0x00:基本介绍
0x01:html 实体编码
0x02:新增的实体编码 实体编码变异以及浏览器的某些工作原理!
0x03:javascript 编码
0x04:base64 编码
0x05:闲扯

0x00 基本介绍

提起 XSS 想到的就是插入字符字符编码与各种解析了!

这也就是各种 xss 编码插件跟工具出世的原因!之前不懂浏览器是如何对我们编码过的代码进行解析的时候就是一顿乱插!

各种编码 各种插 没把编码还原就算了 还原了就算运气好!后来到 PKAV 经过二哥和短短的调教后才算是弄清楚了一点编码与解析方面的知识!

现在也算是运用自如了把!

现在介绍一下在 xss 中最经常用到的编码

html 实体编码(10 进制与 16 进制):

如把尖括号编码[<] ----->html 十进制: < html 十六进制:<

javascript 的八进制跟十六进制:

如把尖括号编码[<] -----> js 八进制:\74 js 十六进制:\x3c

jsunicode 编码:

如把尖括号编码[<] ----->jsunicode:\u003c

url 编码 base64 编码:

如把尖括号编码[<] -----> url: %3C base64: PA==

0x01 html 实体编码

html 实体编码本身存在的意义是防止与 HTML 本身语义标记的冲突。

但是在 XSS 中却成为了我们的一大利器，但是也不能盲目的使用！

html 中正常情况只识别:html10 进制,html16 进制！

现在介绍一下我们应该如何在 xss 过程中灵活的使用各种编码呢？

比如现在你的输出点在这：

<img src="[代码]">

在这里过滤了 script 〈 〉 / \ http: 以及各种危险字符 比如创建一个 html 节点什么的！

有的站只允许你引用一个 img 文件夹里的图片 但是图片是你可以控的 可以通过抓包来修改的！

我们如果想加载外部 js 或者一个 xss 平台的钩子我们应该怎么写呢？

那么我们可以在这里 闭合双引号 写事件: onerror=[html language="实体编码"][/html][/html]

比如我现在弹个窗:

<img src="x" onerror="alert(1)">

原 code:

<img src="x" onerror="alert(1)">

这里我用的是 html 十进制编码 也可以使用十六进制的 html 实体编码！

但是为什么这里我没有用 jsunicode 以及 js 八进制跟 js 十六进制呢！

浏览器是不会在 html 标签里解析 js 中的那些编码的！所以我们在 onerror=后面放 js 中的编码是不会解析 你放进去是什么 解析就是什么！

大多数网站是不会&#号的，如果过滤了怎么办呢？

那么再来讲一下另外一个案例：

源码如下：

```
view-source:x[redacted]et=&searchtype_yjbg=yjjg&s
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
<title>研究报告_证券频道_腾讯网</title>
<style>
body{font-size:12px;}
.main ul{list-style:none;}
.ffy{text-align:left;}
.clear{clear:both;}
.fanye{height:30px;line-height:30px;}
.fanye li{float:left;margin-right:5px;display:inline;font-family: Tahoma;color:
.fanye li a{color:#333;}
.fanye li a:hover{color:#cc0000}
.fanye li .yebg{width:34px;background:url(./img/bt.gif) no-repeat 0px 6px;text-
#pagenum{width:50px;}
</style>
</head>
<script>window.onerror = function(){return true;};</script>
<body>
<div class="main">
        腾讯新闻:</br>
        <ul>
        <li><a href="#">证券报道系列之 0</a></li><li><a href="#">证券报道系列之 1</a>
道系列之 3</a></li><li><a href="#">证券报道系列之 4</a></li><li><a href="#">证券报道系
href="#">证券报道系列之 7</a></li><li><a href="#">证券报道系列之 8</a></li><li><a hre
</div>
<div class="ffy clear">
        <div class="fanye">
                <ul>
                        <li><div class="yebg">首页</div></li>
                        <li><div class="yebg">上页</div></li>
                                        <li><div class="yebg"><a href="
下页</a></div></li>
                        <li><div class="yebg"><a href="./3.3.php?offset=0&searc
                        <li>0/0</li>
                        <li class="lh0814">转到第</li>
                        <li><input type="text" id="pagenum"  class="inputstyle0
location.href='./3.3.php?offset='+this.value+'&searchtype_yjbg=yjjg&searchvalue_
                        <li><div class="yebg"><a href="javascript:location='./3
offset='+document.getElementById('pagenum').value+'&searchtype_yjbg=yjjg&search
                </ul>
        </div>
</div>
</body>
</html>
```

