给开发者的

终极 XSS

防护备忘录 V1.0

Ajin Abraham

OWASP Xenotix XSS 漏洞利用框架作者 | opensecurity.in

提供给开发者的 Web 应用程序 XSS 防护快速指南

简介:什么是 XSS?

XSS 或者说跨站脚本是一种 Web 应用程序的漏洞，当来自用户的不可信数据被应用程序在没有验证以及反射回浏览器而没有进行编码或转义的情况下进行了处理，导致浏览器引擎执行了代码。

XSS 类型

- 反射型 XSS
- 存储型 XSS
- DOM XSS
- 突变 XSS

反射型 XSS

反射或者非持久型 XSS 是当不可信的用户输入被服务器在没有任何验证下处理并在没有编码或转义的情况下反射回响应文中，导致代码在浏览器执行的一种 XSS 漏洞。

存储型 XSS

存储或持久型 XSS 是当不可信的用户输入被处理并在没有任何验证的情况下保存在文件或数据库，同时该不可信的数据从存储中被获取然后在没有编码或转义的情况下反射回响应文中，导致了永久性的每次存储数据反射回响应文代码就会在浏览器中执行的一种 XSS 漏洞。

DOM XSS

DOM XSS 是客户端 XSS 的一种形式，数据来源在 DOM 中，接收器也在 DOM 中，而数据流从来没有离开浏览器。它发生在一个不可信的数据在源中被给予并被执行，结果导致修改了 DOM 在浏览器中的“环境”。DOM XSS 攻击发生在不可信数据相对于上下文没有被编码或转义的情况下。

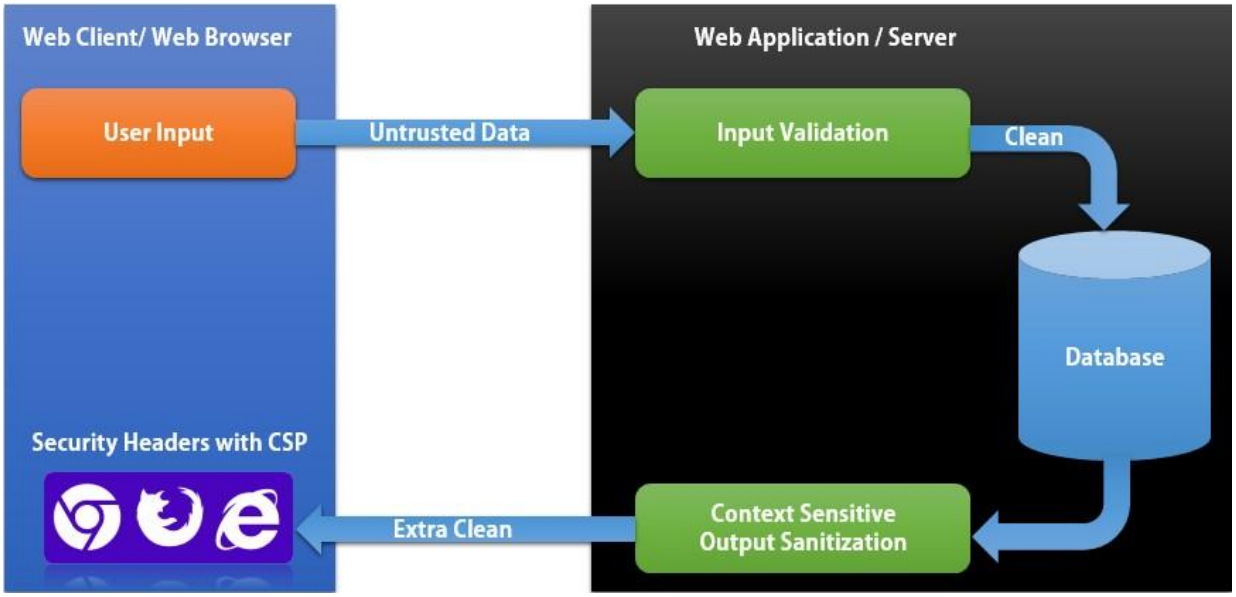
突变 XSS

mXSS 或突变 XSS 是当不可信数据在 DOM 的 innerHTML 属性的上下文被处理并通过浏览器发生突变，导致变成一种有效的 XSS 向量的一种 XSS 漏洞。在 mXSS，一个看起来无害的可以通过客户端或服务端 XSS 过滤器的用户指定的数据通过浏览器执行引擎发生突变可以反射回一个有效的 XSS 向量。XSS 过滤器不能防止 mXSS。为了防止 mXSS，应实施有效的 CSP，框架应该不被允许，HTML 文档应该定义文档类型，强制浏览器遵循标准呈现内容以及执行脚本。

