2017年5月10日 由YUER

为htmlpurifier订制xss过滤

xss简介

我们每天读的公众号文章，其内容是小编通过富文本编辑器生成的。内容发布者在富文本编辑器里可视化的编写文字与图片并进行简单的排版，背后其实是编辑器动态的生成各种html标签，最终文章对应的完整html会被提交到服务端保存。当读者打开文章时，服务端将保存的html直接返回给浏览器进行渲染。

读者直接获取并展现小编提交的html，这是极其危险的，万一html里有一段<script>包裹的Js代码，那读者的信息就可以被小编通过Js代码盗走，这种攻击手段叫做xss。当然，像<script>这种xss手法是很好防御的，只需要检查小编提交的html过滤掉<script>标签即可，但是xss远没有这么简单。恶意的小编会利用错乱的<、>、引号等导致最终读者的页面出现意想不到的标签闭合效果，或者利用onclick，onerror等标签的事件函数来实现js代码的执行，等等诸如此类的注入手法。

那么xss过滤就是用来检查小编提交的html，将存在安全隐患的代码移除掉，将不规范的代码整理或移除，最终留下一个安全可靠的html，安全到可以直接展现给读者。

htmlpurifier基础使用

既然xss的注入手段各式各样，光是事件函数的类型都数不过来，因此不如找一个历史积累丰富的xss过滤开源库，那就是著名的htmlpurifier了，像有名的Yii2框架也是内置了这个库。

如果你是composer项目，可以通过composer安装与自动加载。如果是比较老的项目，可以下载standalone版本，并通过require_once引入。

基础用法不需要任何配置，直接将小编提交的html传给htmlpurifier即可完成过滤：

```php
$config = HTMLPurifier_Config::createDefault();
$purifier = new HTMLPurifier($config);
$clean_html = $purifier->purify($dirty_html);
```

htmlpurifier添加标签和属性

这个库遵循HTML标准，会保留常见的各种HTML标签，而删除库中未定义的标签。对于标签属性也是如此，只会保留合法的HTML定义的标准属性。通常来说，你不必对它做什么订制就可以正常工作了，但是也不排除特殊需求。

一些情况下，你会希望保留某些自定义属性，或者保留某些自定义的标签，这时候必须订制htmlpurifier。订制并不是很复杂，可以参考官方文档来学习如何订制标签和属性，网上鲜有博客说明这一块的具体做法。

我遇到的情况是，公司的某个富文本中包含<dir>标签被htmlpurifier删除了，我查了一下HTML标准获知，这个标签是符合标准的，但是不推荐使用了。没办法，我必须订制它，避免htmlpurifier将其过滤：

```php
        // 默认配置
        $config = HTMLPurifier_Config::createDefault();
        // 自定义配置 http://htmlpurifier.org/docs/enduser-customize.html
        $config->set('HTML.DefinitionID', 'smzdm library version');
        $config->set('HTML.DefinitionRev', 2);
        if ($def = $config->maybeGetRawHTMLDefinition()) {
            // 允许卡片的<dir>标签
            $def->addElement(
                'dir',
                'Block',
                'Flow',
                'Common',
                [
                    'res-data-id' => 'Number'
                ]
            );
        }

        $this->html_purifier = new HTMLPurifier($config);
```

这里多了2个set操作，第一个是设置这个配置的名称，第二个是配置的版本，它俩会作为配置的缓存key，最终这份配置会被缓存到磁盘到一个文件里。

maybeGetRawHTMLDefinition()会检查"名称+版本"的缓存文件是否存在，如果已经存在则返回的$def为null并直接使用缓存文件的配置，否则会执行你的配置并缓存到文件中去。因此，如果你正在调试各种配置项，那么每次修改完配置代码都应该修改版本号来避免加载缓存的配置，以便看到实时的修改结果。

另外一种调试期间防止缓存的方法，是将maybeGetRawHTMLDefinition替换成永远不加载缓存的API：

```php
        // 默认配置
        $config = HTMLPurifier_Config::createDefault();
        // 自定义配置 http://htmlpurifier.org/docs/enduser-customize.html
        $config->set('HTML.DefinitionID', 'smzdm library version');
        $config->set('HTML.DefinitionRev', 1);
        $config->set('Cache.DefinitionImpl', null); // 清理配置缓存,上线时关掉这句
        $def = $config->getHTMLDefinition(true);
        // 允许卡片的<dir>标签
        $def->addElement(
            'dir',
            'Block',
            'Flow',
            'Common',
            [
                'res-data-id' => 'Number'
            ]
        );

        $this->html_purifier = new HTMLPurifier($config);
```

这可以方便你调试配置时立即看到效果，但是上线时请用第一份代码，因为它会生成并且只生成一次缓存，而第二份代码永远不会生成缓存，性能差并且会发出代码警告。

我又通过addElement订制了一个<dir>标签。

它是一个块级元素（Block），我们知道合理的HTML规范是块级元素内可以有块级元素与内联元素，而内联元素内只能有内联元素。

Flow指定了<dir>允许孩子是块级元素或者内联元素，这和HTML规范一致。

Common指定了<dir>可以包含一些常见的属性：`id`, `style`, `class`, `title` and `lang`..