页面中的 Go 按钮中包含一个 a 标签 输入的值会存在于 a 标签的 href 属性中，
href 中用了 javascript 伪协议，可以在 href 跳转时执行 js 代码！

所以造成了 xss！

我们提交的值如下：

wooyun%26%23x27,alert(1)%2b%26%23x27

由于页面对单引号 & 符号 以及 #符号过滤!但是 html 中可以识别 html 实体编码！但是实体编码是由&#组成！

这个时候&#已经被过滤 我们只能通过 url 编码来对 & # 两个符号进行编码！再让浏览器解码成 &# 然后拼接 x27 最后就成为了单引号的 html16 进制编码！

解码后：我们的提交值为：

',alert(1)'

href 代码为：

GO

ps:在之前说了 html 标签中识别 html 实体编码，并且会在 html 页面加载时会对编码进行解码！那么&#x27 已经是单引号了 但是并不会闭合！ 然后在点击过程中执行 javascript 代码 这个时候由于 html 里&#x27 被解析成单引号但是没闭合 这个时候 js 被执行 这个我们提交的在 html 加载时解析成了字符串单引号但是不能闭合之前的引号 因为现在是把我们提交的编码了的单引号 当成字符串来显示 但是现在他是存在于 a 标签中的 href 里的 href 链接里的地址是 javascript 伪协议，我们现在点击的时候 会执行里面的代码 关键来了 这个时候我们之前被当做字符串的单引号 被再次解析 这个时候就没任何过滤规则来过滤它 程序也没那么智能 之前当做字符串的单引号起作用了 javascript 不知道他是个字符串 它只知道浏览器解析成了什么 他就带入进去!就在这个时候我们的字符串单引号就成功的闭合了！当点击 go 时 我们的代码执行！

上面这个例子讲了 html 编码 以及特殊情况下的编码那么再讲下当你的输入点存在于 script 标签中的时候！我们就应该用 js 中的编码了！

既然知道是如何解析的了 那么便又有了以下新的想法！

0x02 新增的实体编码，变异以及浏览器的某些工作原理

通常程序做 XSS 防御的时候会考虑到一些 HTML 编码的问题，会拦截或转义 " \ 这样的东西 那么我的双引号跟尖括号就被拦截了！

但基础这种黑名单方式可能出现的问题：

1. 不认识 HTML5 新增的实体命名编码,如

: => [冒号]

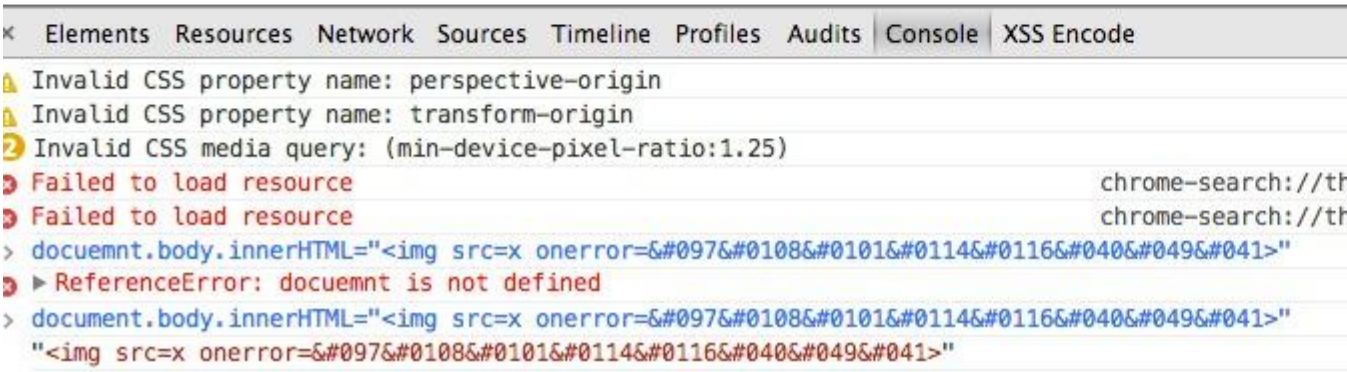
 => [换行]

case: <a href="javasc
ript:alert(1)">click

2.对 HTML 编码的解析规则不够熟悉,就像十进制和十六进制编码的分号是可以去掉的。

还有，数字编码前面加「0」，这也是一条很好的绕过 WAF 的向量。

如下图（我去掉了后面的分号 另外在每个数字前加了一个零）：



数字前面是可以加多个 0 的 闲的蛋疼的基友可以自己试下！

<a href="javasc
ript:alert(1)">click

这句代码能够执行么？

不知道那些不是很清楚浏览器工作原理的基友，在最开始有没有怀疑这段代码能不能执行！

起码我最开始 怀疑过！即使编码被解析回来了 换行了还能执行么！

当时就去问了我的好基友 XX 大神[一位跟短短一样拥有着跟常用不一样的思维在我们看来是很不同于正常人的人]