XSS 防护

如果你可以使得一个 Web 应用程序满足以下规则，XSS 可以被减少。

1. 验证输入并且基于语境和按照正确的顺序转义不可信数据



输入验证

所有不可信数据应该针对 Web 应用程序的逻辑在处理和存储前进行验证。

浏览器解析顺序

HTML Parser >> CSS Parser >> JavaScript Parser

浏览器解码顺序

HTML Decoding >> URL Decoding >> JavaScript Decoding

解码和解析顺序意味着很多东西。如果对不可信数据的编码或解码以错误的顺序或错误的环境，将再次有机会导致 XSS 漏洞的发生。编码或者转义对不同的环境要求不同。这些编码的顺序应该取决于应用程序的逻辑。

一个典型的不可信数据可以反射在 HTML，HTML 属性，脚本变量，脚本块，含状态传输的参数，URL、风格等中。不同的转义方法为了确保 XSS 的防护必须要在不同的环境中实现。

顺序和上下文相关转义

1. HTML 中的字符串

对 HTML 中不可信字符串进行 HTML 转义。

Example:

```
1. <h1> Welcome html_escape(untrusted string) </html>
```

符号	编码
&	&amp;
<	<
>	>
"	"
`	`
'	'
/	/

2. HTML 属性中的字符串

对于 HTML 属性中不可信字符串进行 HTML 转义，并且总是为你的属性加上引号，无论是(‘或“)，不要使用反引号 (`)。

Example:

```
1. <img src="x" alt="html_escape(untrusted string)">
```

除了字母数字字符，用 &#xHH; 格式(或者命名实体，如果可用)转义所有 ASCII 值小于 256 的字符以防止开关的值伸出属性。 恰当的为属性加上引号可以只被对应的引号转义。不带引号的属性可以被分解为许多个字符, 包括[space] % * + , - /< = > ^ 种 。

3. 事件句柄属性和 JavaScript 中的字符串

对于事件句柄属性中的不可信字符串，先进行 JavaScript 转义，然后执行 HTML 转义，因为浏览器在执行 JavaScript 字符串解码前执行 HTML 属性解码。对于 JavaScript 中的非可信数据，进行 JavaScript 字符串转义并且总是将属性加上引号，无论是(‘ 或 “)，但不要使用反引号(`)。

> Example:
>
> ```
> 1. //In the context of Event Handler
> 2. <img src="x"
>    onload="jsFunction('html_escape( javascript_string_escape(untrusted string))')">
> 3. //In the context of JavaScript
> 4. <script type="text/javascript">
> 5. var abc = 'javascript_string_escape(untrusted string)';
> 6. </script>
> ```

除了字母数字字符，以 \ **xHH** 格式转义小于 256 的所有字符，以防开关的值伸到脚本中或另一个属性。不要使用类似\"的转义符号，因为引号符号可能被 HTML 属性解析器在第一次运行被匹配。这些转义字符也容易受到” 转义已转义” 的攻击，攻击者发送 \" 和漏洞代码转为\\" 使得可以成为引号。如果事件句柄属性被正确引号，需要通过相应的引号来闭合。不带引号的属性可以被分割为许多字符，包括**[space] % * + , - / ; < = > ^** 和 **|**. 另外，一个 **</script>** 闭合标签可以闭合脚本块，即使它是一个带引号的字符串。需要注意的是，HTML 解析器在 JavaScript 解析器之前运行。

4. HTML 属性中的 URL 路径

对 HTML 属性中的 URL 路径进行转义而不是完整的 URL。总是为属性加上引号，无论是 (‘ 或 “)，但不要使用反引号 (`)。绝不允许 **href** 或 **src** 包含格式像**javascript:** 或**data:** 或者他们的组合(如 **javas	cript**)。

Example:

```
1. <a href="http://xy.com/index?test=url_escape(untrusted string)">l</a>
2. <img src="/url_escape(untrusted url path string)">
3. <form action="/?i=url_escape(untrusted string)" method="GET"></form>
```