最后一个参数是为<dir>添加了自定义属性，这样res-data-id属性就不会被htmlpurifier过滤掉了，它的合法值是一个整形，否则将被过滤掉。

封装htmlpurifier

公司的富文本框做了一些订制，比如允许嵌入embed视频。为了更方便配置和管理Htmlpurifier，我仿照这个开源项目对其进行了进一步封装如下。注意，该开源项目在配置文件缓存使用方面存在BUG，我已对其修复。

```php
<?php

require_once(__DIR__ . '/HTMLPurifier.standalone.php');

/**
 * Class purifier
 *
 * 富文本过滤器
 *
 * 封装 http://htmlpurifier.org/docs
 */

class Purifier
{
    /**
     * @var config
     */
    protected $config;

    /**
     * @var HTMLPurifier
     */
    protected $purifier;

    /**
     * Constructor
     *
     * @throws Exception
     */
    public function __construct()
    {
        $this->config = require(__DIR__ . '/config/purifier.php');
        $this->setUp();
    }

    /**
     * @param $key
     * @param null $default
     * @return null
     */
    private function config($key, $default = null) {
        $pathes = explode('.', $key);

        $cur_config = $this->config;
        foreach ($pathes as $path) {
            if (!isset($cur_config[$path])) {
                return $default;
            }
            $cur_config = $cur_config[$path];
        }
        return $cur_config;
    }

    /**
     * Setup
     *
     * @throws Exception
     */
    private function setUp()
    {
        // Create a new configuration object
        $config = HTMLPurifier_Config::createDefault();

        // 基础配置
        $config->loadArray($this->getConfig());

        // 订制配置
        $definition = $this->config('settings.custom_definition');
        if (!empty($definition)) {
            $config->set('HTML.DefinitionID', $definition['id']);
            $config->set('HTML.DefinitionRev', $definition['rev']);
            // Enable debug mode
            if (!isset($definition['debug']) || $definition['debug']) {
                $config->set('Cache.DefinitionImpl', null);
            }
            // 优先加载缓存的配置，如果不存在就执行订制代码
            if ($def = $config->maybeGetRawHTMLDefinition()) {
                $this->addCustomDefinition($definition, $def);
            }
        }

        // Create HTMLPurifier object
        $this->purifier = new HTMLPurifier($this->configure($config));
    }

    /**
     * Add a custom definition
     *
     * @see http://htmlpurifier.org/docs/enduser-customize.html
     * @param array $definitionConfig
     * @param HTML_Purifier_Config $configObject Defaults to using default config
     *
     * @return HTML_Purifier_Config $configObject
     */
    private function addCustomDefinition(array $definitionConfig, $defObj = null)
    {
        // Create the definition attributes
        if (!empty($definitionConfig['attributes'])) {
            $this->addCustomAttributes($definitionConfig['attributes'], $defObj);
        }

        // Create the definition elements
        if (!empty($definitionConfig['elements'])) {
            $this->addCustomElements($definitionConfig['elements'], $defObj);
        }
    }

    /**
     * Add provided attributes to the provided definition
     *
     * @param array $attributes
     * @param HTMLPurifier_HTMLDefinition $definition
     *
     * @return HTMLPurifier_HTMLDefinition $definition
     */
    private function addCustomAttributes(array $attributes, $definition)
    {
        foreach ($attributes as $attribute) {
            // Get configuration of attribute
            $required = !empty($attribute[3]) ? true : false;
            $onElement = $attribute[0];
            $attrName = $required ? $attribute[1] . '*' : $attribute[1];
            $validValues = $attribute[2];

            $definition->addAttribute($onElement, $attrName, $validValues);
        }

        return $definition;
    }

    /**
     * Add provided elements to the provided definition
     *
     * @param array $elements
     * @param HTMLPurifier_HTMLDefinition $definition
     *
     * @return HTMLPurifier_HTMLDefinition $definition
     */
    private function addCustomElements(array $elements, $definition)
    {
        foreach ($elements as $element) {
            // Get configuration of element
            $name = $element[0];
            $contentSet = $element[1];
            $allowedChildren = $element[2];
            $attributeCollection = $element[3];
            $attributes = isset($element[4]) ? $element[4] : null;

            if (!empty($attributes)) {
                $definition->addElement($name, $contentSet, $allowedChildren, $attributeCollection, $attributes);
            } else {
                $definition->addElement($name, $contentSet, $allowedChildren, $attributeCollection);
            }
        }
    }

    /**
     * @param HTMLPurifier_Config $config
     *
     * @return HTMLPurifier_Config
     */
    protected function configure(HTMLPurifier_Config $config)
    {
        return $config;
    }

    /**
     * @return array|null
     */
    protected function getConfig()
    {
        $default_config = [];
        $default_config['Core.Encoding'] = $this->config('encoding');

        $config = $this->config('settings.default');

        if (!is_array($config)) {
            $config = [];
        }

        $config = $default_config + $config;

        return $config;
    }

    /**
     * @param      $dirty
     *
     * @return mixed
     */
    public function purify($dirty)
    {
        return $this->purifier->purify($dirty);
    }
}
```