然后大神给了我一份比较详细的浏览器工作原理 很长很长！

我就把最主要的 copy 下来贴上吧！

解析器－词法分析器 Parser－Lexer combination

解析可以分为两个子过程——语法分析及词法分析

词法分析就是将输入分解为符号，符号是语言的词汇表——基本有效单元的集合。对于人类语言来说，它相当于我们字典中出现的所有单词。

语法分析指对语言应用语法规则。

解析器一般将工作分配给两个组件——词法分析器（有时也叫分词器）负责将输入分解为合法的符号，解析器则根据语言的语法规则分析文档结构，从而构建解析树，词法分析器知道怎么跳过空白和换行之类的无关字符。

然后我的理解是这样的：

<a href="javasc
ript:alert(1)">click

首先 html 编码被还原出来 然后就成了换行 跟冒号

<a href="javasc
ript:alert(1)">click

为什么换行后还能够执行 是因为浏览器中的解析器中词法分析器 起的作用会跳过空白跟换行之类的无效字符。

然后就构造成了一个完整的语句

click

代码执行!

看完那些之后瞬间心里觉得原来跟原理性相关的东西真的很重要！能够让你写 xss payload 更加灵活!

0x03 javascript 编码

---

javascript 中只识别几种编码：Jsunicode js8 进制 js10 进制

就拿下面这个例子来讲吧!

第一种情况 你输入的值存入某个变量 然后最后出现在某个能把字符串当做 js 代码来执行的函数里!

如：

eval()　　setTimeout()　　setInterval()

以上都是会将字符串当做 js 代码执行的函数！ 如果是以下情况：

```
var search = "可控点";
document.getElementById().innerHTML=search;
```

以上情况很多都是出现在你搜索后 然后显示的 你所查询的关键字

如果过滤了 <> ’ ″ & % 等等这些！然后再输出到页面上！

按理说这样是安全了！但是我们把输入的值改成 jsunicode 编码

如 我们改成〈img src=x onerror=alert(1)〉然后进行 js 八进制编码　　然后服务器端接受后 经过过滤器 没有发现该过滤的就进入到了 innerHTML 中