除了字母数字字符, 用 %HH 格式转义所有 ASCII 值小于 256 的字符。如果 href 或 src 属性被正确的引号起来，突破需要对应的引号。未被引号属性可以使用许多字符进行突破，包括 [space] % * + , - / ; < = > ^ 和^ < = >。请注意，这种情况下，实体编码是无用的。

5. HTML 风格属性和 CSS 中的字符串

对 HTML 样式属性内的不可信字符串先做 CSS 字符串转义，然后进行 HTML 转义，因为解析器的解析顺序是先 HTML 解析器然后再 CSS 解析器。总是给你的属性加上引号，如本例子中的风格属性加上(“)，CSS 字符串加上 (‘)，不要使用反引号(` )。为在 CSS 中的不可信字符串做 CSS 字符串转义。也要确保不可信字符串在引号 ( ‘ 或 “ ) 之间，不要使用反引号 ( `)。 也不要允许 expression 以及它的复杂组合，如(expre/**/ssion)。

Example:

```
1. //In the context of HTML style Attribute
2. <p style "font
   family:'html_escape(css_string_escape(unntrusted string))'">
3. Hello World!
4. </p>
5. //In the context of CSS
6. <style>
7. #css_string_escape(untrusted string)
8. {
9.     text-align: center;
10.    color: red;
11. }
12. </style>
```

除了字母数字字符, 用 \HH 格式转义所有 ASCII 值小于 256 的字符。不要使用任何类似 \" 的转义符号，因为引号符号可能被 HTML 属性解析器在第一次运行被匹配。这些转义字符也容易受到” 转义已转义” 的攻击，攻击者发送\" 和漏洞代码转为\\" 使得可以成为引

号。如果属性被正确引号，需要通过相应的引号来闭合。不带引号的属性可以被分割为许多字符，包括[space] % * + , - / ; < = > ^ 和 |. 。同时，</style> 标签会关闭风格块，即使是在一个被引号的字符串内。请注意，HTML 解析器在 CSS 解析器前运行。

6. JavaScript 中的 HTML

对于 JavaScript 字符串中不可信的 HTML，先执行 HTML 转义，然后执行 JavaScript 字符串转义， 保持这个顺序。

Example:

```
1. <script>
2. function xyz()
3. {
4. var elm=document.getElementById("disp");
5. elm.innerHTML="<strong>javascript_string_escape(html_escape(
   untrusted string))</strong>";
6. }
7. </script>
8. <body onload=xyz()>
9. <div id="disp"></div></body>
```

2. 始终遵循白名单优于黑名单的做法

创建个 Web 应用程序应该允许的来自用户的标签和属性的白名单。黑名单可以很容易的被绕过。

3. 使用 UTF-8 为默认的字符编码以及设置 content 为 text/html

在 HTML 文档，你可以指定 meta 标签，如 <meta http-equiv="content-type" content="text/html; charset=UTF-8">

4. 不要将用户可以控制的文本放在<meta>标签前。通过使用不同的字符集注射可以导致 XSS。

在 meta 之前，可以通过注射覆盖默认的字符集，并允许宽字节创建一个有效的 XSS 向量。

Demo

http://opensecurity.in/labz/opensecurity_meta.html

5. 使用<!DOCTYPE html>

DOCTYPE (DTD or Document Type Declaration，译者注:文档类型声明) 告诉你的浏览器遵循标准进行 HTML，CSS 的渲染以及如何执行脚本。总是在**<html>**. 之前使用 **<!doctype html>** 。

6. 使用推荐的 HTTP 响应头进行 XSS 防护

HTTP 响应头	描述
X-XSS-Protection: 1; mode=block	该响应头会开启浏览器的防 XSS 过滤器。
X-Frame-Options: deny	该响应头会禁止页面被加载到框架。
X-Content-Type-Options: nosniff	该响应头会阻止浏览器做 MIMEtype（译者

	注:Multipurpose Internet Mail Extensions，代表互联网媒体类型）嗅探。.
Content-Security-Policy: default-src 'self'	该响应头是防止 XSS 最有效的解决方案之一。它允许我们定义从 URLS 或内容中加载和执行对象的策略
Set-Cookie: key=value; HttpOnly	Set-Cookie 响应头通过 HttpOnly 标签的设置将限制 JavaScript 访问你的 Cookie。
Content-Type: type/subtype; charset=utf-8	始终设置响应的内容类型和字符集. 例如: 返回 json 格式应该使用 application/json, 纯文本使用 text/plain, HTML 使用 text/html 等等，以及设置字符集为 utf-8。

