings. Believe it or not, these character encodings are actually still being used and, for some strange reason, people from China and Japan do use PHP.

If you want to be completely paranoid, you can either check the input for invalid UTF-8 (Drupal and HTMLPurifier have reusable functions/classes for this), and/or run it through a conversion function which should theoretically filter out the naughty bits:

```
$input = mb_convert_encoding($input, 'UTF-8', 'UTF-8');
```

This is probably a good idea for older PHP versions pre 2010 or earlier but recent PHP versions have specifically improved htmlspecialchars() to disallow invalid characters such as the above (if you set the right character encoding!). You should be aware, though, that htmlspecialchars() may still return blank strings on certain malformed input and, since PHP 5.4, will not issue any warnings about this.

I Broke It! I Broke It!

Source code

```
<?php header('Content-Type: text/html; charset=UTF-8'); ?>
<!DOCTYPE html>
<?php
$input1 = 'fakeimage'."\xC0";
$input2 = <<<INPUT2
onerror=alert(/Meow!/)//
INPUT2;
/**
 * If you think PHP 5.4 will save you - empty strings make it guess the encoding
 * or use the default_charset value from php.ini. You sure everyone on the whole
 * planet uses UTF-8? Under 5.3 - empty strings === default encoding.
 */
$encoding = ''; // from outside source or unvalidated variable
$output1 = htmlspecialchars($input1, ENT_QUOTES, $encoding);
$output2 = htmlspecialchars($input2, ENT_QUOTES, $encoding);
?>
<html>
<head>
    <title>Swallowed Quotes</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
    <div>
        <img src="http://example.com
```

```
7. </body>
8. </html>
```

Setting the third $encoding parameter of htmlspecialchars() to an empty string in PHP 5.4 will set the encoding to be auto-detected, grabbed from the php.ini value of default_charset, or guessed from the current locale (in that order). Be very careful under PHP 5.4 NEVER to let this happen. Don't leave your escaping parameters to chance.

Use empty() or strlen(), for example, to spot this issue if accepting encodings from another source or variable that might allow for empty strings. Again, this behaviour is very secure and there's nothing wrong with it whatsoever. Oh, who am I kidding… This is the dumbest parameter behaviour ever invented. NULL means use the default encoding; blank string means play a guessing game. Even Vogon poetry pales in comparison to such nonsense. One slip and an empty parameter string can rip apart this house of cards because who knows which character encoding will be used.

Oooh, I wonder what this does under PHP 5.3… Yes, er, don't allow blank encoding parameter strings under PHP 5.3 either. Setting an empty string in PHP 5.3 is interpreted as setting the default character encoding, i.e. ISO-8859-1, instead of triggering the expected warning about an unsupported encoding.

So, be careful kids. When setting the encoding for htmlspecialchars() do a safety check to make sure it's not an empty string you are passing in. Keep it predictable and consistent.

There's also one other curious behaviour when using htmlspecialchars().

```php
<?php header('Content-Type: text/html; charset=UTF-8'); ?>
<!DOCTYPE html>
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);
$input1 = 'fakeimage'."\xC0";
$input2 = <<<INPUT2
onerror=alert(/Meow!/)//
INPUT2;
/**
 * Invalid encoding makes htmlspecialchars() throw a warning but it continues
 * the current operation anyway using the default encoding even if the default
 * is an unsafe choice for the application. Don't allow invalid encodings!
 */
$encoding = 'invalid-encoding'; // from outside source or unvalidated variable
$output1 = htmlspecialchars($input1, ENT_QUOTES, $encoding);
$output2 = htmlspecialchars($input2, ENT_QUOTES, $encoding);
?>
<html>
<head>
    <title>Swallowed Quotes</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
    <div>
        <img src="http://example.com/images/<?php echo $output1 ?>"
```

```
0. </html>
```

When you set an invalid character encoding, not the empty string of doom, htmlspecialchars() will issue a Warning level error…and continue merrily on its way by reinstating the default encoding. In a production scenario, you will likely have display_errors disabled and this warning will be logged and possibly ignored by some users. If this makes it through, setting an invalid character encoding whether by a deliberate user value or simple programmer error may create an exploitable scenario.

