1. **前言**

XSS是大家经常会遇到的一种漏洞，关于其成因大家都很了解，就不费口舌了，在对XSS进行绕过的时候，编码是一种十分常见且有效的手法。在对XSS payload进行编码时，如果对浏览器解码的原理不清楚，可能就会出现将各种编码方式胡乱堆叠的现象，不仅降低了效率，还不会有什么效果。因此本文主要为大家讲解一下关于浏览器解码原理以及如何应用在XSS攻击绕过的方式。

1. **浏览器解码原理**

我们从输入url对网站进行访问开始，假如我们访问如下url：

http://www.test.com/vul.php?name=%3cimg%20%src%3d1%20onerror%3dalert%281%29%3e

虽然这里以php的网站举例，但是对于浏览器来说它的编码原理是不会随着不同的后台语言改变的。

参数name传到服务器后服务器会首先对其进行url解码，这个过程是在服务器进行的，不属于浏览器的工作范畴。随后经过服务器对name参数的各种操作后，其被嵌入到html代码中发送到客户端，由浏览器解析执行。

对于浏览器来说，此时它接收到服务器返回的页面数据中，name的值为url解码后的<img src=1 onerror=alert(1)>。随后浏览器对接收到的页面数据进行解析，解析的过程就是根据html元素构建DOM树的过程，这个过程是迭代的，解析器会不断从词法分析器取各种符号，并尝试用这些符号匹配语法规则，如果匹配到则将其放到有DOM节点与属性节点组成的解析树中。比如对于如下代码来说，其构造的DOM树如图1所示：

<html>

<body>

<p>

Hello DOM

</p>

<div><img src=”example.png” /></div>

</body>

</html>

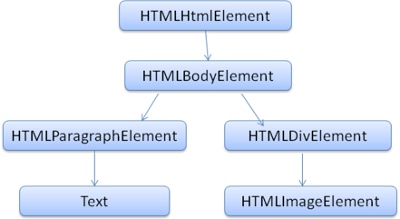
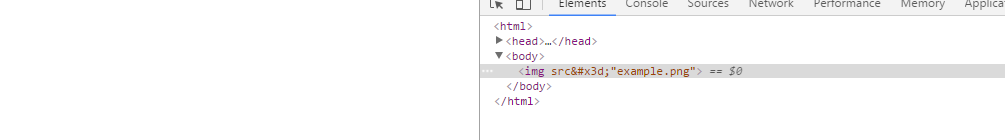


图1

在上面的整个过程中，浏览器不会做任何解码的工作，给它什么节点它就会去匹配什么节点，如果匹配不到就放到一边。因此如果浏览器碰到如下内容是不会识别并放入解析树中的：

<img src&#x3d;’example.png’>



可以看到，浏览器未能成功对该img标签进行解析。

在上述过程结束后，浏览器会首先对构造的DOM树中的元素进行html解码，这个过程中如果遇到<script></script>，浏览器会调用javascript脚本解析器解析javascript，并执行脚本，然后继续解析其他的HTML内容，如果遇到javascript伪协议，onerror等事件处理器需要触发才能执行的事件，只有当触发事件发送时，脚本解析器才会解析其中的脚本，在事件触发之前，它是HTML的一部分，会先进行html解码。

如上所述，在碰到<script>标签时，浏览器会调用javascript脚本解析器解析javascript。Javascript解析器会首先对内容进行js解码，随后渲染执行。而普通html标签href属性中的javascript伪协议以及事件处理器中的内容会在事件得到触发时由Javascript解析器进行js解码并执行。

因此，总结如下：

1. 类似于如下普通html标签或属性中的内容，浏览器会对其进行html解码：

<img src=’{input}’>

1. 对于href属性中的javascript伪协议以及事件处理器中的内容，解码顺序为：HTML解码 --> JavaScript解码

<img src=’example.png’ onerror=’{input}’>

1. 对于<script>标签中有DOM操作的内容，其解码顺序为：

Javascript解码 –> HTML解码

<script>

document.getElementById(‘n’).innerHTML=’{input}’

</script>

当然，在不同的情况下，还存在着更多更复杂的解码顺序的情况，可以根据上述浏览器解码的基本原理在具体环境中推测得到浏览器的解码顺序。

1. **浏览器解码原理在XSS攻击中的应用**

上面介绍了浏览器解码方面的知识，那么在具体的XSS攻击时有什么作用呢？我们举几个简单的例子进行说明：

1. 如果浏览器未能成功识别该标签并将其放入DOM树中进行解析，则无法进行XSS，具体比如：

<img src&#x3d;’example.png’ onerror=’alert(1)’>

&lt;img src="example.png" onerror="alert(1)"&gt;

1. 如果浏览器先对内容做了HTML解码，再对内容做了Javascript解码，那么如果Payalod中先做了HTML编码，再做了Javascript编码，是无法绕过的，比如：

<img src="example.png" onerror="al\u0026#101;rt(1)">

而如下先Javascript编码再HTML编码的方式则可以绕过执行：

<img src="example.png" onerror="al\u&#48;065rt(1)">

这里只给出这两个例子进行说明，更多的绕过技巧还是要在不断实战中总结积累。