7. 防止 CRLF 注入/HTTP 响应拆分

对所有用户提供的数据在它们传递通过 HTTP 头前进行正确的清洁和编码。CRLF 注入可以摧毁和绕过 CSP，X-XSS 等所有的安全响应头。

8. 禁止 TRACE 和其他非必要方法

TRACE 是一种用于调试的 HTTP 方法，将反射来自客户端的请求头，在 HTTP 响应中返回给客户端。使用 TRACE 方法在请求头进行注入可以导致 XSS。

有效实施内容安全策略(CSP)的简易指南

CSP 或内容安全策略将在浏览器上强制执行策略，指定浏览器应该加载哪些资源，浏览器应该从哪里加载资源以及通过指令进行资源定义，指定资源加载行为。该文档将给告诉你如何根据您的要求定义一个 CSP。

我们感兴趣的 CSP 的指令是：

指令	描述
default-src	该指令在某种资源类型指定指令没有被定义的情况下指定了所有资源类型的加载策略(译者注：即默认的资源加载策略)。
script-src	该指令指定了 Web 应用程序可以加载的脚本的域或 URL。
object-src	该指令指定了 Web 应用程序可以加载的插件，如 Falsh。
style-src	该指令指定了 Web 应用程序可以加载的 CSS 样式表的域或 URL。
img-src.	该指令指定了 Web 应用程序可以加载的图片的域或 URL。
media-src	该指令指定了 Web 应用程序可以加载的音视频的域或 URL。
frame-src	该指令指定了 Web 应用程序可以加载的框架的域或 URL。
font-src	该指令指定了 Web 应用程序可以加载的字体的域或 URL。
connect-src.	该指令指定了 Web 应用程序可以加载的像 XHR, WebSockets, 以及 EventSource 等脚本接口。

plugin-types	该指令指定了可以哪些 MIME 类型的插件可以被加载（最新尚未有浏览器正确支持）。
form-action	该指令指定了 HTML 表单可以提交的 URLS（最新尚未有浏览器正确支持)。
reflected-xss	该指令告诉浏览器开启或关闭任何用于过滤或阻止反射跨站脚本攻击的启发式算法，这相当于 X-XSS-Protection 响应头的效果（最新尚未有浏览器正确支持)。reflected-xss 相当于 X-XSS-Protection: 1; mode=block

四种源表达式：

源表达式	描述
none	什么也不匹配。
self	仅从当前域匹配，不包含子域。
unsafe-inline	允许内联 JavaScript 和 CSS。你不应该使用这个，除非你确认一个没有清洁过的用户输入数据不会返回到内联。
unsafe-eval	允许使用 JavaScript 的 eval()方法。你不应该使用这个，除非你确定一个没有清洁或危险的用户输入不会插入函数 eval() 。

一个典型的现代 Web 应用程序的正常运行需要 unsafe-inline、unsafe-eval 源表达式和 script-src 指令。

一个 CSP 报头(如，Content-Security-Policy: default-src 'self') 不能适用于大多数现代 Web 应用程序。

这 (default-src 'self') 策略意味着字体，框架，图片，多媒体，对象、脚本以及风格都将从同一个域或源加载，连接将在同源进行。 然而，这对于大多数现代 Web 应用程序是不可行的，因为例如 Web 应用程序可能使用谷歌字体，在一个框架中显示幻灯片分享文档，或者包括加载 Jquery 库或在页面嵌入 Twtter 或 Facebook 的插件脚本。所以开发人员倾向于不使用 CSP，认为这是一种复杂的方式或者用了错误的方式来实现 CSP。

我们总是覆盖 default-src 指令，有效地将其用于实时 CSP。这与我们的需要相适合，但也容易发生 XSS 攻击。

考虑一个典型的 CSP

Content-Security-Policy: default-src 'self'; style-src 'unsafe-inline' 'self' http://fonts.googleapis.com http://themes.googleusercontent.com; frame-src http://www.slideshare.net www.youtube.com twitter.com; object-src 'none'; font-src 'self' data: http://themes.googleusercontent.com http://fonts.googleapis.com; script-src 'unsafeeval' 'unsafe-inline' 'self' http://www.google.com twitter.com http://themes.googleusercontent.com; img-src 'self' http://www.google.com data: https://pbs.twimg.com http://img.youtube.com twitter.com