现在我们来看看 输出是什么效果！

我就用 chrome console 来演示吧！

看到了把 经过 js 的解码 我们的代码又还原回来了 并且注入到了网页中！这时候代码执行！成功弹窗！

在 js 中是可以用 jsunicode js16 进制 js8 进制的！

为什么这里不用 16 进制 跟 unicode 编码！是因为 八进制的相对而言最短！

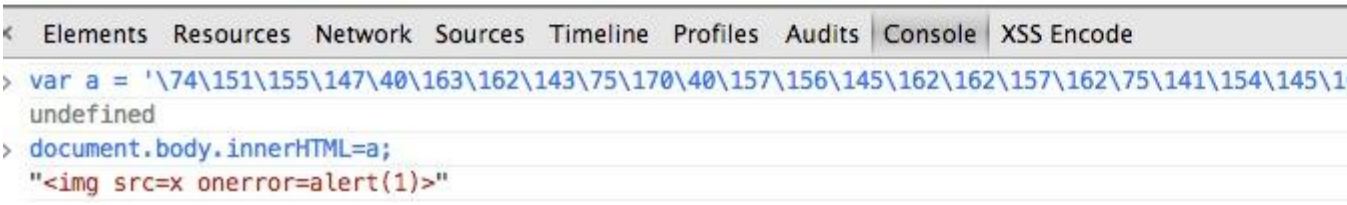
在 xss 中字符数的长短 也是一个很重要的问题！越短越好！

在 asp 的站中插 XSS 代码的时候，存储型 会因为你数据库中字段的长度不够

而存不进去 然后报错！这种情况经常发生！所有养成用最少的字符 来达到你的目的 是最好的！

既然提到了 js 中的十六进制编码 跟 js 中的 unicode 编码 那么也上两张图吧！

十六进制在 js 中是\x[16hex] 来表示的 如：< \x3c

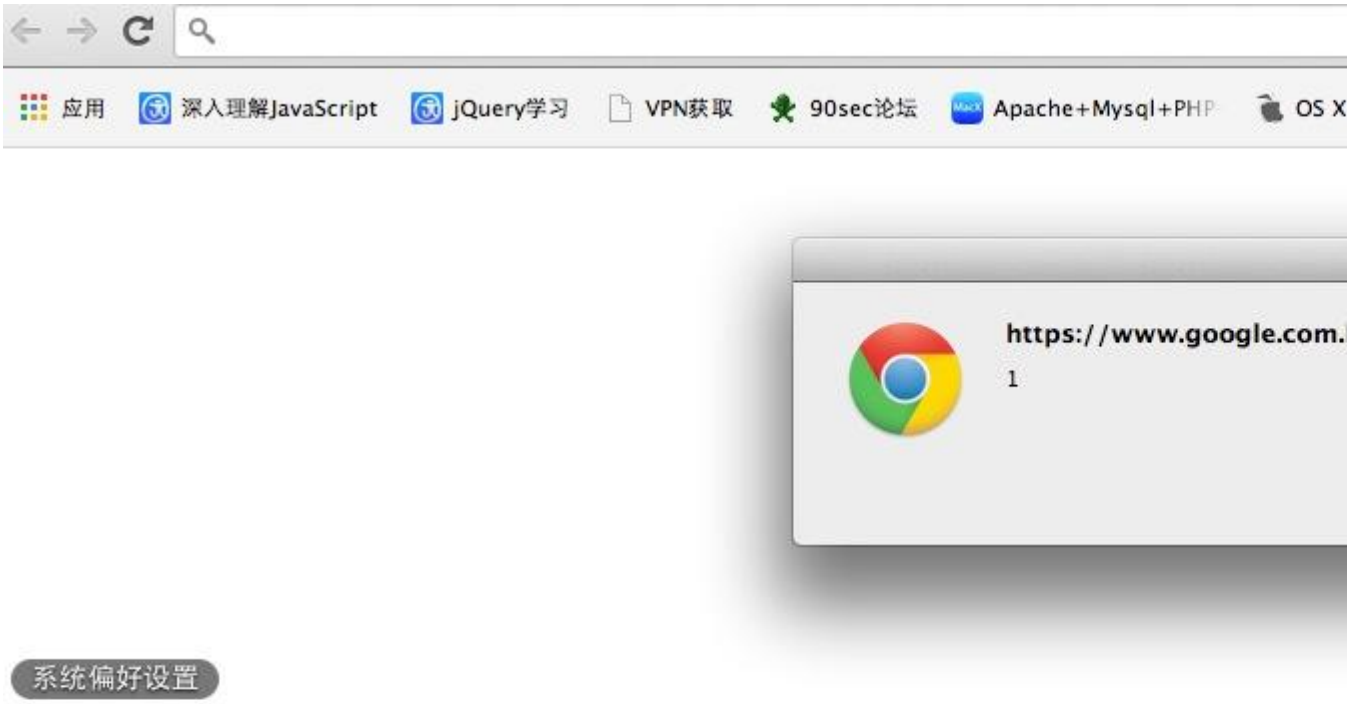


大家看到跟八进制的用法也是一样的！只不过多了一个字符 X 虽然我很喜欢这个字符 但是我更喜欢八进制的短小精悍！

下面再说说 jsunicode 编码：

他的表示方式是这样的：\uxxxx \uxxx < 转码后：/u003c

上图：



0x04 base64 编码

到目前为止 我遇到使用 base64 编码的情况 大多数是这样！

<iframe src="可控点">

在这种情况下 如果过滤了<> ’ ″ javascript 的话 那么要 xss 可以这样写 然后利用 base64 编码！

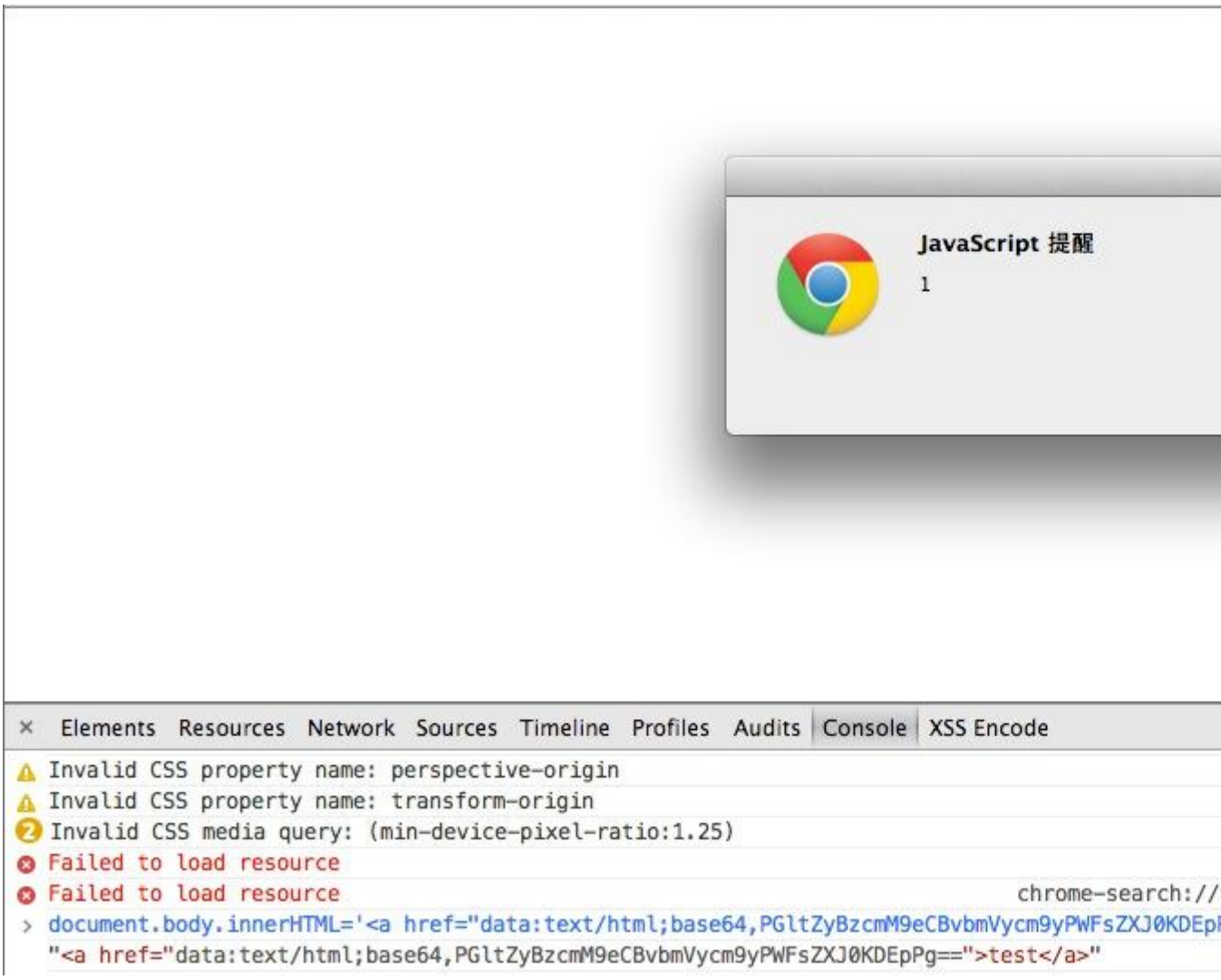