配置文件可以这样来写：

```php
<?php
/**
 * Ok, glad you are here
 * first we get a config instance, and set the settings
 * $config = HTMLPurifier_Config::createDefault();
 * $config->set('Core.Encoding', $this->config->get('purifier.encoding'));
 * $config->set('Cache.SerializerPath', $this->config->get('purifier.cachePath'));
 * if ( ! $this->config->get('purifier.finalize')) {
 *     $config->autoFinalize = false;
 * }
 * $config->loadArray($this->getConfig());
 *
 * You must NOT delete the default settings
 * anything in settings should be compacted with params that needed to instance HTMLPurifier_Config.
 *
 * @link http://htmlpurifier.org/live/configdoc/plain.html
 */

return [
    'encoding'      => 'UTF-8',
    'settings'      => [
        // 默认配置
        'default' => [
            // 采用Htmlpurifier默认即可，一旦配置将完全覆盖Htmlpurifier的默认值，自定义很难覆盖所有tag和attr

            //'HTML.Doctype'             => 'HTML 4.01 Transitional',
            //'HTML.Allowed'             => 'div,b,strong,i,em,u,a[href|title],ul,ol,li,p[style],br,span[style],img[width|height|alt|src]',
            //'CSS.AllowedProperties'    => 'font,font-size,font-weight,font-style,font-family,text-decoration,padding-left,color,background-color,text-align',
            // 给文字包裹<p>标签
            //'AutoFormat.AutoParagraph' => true,
            // 删除没有孩子的标签
            //'AutoFormat.RemoveEmpty'   => true,

            'HTML.SafeEmbed' => true, // 允许embed视频
            'HTML.SafeIframe' => true, // 允许iframe
            'URI.SafeIframeRegexp' => '%^http://(.+?\.youku\.com/|.+?\.tudou\.com/|.+\.56\.com/)%', // iframe的src校验
            'Attr.DefaultImageAlt' => '',
        ],
        'custom_definition' => [
            'id'  => 'my-definitions',
            'rev' => 2,
            'debug' => false,
            'elements' => [
                // http://developers.whatwg.org/sections.html
                ['section', 'Block', 'Flow', 'Common'],
                ['nav',     'Block', 'Flow', 'Common'],
                ['article', 'Block', 'Flow', 'Common'],
                ['aside',   'Block', 'Flow', 'Common'],
                ['header',  'Block', 'Flow', 'Common'],
                ['footer',  'Block', 'Flow', 'Common'],

 // Content model actually excludes several tags, not modelled here
                ['address', 'Block', 'Flow', 'Common'],
                ['hgroup', 'Block', 'Required: h1 | h2 | h3 | h4 | h5 | h6', 'Common'],

 // http://developers.whatwg.org/grouping-content.html
                ['figure', 'Block', 'Optional: (figcaption, Flow) | (Flow, figcaption) | Flow', 'Common'],
                ['figcaption', 'Inline', 'Flow', 'Common'],

 // http://developers.whatwg.org/the-video-element.html#the-video-element
                ['video', 'Block', 'Optional: (source, Flow) | (Flow, source) | Flow', 'Common', [
                    'src' => 'URI',
 'type' => 'Text',
 'width' => 'Length',
 'height' => 'Length',
 'poster' => 'URI',
 'preload' => 'Enum#auto,metadata,none',
 'controls' => 'Bool',
                ]],
                ['source', 'Block', 'Flow', 'Common', [
 'src' => 'URI',
 'type' => 'Text',
                ]],

 // http://developers.whatwg.org/text-level-semantics.html
                ['s',    'Inline', 'Inline', 'Common'],
                ['var',  'Inline', 'Inline', 'Common'],
                ['sub',  'Inline', 'Inline', 'Common'],
                ['sup',  'Inline', 'Inline', 'Common'],
                ['mark', 'Inline', 'Inline', 'Common'],
                ['wbr',  'Inline', 'Empty', 'Core'],

 // http://developers.whatwg.org/edits.html
                ['ins', 'Block', 'Flow', 'Common', ['cite' => 'URI', 'datetime' => 'CDATA']],
                ['del', 'Block', 'Flow', 'Common', ['cite' => 'URI', 'datetime' => 'CDATA']],

                // 允许使用dir
                ['dir', 'Block', 'Flow', 'Common', [
                    'res-data-id' => 'Number',
                ]],
            ],
            'attributes' => [
                ['iframe', 'allowfullscreen', 'Bool'],
                ['table', 'height', 'Text'],
                ['td', 'border', 'Text'],
                ['th', 'border', 'Text'],
                ['tr', 'width', 'Text'],
                ['tr', 'height', 'Text'],
                ['tr', 'border', 'Text'],
            ],
        ],
    ],

];
```

这份代码可以正常工作，通过配置即可改变htmlpurifier的行为，便于你探索与调试。

本文抛砖引玉，如果有疑惑就认真读一下官方文档，多测试多尝试。