每一个指令和源表达式的说明

策略	描述
default-src 'self';	允许字体、框架、图片、媒体、对象、脚本以及风格只从相同的域加载。
style-src 'unsafe-inline' 'self' http://fonts.googleapis.com http://themes.googleusercontent.com;	允许使用来自同一个域的内联样式或样式表，http://fonts.googleapis.com and http://themes.googleusercontent.com。

frame-src youtube.com twitter.com;	允许 Web 应用程序只从 youtube.com 和 twitter.com 加载框架。
object-src 'none';	允许空对象。
font-src 'self' data: http://themes.googleusercontent.com	允许 Web 应用程序从同一个域和 http://themes.googleusercontent.com 加载字体。
script-src 'unsafe-eval' 'unsafe-inline' 'self' http://www.google.com twitter.com http://themes.googleusercontent.com;	允许 Web 应用程序从同一个域、http://www.google.com, twitter.com 和 http://themes.googleusercontent.Com 加载脚本。'unsafe-inline' 源表达式允许执行内嵌的 JavaScript，'unsafe-eval' 源表达式允许使用 JavaScript 中的 eval()函数（危险）。
img-src 'self' data: http://www.google.com https://pbs.twimg.com http://img.youtube.com twitter.com	允许 Web 应用程序从同一个域、data URI、http://www.google.com、https://pbs.twimg.com、http://img.youtube.com 和 twitter.com 加载图像。

在 Web 应用开发环境中使用的 XSS 防护机制

JavaScript 的 XSS 防护

有一个用于编码的 JavaScript 库是 Encoder.js。它提供了不同的转义方法：

方法	描述
HTML2Numerical	该方法转义 HTML 实体为其数值当量。
numEncode	该方法编码数值为 unicode 字符。
htmlEncode	该方法编码 HTML 为其他数值或者 HTML 实体。这是由编码类型属性决定的。
XSSEncode	该方法用于编码 XSS 攻击中的基础字符为畸形的 HTML。
correctEncoding	该方法纠正任何双编码的符号。
stripUnicode	该方法移除所有的 Unicode 字符。

文档

http://www.strictly-software.com/htmlencode

下载

http://www.strictly-software.com/scripts/downloads/encoder.js or https://gist.github.com/ajinabraham/1af8216dfb6f959503e0

DOMPurify 是一个 HTML, MathML 和 SVG 的 DOM XSS 净化器。它防止 DOM 攻击和支持白名单.它可以在其他的 JavaScript 框架中使用。

下载

https://github.com/cure53/DOMPurify

Usage:

```
<script type="text/javascript" src="purify.js"></script>
var clean = DOMPurify.sanitize("<p>text<iframe/\/src=jAva script:alert( 3)>");
//becomes <p>text</p>
alert(clean);
```

node.js

Js-xss 是一个转义库。它也包含白名单。

下载

https://github.com/leizongmin/js-xss

XSS 是一个用于转义和净化的 node.js 模块。

Install
$ npm install xss

Usage:

```
var xss = require('xss');
var html = xss('<script>alert("xss");</script>');
console.log(html);
```

jQuery

使用**.text()** 方法替换**.html()** 方法进行转义。 、

yui

使用 **html()** 方法编码 (& < > " ' / ` )。 它也编码反引号字符 ( `) ， IE 将其解释为一个属性定界符。

mootools

mootools 提供 **escapeRegExp()** 方法转义字符串中的正则表达式。

stripScripts() 方法会对标签和其他任何字符串之间的字符进行清理。不要使用 **stripScripts(true)** ，该方法将会在清理前 evaluate 字符串。

漏洞例子

```
1. "<script>alert(1)</script>".stripScripts(true); // This method will trigger the
   alert() function and then strip the string.
```

安全例子

```
1. "<script>alert(1)</script>".stripScripts(); //This method will strip  the
   "<script></script>" tags including "alert(1)".
```

JSON.decode() 转换一个 JSON 字符串为一个 JavaScript 对象，并且检查危险的语法，如果找到危险语法就返回空。

Backbone.js

使用**model.escape()** 转义 HTML

如果你正在使用 Underscore 模板, 使用 **<%-** 替换 **<%=** 进行 HTML 转义。 使用 **<%-string%>** ，不要使用**<%=string%>** 进行 HTML 转义，它会 evaluate 字符串。

