10.1.2 迟到的 CSP 策略

前面我们提到 Web 前端混乱局面，比如 IE 下的 CSS 的 expression 可以写 JavaScript，再如，HTML 的标签<script>、标签 on 事件、标签 style 属性、标签 src/href/action 等属性都可以内嵌 JavaScript 执行。为什么没有很好地分离？HTML 仅做 HTML 的事，JavaScript/CSS 都通过加载独立文件的方式被执行。如果这样分离，我们不用担心 HTML 中直接出现 JavaScript 的风险，而且 JavaScript/CSS 独立文件所在的域可以配置为白名单，这样就能有效地防止加载攻击者域上的相关资源文件。这就大大提高了 XSS 攻击的难度，这就是 CSP 策略的最大设计初衷。

CSP 策略使得 Web 前端更有序，从而更安全，这是一个好趋势，W3C 已经在大力推进这样的策略（http://www.w3.org/TR/CSP/）。Firefox 与 Chrome 已经开始支持，IE 10 也会开始支持，以下描述的是一种按标准实现理想的状态，经过我们测试，有些标准并没有严格实现，导致不该出现的风险还可能出现。

> 注：
>
> 目前，Chrome 支持 CSP 策略的头部是 X-WebKit-CSP，而不是标准的 X-Content-Security-Policy，但是具体策略都一样，以下统一使用 X-Content-Security-Policy 进行描述。

CSP 策略由一些指令构成，每个指令以分号（;）分隔，语法格式如下：

```
X-Content-Security-Policy: [指令 1] [指令值 1] [指令值 2]; [指令 2] [指令值 1] ...
```

目前 X-Content-Security-Policy 包含的指令及描述如表 10-1 所示。

表 10-1　X-Content-Security-Policy 指令

指　令	描　述
default-src	该指令的值会影响以下所有的指令，支持通配符来表明外部资源的来源（orgin），比如： ① 可以用*表示允许所有的来源； ② 用*.foo.com 表示来源 foo.com 的所有子域内容；

续表

指　令	描　述

default-src	③ https://foo.com 表示来源 https 协议下的 foo.com。 另外，还支持特殊的指令值（以下单引号必须有）： ① 'none'表示一个空集合，没有任何来源匹配到，即外部资源不被允许加载； ② 'self'表示匹配同域内的资源，即只有同域内的资源允许被加载； ③ 'unsafe-inline'表示允许内嵌的 JavaScript/CSS，如<script>里的、javascript:里的、on 事件里的、<style>里的等，默认是不被允许的； ④ 'unsafe-eval'表示允许 eval/setTimeout/setInterva/Function 等可以直接执行字符串的函数。 除此之外，还有一个 data 指令值，允许 data:协议。 注意：以上指令值都以空格分隔
script-src	表示脚本来源，指令值同 default-src
object-src	表示<object><embed><applet>等对象的来源，指令值同 default-src
img-src	表示<img>来源，指令值同 default-src
media-src	表示<audio><video>的来源，指令值同 default-src
frame-src	表示<frame><iframe>的来源，指令值同 default-src
font-src	表示@font-face 字体的来源，指令值同 default-src
connect-src	表示 XMLHttpRequest、WebSocket 等跨域的来源，指令值同 default-src
style-src	表示样本来源，指令值同 default-src

除了以上常规指令外，还有一个特殊指令 report-uri，用于将违法 CSP 策略的告警信息传输给 report-uri 指定的地址中，这样可以有效地评估目标页面的 CSP 策略动态，比如攻击者尝试的 XSS 攻击。

针对以上指令，下面举几个应用 CSP 的场景。

场景一：不允许任何外部的资源加载，且允许内嵌脚本执行。

响应头如下：

```
X-Content-Security-Policy: default-src 'unsafe-inline' 'self'
```

在这样的场景中，当进行 XSS 攻击时，就无法注入远程的 js 文件。

场景二：仅允许白名单的外部资源加载，不允许内嵌脚本执行。

响应头如下：

```
X-Content-Security-Policy: default-src *.foo.com
```

在这样的场景中，当进行 XSS 攻击时，要想成功，只能注入<script>对象，并加载攻击者可控的*.foo.com 白名单下任意脚本文件，比如，一些 JSON callback 文件（如果可以注入任意 JavaScript 的话）。

这个场景还可以对外部资源进行细分，比如图片、Flash、样式、脚本等文件，这样的场景保证了 HTML 仅做 HTML 的事，脚本逻辑等都放在了独立的资源文件中，是一种非常漂亮的分离策略，这样的策略除了自身很优美之外，还大大提高了前端安全性。

当 CSP 策略开始普遍流行的时候，跨站师们就开始头疼了，XSS 攻击会因为前端的有序化而逐渐失去光彩，只是这需要时间。很多网站要做好 CSP 策略的兼容，也是需要时间与精力去改变的，但这绝对会是一个趋势。