So, make sure you also validate the character encoding. Don't just leave it up to htmlspecialchars() since it allows the continued execution of the application. Arguably this should be a fatal error since a bad encoding is itself a security problem.

Seriously, this function is like handing a box of matches to a Human and telling them there's a rainforest nearby that's essential to all life on Earth…

Internet Explorer: Master Of Supporting Stupid Character Encodings

Internet Explorer is unique in the Universe. Designed by Mice to be the dumbest, most frustrating, most stubbornly non-upgradeable piece of crap ever, it does things that make XSS far easier. The terrible part is that IE is popular with corporations and businesses using commodity hardware imported from whichever country currently has the lowest paid PC assemblers on Earth. One would think they'd like a more secure browser to protect their money making endeavours.

It's no wonder that Dolphins had to think long and hard

intelligence by showing how easy it is to make Humans cater to their every need…for free. Even their main rivals, Dogs, are expected to do useful work like herding sheep, chasing cars, digging holes, barking at strangers, and keeping bill bearing postal workers at bay.

All versions of Internet Explorer support a troublesome character encoding called UTF-7 which, oddly enough, is not supported by htmlspecialchars(). You can probably see where this is going. How do you escape a character encoding that your escaper doesn't even support? Easy, you can't. JUST DON'T USE UTF-7! EVER! UTF-7 has the distinction of definitely not being ASCII compatible – it encodes angle brackets (used to open and close HTML tags) very differently so they are never detected by filters or escapers relying on other character encodings.

Unfortunately, some applications do allow users to cherry pick an encoding. It's not uncommon in international websites (e.g. Google which had this problem). Here's an example of what not to do:

Source code

```
1. <?php
2. $input1 = 'UTF-7';
3. $input2 = <<<INPUT2
4. <script>alert(/Meow!/)//</script>
5. INPUT2;
6. $input2 = mb_convert_encoding($input2,
   'UTF-7', 'UTF-8');
7. $output1 = htmlspecialchars($input1,
   ENT_QUOTES, 'UTF-8');
8. $output2 = htmlspecialchars($input2,
```

```
charset='.trim($output1));
?>
<!DOCTYPE html>
<html>
<head>
    <title>Mismatched Encoding</title>
    <meta http-equiv="Content-Type"
    content="text/html; charset=<?php echo $output1 ?>">
</head>
<body>
    <div>
        <?php echo $output2 ?>
    </div>
</body>
</html>
```

This works in all IE versions. The problem here is that we're letting the user set the character encoding without validating it against a safe whitelist of encodings that we can actually escape. This also works even when you plead with the Supreme Spaghetti Monster and try passing UTF-7 to htmlspecialchars() since the function simply issues a warning and reinstates its ISO-8859-1 or UTF-8 default before continuing on its merry way to making you vulnerable to XSS. Yes, very secure behaviour there…

Note: putting the @ symbol in front of htmlspecialchars() to hide these warning errors during development is not considered an act worthy of an intelligent species. Don't let the cats win!

Now, you might think that this would be the end of it, but there's one other problem afflicting older browsers (fixed as of Internet Explorer 9). In certain scenarios you can trick the browser into rendering pages as UTF-7 even when you can't set the page's character encoding. This is due to a bug in

To pull off this exploit, you need to first set some UTF-7 text which is persisted across requests, e.g. a blog comment. Since we can't escape UTF-7 in PHP, the persisted text will contain some UTF-7 encoded XSS code. Just in case, you're smart and you're thinking that mbstring functions might help detect UTF-7 – they won't. mbstring will detect UTF-7 as UTF-8, and UTF-8 as UTF-7 depending on the detection order set in mb_detect_encoding(). After that it's a long winded story of using iframes to trick a browser into rendering the innocent looking UTF-7 strings on your webpages as UTF-7.

Where escaping fails, some common sense should win out. Just make sure all the responses you serve have a header that sets the appropriate character encoding for the content (use a valid encoding string, not an invalid string form). In HTML, use the relevant meta tag to indicate the content's character encoding as a backup should the header be somehow omitted.