Spinejs

spine 中转义的语法应该小心使用。

使用**<%= @string %>** 将会转义和打印字符串。

相反于 Backbone.js， **<%- string %>** 将 evaluate 和打印字符串，返回为进行转义的值。

所以如果你同时使用 JS 框架，不要感到困惑。

AngularJS

对于 XSS 防护, AngularJS 采用严格的语境转义 (SCE)。SCE 也允许白名单。技术上 `$sce` 服务提供了 XSS 防护。我们需要将 `$sce` 服务包含到代码里。 SCE 定义了在上下文中不同的信任。考虑 SCE 中的以下方面：

SCE 服务	描述
$sce.HTML	应用中安全的 HTML 源。ngBindHtml 指令使用此内容进行绑定。如果遇到一个不安全的值，并且 $sanitize 模块当前存在，这将净化值而不是抛出错误。
$sce.CSS	应用中安全的 CSS 源。当前未使用。你可以在你的指令中使用它。
$sce.URL	遵循安全的连接，当前未使用(<a href= he <img src= 净化 URL，并且不会构成一个 SCE 语境。
$sce.RESOURCE_URL	不仅是遵循安全的连接，内容也是安全的，包含在你应用程序中的网址。 例子包括 ng-include, src / ngSrc 绑定标签， 除了 IMG (如 IFRAME, OBJECT, 等等

需要注意的是$sce.RESOURCE_URL 相对$sce.URL 对 URL 做了更为强硬的声明， 因此内容需要的值受$sce.RESOURCE_URL 信任,可以用于任何地方，受$sce.URL 信任的值是必要的。对已 JavaScript，$sce.JS 在你的应用程序中安全的执行，当前未使用。

文档

https://docs.angularjs.org/api/ng/service/$sce

PHP 的 XSS 防护

htmlspecialchars(string, flag, charset) - 该函数只编码（< > " & ）。如果 ENT_QUOTES 标签被赋值，它也编码 (') 为（**'**）。它始终是安全的使用'utf-8'作为字符集。UTF-8 是从 PHP5.4 开始被设置为默认的字符集。该版本也支持一些相对 ENT_QUOTES 的其他新标签，如

标签	描述
ENT_HTML401	处理编码为 HTML 4.01。(默认)
ENT_XML1	使处理编码为 XML 1。
ENT_XHTML	处理编码为 XHTML。
ENT_HTML5	处理编码为 HTML 5。

Example:

```
1. echo htmlspecialchars($string, ENT_QUOTES | ENT_XHTML, 'UTF-8');
```

值得注意的 **htmlspecialchars()** 不能在 JavaScript、风格以及 URL 中防止 XSS。

urlenconde() 可以用于编码 URL。

在 PHP 中使用 **utf8_encode()** 函数编码用户指定的数据为 UTF-8。

如果使用 PHP 函数 **json_encode()**, 回显某些东西，有一个更安全的方式来使用，如
echo json_encode($string, JSON_HEX_QUOT|JSON_HEX_TAG|JSON_HEX_AMP|JSON_HEX_APOS);
连同设置 Content-Type 头为 **application/json; charset=utf8**

HTML Purifier 是一个用于防止 XSS 的理想的 PHP 库，它工作的原理基于白名单。
它是简单和易于配置使用的。

```php
require_once '/path/to/HTMLPurifier.auto.php';
$config = HTMLPurifier_Config::createDefault();
$purifier = new HTMLPurifier($config);
$clean = $purifier->purify($output_to_be_reflected_at_browser);
echo $clean;
```

文档

http://htmlpurifier.org/live/INSTALL

PHPIDS (PHP-Intrusion Detection System)是一个基于 PHP 的应用程序安全层。 IDS 既不清除、净化也不过滤任何恶意输入，它只是识别攻击者是否试图破坏您的网站，然后完全按照您想要的方式作出反应。

更新内容

https://phpids.org/

PHP 模板框架:Smarty

Smarty 提供了变量修饰符的转义。它被用于编码和转义 HTML 内容、url、单引号、十六进制、十六进制实体、javascript 和邮件中变量。缺省情况下是 html。

转义修饰符	描述
{$string|escape,'UTF-8'}	默认:html, 基础 HTML 编码，转义 (&"'<>)
{$string|escape:'html','UTF-8'}	基础 HTML 编码，转义 (&"'<>)
{$string|escape:'htmlall','UTF-8'}	HTML 编码 (转义所有 html 实体)
{$string|escape:'url'}	URL 编码
{$string|escape:'quotes'}	转义引号