<a href="data:text/html;base64,
PGltZyBzcmM9eCBvbmVycm9yPWFsZXJ0KDEpPg==">test

这样当 test A 链接点击时 就会以 data 协议 页面以 html/text 的方式解析 编码为 base64 然后单点击 a 链接时 base64 的编码就被还原成我们原本的

<img src=x onerror=alert(1)>

然后成功弹窗！

如下图：



在 iframe 里也同样可以使用！大家自己测试把！

0x05 闲扯

---

web 前端的世界真的太难让人捉摸透！

其实好多阻挡在面前的就是那些原理性的东西！懂了就好了！

看着二哥挖掘反射型 xss 让我感觉很像 js 代码审计！

用二哥的话来说，首先你 js 的功底得比写 js 的人功力要高 然后你就比较容易挖掘到 xss!所以我感觉 js 比 xss 来说相当重要！二哥一直写了六七年的 js 才练就今天的功力！所以我特别坚信这句话！

一般测试 xss 首先我会先测试一些反射型！提交几个非法字符 然后看过滤成什么了！然后再打开 chrome console 然后再追踪当前网页中 以及网页所引用文件中的一些关键字！比如这个输入框的 id =xxxx 然后我就一直追踪下去！如果某个数组 或者变量 的值把它传入进去了 然后就一直追踪下去 一直找到源头！把代码都看一边 如果能绕过的话 自己在 chrome 稍稍调试下就能挖掘出来了！

当然这种方法也是二哥教！二哥跟短短一直都是我膜拜的对象！短短跟我讲了讲浏览器的解析，二哥讲了 dom 的渲染 js 的解析以及等等 说不完的一些技巧！

顿时赶脚 挖掘 XSS 是那么的容易！