Conclusion

Htmlspecialchars() as a function for escaping output has its limitations. If you're unaware of these and wish to persist in using it incorrectly, you should expect to be burned. No, seriously, there really is an incinerator for those labelled as biohazardous waste over in Alpha Centauri.

I get the feeling I've written enough for you today. I'm very sorry for the 0.006% of you that Vogon studies indicate are now sitting at their desk drooling all over their keyboards from encroaching insanity. If you're worried about joining the 0.006%, please submit the correct form in triplicate, completed in capitals using a blue ball-point pen, to your local

So, what next? In Part 2, we continue our voyage into madness with more examples using htmlspecialchars() though in another direction this time. In the meantime, you have a lot of examples (aka ammunition) and there are a lot of applications/frameworks/libraries (targets). I figure the rest is obvious.

See you for Part 2!

Share this:

Character encodings in HTML Cross-Site Scripting php xss

This entry was posted by padraic on March 12, 2012 at 8:49 pm, and is filed under PHP General, PHP Security, Zend Framework. Follow any responses to this post through RSS 2.0.You can leave a response or trackback from your own site.

- *Roberto Luengo*

  Nice article… interesting despite all the crap about dolphins, mice, etc.

  - *http://blog.astrumfutura.com Pádraic Brady*

    All the crap is in reference to Douglas Adam's excellent Hitchhiker's Guide To The Galaxy ;).

    - *Panajotis Zamos*

      nice crap 😀

    - *Muzammil Hussain*

      http://www.allmood.com/classifieds/

- *http://twitter.com/g0b0ss Michał Wyrwalski*

  A very valuable article. And the "crap about dolphins, mice, etc." makes it an easier read IMO 🙂

- *http://twitter.com/andremaha Andrey Esaulov*

  Wow, this is golden mine for the security know-how! Thanks for putting it out there! Most of the things I know, but what struck me is the character encoding problem in IE. Didn't know that!

  - *Muzammil Hussain*

    http://www.allmood.com/classifieds/

- *http://www.facebook.com/people/Robert-Baldessari/1788856485 Robert Baldessari*

  This would be a nice article for anyone who has the time to filter through all the crap. I got to about paragraph 6.

- *Muzammil Hussain*

  wwwallmood.com

- *Muzammil Hussain*

  best free classifieds http://www.allmood.com/classifieds/

- *http://www.facebook.com/hopeseekr Theodore R. Smith*

  I want to salute your awesome article! Already the plebs in reddit's /r/php are bemoaning your article, even going so far as to say "TL;DR".

  Your article is one of the best I've read in years, but you need to state how using htmlentities() to encode *all* UTF-8 characters can be very advantageous, as several UTF-7 and -8 chars are interpreted as both quotes and/or dashes by various browsers (particularly troubling: all IEs up to and including IE 9), even if you do all your coding right w/ htmlspecialchars().

  Please see my discussion of this on my StackOverflow answer: http://stackoverflow.com/a/3623297/430062

  - *http://www.survivethedeepend.com Pádraic Brady*

    Good point, htmlentities() has the exact same weakness and it also doesn't support UTF-7. As for the reddit plebes, they can go bite me ;). The TL;DR for this has been circulating for years and is still regularly ignored.

    - *Artur Bodera*

      I just finished praying to the Supreme Spaghetti Monster for that to never happen 🙂
      I welcome everyone to send their "TL;DR" to complaints box downstairs. Follow the white rabbit and watch your

so much fun your posts, I believe everyone would probably benefit more from this insightful info if you would make it more executive and to the point. Anyway, thanks really a lot!!!