{$string|escape:"hex"}	十六进制编码
{$string|escape:"hexentity"}	十六进制实体编码
{$string|escape:'mail'}	转换邮件为文本内容
{$string|escape:'javascript'}	转义 JavaScript, 并且请记住这个实现只支持字符串。

JAVA 的 XSS 防护

使用 OWASP Java Encoder，支持基础 HTML 内容、HTML 正文、HTML 属性、JavaScript 块、URL 参数值、 REST URL 参数、完整的不可信 URL 中用户提交数据的编码。

文档

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project#tab=Use_the_Java_Enco der_Project

Coverity Security Library (CSL) 是一套在 Java Web 应用程序中修正跨站脚本攻击（XSS）的转义方案。它支持 Java 表达式语言（EL）表达法和普通 Java 函数。

转义方法	描述
cov:htmlEscape(string)	执行 HTML 编码
cov:jsStringEscape(string)	执行 JavaScript 字符串编码
cov:asURL(string)	执行 URL 编码和净化危险的 scheme，如**javascript:**
cov:cssStringEscape(string)	执行 CSS 字符串编码
cov:asNumber(string)	检查输入额字符串是一个数值，默认值为 0
cov:asCssColor(string)	允许将颜色字符串指定为文本或者十六进制并且防止注入
cov:uriEncode(name)	执行 URL 编码

文档和下载

https://github.com/coverity/coverity-security-library

OWASP ESAPI (The OWASP Enterprise Security API ）是一个免费、开源的 Web 应用程序安全控制库，可以净化不可信数据。

Method	Description
ESAPI.encoder().encodeForHTML()	转义 HTML
ESAPI.encoder().encodeForHTMLAttribute()	转义 HTML 属性
ESAPI.encoder().encodeForJavaScript()	转义 JavaScript 字符串
ESAPI.encoder().encodeForCSS()	转义 CSS 字符串
ESAPI.encoder().encodeForURL()	转义 URL

文档

https://code.google.com/p/owasp-esapi-java/wiki/Welcome?tm=6

.NET 的 XSS 防护

HttpUtility Class (**System.Web.HttpUtility** ）是.NET 提供的各种方法中用于转义不可信数据的方法。

Methods	Description
HtmlEncode()	HTML 编码。
HtmlAttributeEncode()	基础 HTML 编码，它只编码 (" & < \ )。
UrlEncode()	URL 编码。
JavaScriptStringEncode()	JavaScript 字符串编码。

对于普通的 Web 应用程序来说， **HttpUtlity** 就足够了。但是对于处理和反射数据到 XML 或其他语境的 Web 应用程序来说，可以使用**AntiXssEncoder** 类 (**System.Web.Security.AntiXss.AntiXssEncoder**)，它添加了新的功能如白名单，并且在.NET 4.5 中内置使用。它包括用于 XSS 防护的这些方法：

方法	描述
HtmlEncode()	HTML 编码，并且可选指定是否使用 HTML 4.0 的命名实体。
HtmlAttributeEncode()	编码 HTML 属性中反射的数据。
HeaderNameValueEncode()	编码 header 的名称和值为可以用用于 HTTP header 的字符串。
HtmlFormUrlEncode()	编码用于表单提交的 MIME 类型为 "application/x-www-formurlencoded" 的数据，并选择指定的字符串编码。
JavaScriptStringEncode()	JavaScript 字符串编码。
UrlEncode()	URL 编码并选择指定的字符串编码。
UrlPathEncode()	编码 URL 中使用的路径。
XmlAttributeEncode() & XmlEncode()	编码 XML 属性中使用的数据。

对于旧的.NET 版本，从以下地址安装 Microsoft Web 保护库：
http://wpl.codeplex.com/

Python django 框架的 XSS 防护

函数	描述
{{ string }}	django 有自动转义功能，这将转义字符串。
escape()	该函数将转义（&"'<>）。
conditional_escape()	该函数类似于 escape() 函数, 除了它不能转义转义过的字符串，所以它不能进行双重转义。
urlencode()	该函数可以用于 URL 编码。

Ruby on Rails 框架中的 XSS 防护

Method	Description
sanitize()	这个方法可以用来清洁/HTML 编码用户指定的数据，由白名单提供支持。它也可以分别清洁无效协议的 href/src 标签, 像javascript: 。它会尽可能的应对任何技巧的使用，比如输入 unicode/ascii/hex 值来通过javascript: 过滤器的技巧。
sanitize_css()	该方法将会清洁字符串，你可以在 CSS 中安全的使用它。
strip_links()	该方法将清除一个字符串中的所有链接标签。
h() or html_escape()	这个方法将分别编码（ &<>"' ）为（ & < > " ') 。
html_escape_once()	该方法类似于 html_escape() 。一个不同的地方在于它将编码之前没有编码过的任何东西。

json_escape()	该方法将分别编码 (& > < \u2028 \u2029) 为 (\u0026 \u003e \u003c \u2028 \u2029) 。同时也要记住，这个方法只能用于有效的 JSON。无效 JSON 值使用可以导致 XSS。

进行清洁

你可以通过sanitize()指定一个白名单方法，如

```
<%= sanitize @article.body, tags: %w(table tr td), attributes: %w(id class style) %>
```

这将清洁任何上述的标签和属性。

清除链接

```
strip_links('<a href="http://www.opensecurity.in">OpenSecurity</a>')
# => OpenSecurity

strip_links('Please e-mail me at <a
href="mailto:email@opensecurity.in">email@opensecurity.in</a>.')
# => Please e-mail me at email@opensecurity.in.

strip_links('Blog: <a href="http://www.opsec.opensecurity.in/" class="nav"
target=\"_blank\">Visit</a>.') # => Blog: Visit.
```

开源软件防火墙或 XSS 防护

ModSecurity - https://www.modsecurity.org/
IronBee - https://www.ironbee.com/

免费或开源的 XSS 检测工具

OWASP Xenotix XSS Exploit Framework
IronWASP
Acunetix Free
arachni
ImmuniWeb Self-Fuzzer Addon for Firefox

致谢

Mario Heiderich, Ahamed Nafeez

参考列表

https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
https://www.owasp.org/index.php/List_of_useful_HTTP_headers
https://www.owasp.org/index.php/HttpOnly
https://www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework
https://www.owasp.org/index.php/OWASP_Java_Encoder_Project
https://code.google.com/p/owasp-esapi-java/
http://www.w3.org/TR/CSP11/
https://w3c.github.io/webappsec/specs/content-security-policy/

http://www.html5rocks.com/en/tutorials/security/content-security-policy/
https://tools.ietf.org/rfc/rfc7034.txt
http://msdn.microsoft.com/enus/library/system.web.security.antixss.antixssencoder(v=vs.110).aspx
http://msdn.microsoft.com/en-us/library/system.web.httputility(v=vs.110).aspx
http://openmya.hacker.jp/hasegawa/security/utf7cs.html
http://www.thespanner.co.uk/2013/05/16/dom-clobbering/
http://www.slideshare.net/x00mario/the-innerhtml-apocalypse/46
http://wpl.codeplex.com/ http://opensecurity.in/
http://cure53.de/fp170.pdf
https://www.modsecurity.org/
https://www.ironbee.com/
http://taligarsiel.com/Projects/howbrowserswork1.htm
https://frederik-braun.com/xfo-clickjacking.pdf
http://mootools.net/docs/core/Types/String
http://www.strictly-software.com/htmlencode http://backbonejs.org/#Model
https://www.ng-book.com/p/Security/
https://docs.angularjs.org/api/ng/service/$sce http://spinejs.com/docs/views
https://github.com/cure53/DOMPurify https://github.com/leizongmin/js-xss
http://api.rubyonrails.org/classes/ERB/Util.html
http://api.rubyonrails.org/classes/ActionView/Helpers/SanitizeHelper.html
http://yuilibrary.com/yui/docs/api/classes/Escape.html#method_html
http://prototypejs.org/doc/latest/language/String/prototype/escapeHTML/
http://docs.php.net/manual/en/function.htmlspecialchars.php
http://www.smarty.net/docsv2/en/language.modifier.escape
https://www.superevr.com/blog/2012/exploiting-xss-in-ajax-web-applications/
http://blog.opensecurityresearch.com/2011/12/evading-content-security-policy-with.html
http://www.janoszen.com/2012/04/16/proper-xss-protection-in-javascript-php-andsmarty/
http://wpcme.coverity.com/wpcontent/uploads/What_Every_Developer_Should_Know_0213.pdf