- *http://www.coreyballou.com/ Corey Ballou*

  I think one of the key missing pieces of information here is how to properly handle invalid multi-byte characters. You describe them triggering an E_WARNING exception, but don't provide users with an appropriate solution that should be used to mitigate these problems and more advanced attack vectors, i.e.:

  try {
  $string = mb_convert_encoding($string, 'UTF-8',
  mb_detect_encoding($string));
  return htmlentities($string, ENT_QUOTES, 'UTF-8', false);
  } catch (Exception $e) {
  return '';
  }

- *Anonymous*

  WinWinHost.com specializes in Information Technology, affordable Web Hosting,

  Web Development, Internet Marketing, Software Engineering and outsourcing. We

  can help you launch a product, or service, whether its a software package for a

  mobile device, or an eCommerce website to sell your inventory, or even a simple

  blog to help market your business.

  http://www.winwinhost.com/

- *Artur Bodera*

  Best.. article… evar! 🙂 As always from Paddy.
  Thank you!

- *http://www.annzoseo.com christinaxio*

  Thanks for sharing this it will help me a lot….

- *amir freed*

  Well the discussion is very interesting, Actually i am looking for this discussion you explain it very well,I can say that you have a quality to

Great artical. THanks

- *Ulf Härnhammar*

  Let's give up on the web and go back to character-based curses interfaces instead.

  // Ulf (kses guy)

- *http://www.plumbersdepot.co.uk/brand/grohe/30/ Grohe*

  It is very nice information to guide for programming.

- *syed nouman*

  http://indopakfashion.com/

- *syed nouman*

  http://indopakfashion.com/

- *syed nouman*

  Haemophilia Inherited disorder impacting blood clog formation
  The human body is a too complicated program. Many people have dedicated their life to the research of the human body & the numerous conditions impacting the human body. The latest years have observed great of research on the all-essential genes & how they impact the offspring. Professionalsrevealed a excellent attention in examining about historic methods & how a situation passes on from the family to the young. Genes perform a too crucial role in the exchange of features by the offspring.
  http://ahealthyclub.com/2012/03/29/haemophilia-inherited-disorder-impacting-blood-clog-formation/

- *Lafayette Internal*

  Let's give up on the web and go back to character-based curses interfaces instead.

- *Anonymous*

  Thank you very much for this post. I am getting a grip on how to use these codes. Thanks http://www.g-covers.com/

- *http://kangjem.blogspot.com/ kangjem*

  Wow, this is golden mine for the security know-how! Thanks for putting it out there! Most of the things I know, but what struck me is the character encoding problem in IE. Didn't know that!

- *http://kangjem.blogspot.com/ kangjem*

  Wow, this is golden mine for the security know-how! Thanks for putting

YourSocialFans.com can help you attract thousands of followers that you can keep informed about any of your product or service offerings instantly. By bringing you a targeted crowd of buyers all you need to do is give them an offer they can't refuse! Branding is also another successful tool Twitter can provide for your business. As more and more people become followers of your page it builds trust in them and they are more likely to buy from you than your competitor, because they have been following the brand and it's a brand they trust.

- *yoursocial fans*

  YourSocialFans.com can help you attract thousands of followers that you can keep informed about any of your product or service offerings instantly. By bringing you a targeted crowd of buyers all you need to do is give them an offer they can't refuse! Branding is also another successful tool Twitter can provide for your business. As more and more people become followers of your page it builds trust in them and they are more likely to buy from you than your competitor, because they have been following the brand and it's a brand they trust.

- *http://ipad3keyboard.com/ ipad 3 keyboard*

  I can not be more agree that IE is just b*****t . Good to see like-minded people.

- *http://cnacertificationtrainingcourses.com/ cna training*

  hey I am like your article really it is very intrusting .
  thanks

- Pingback: Automatic Output Escaping In PHP And The Real Future Of Preventing Cross-Site Scripting (XSS) | Pádraic Brady()
- Pingback: CORY()
- Pingback: 7 & 10 Inch Tablet Pc()
- Pingback: scott()
- Pingback: Mike()
- Pingback: william()
- Pingback: Earl()
- Pingback: Lawrence()
- Pingback: lawyer jargon()
- Pingback: Timothy()

- Pingback: Tommy()
- Pingback: Carlos()
- Pingback: Carlos()
- Pingback: Armando()
- Pingback: dale()
- Pingback: Julian()