# Model-based Security Testing: an Empirical Study on OAuth 2.0 Implementations

报告人：徐梓桑

# 提出问题

- 由于目前关于OAuth相关漏洞的普及，因此，对于OAuth2.0的相关测试越来越受到关注。

- 然而，这些测试工作依赖于手工对OAuth 2.0进行测试，从而发现新的漏洞，或者对于在OAuth中已经被发现的漏洞，在OAuth 2.0中进行自动测试。

# OAuth

- OAUTH协议为用户资源的授权提供了一个安全的、开放而又简易的标准。与以往的授权方式不同之处是OAUTH的授权不会使第三方触及到用户的帐号信息（如用户名与密码），即第三方无需使用用户的用户名与密码就可以申请获得该用户资源的授权，因此OAUTH是安全的。

# OAuth

- 1. 用户访问app，并尝试用IdP登录app
- 2. 应用程序将用户重定向到IdP进行认证，并从步骤2到步骤5绑定一个可变参数STATE。
- 3. 用户使用IdP进行身份验证，然后授权应用程序请求的权限
- 4. IdP向用户返回包含STATE参数的授权码（通常为cookies的哈希函数和一个随机数）
- 5. 若接收到的STATE参数与步骤2中的STATE参数不一致，则用户被重定向到app会拒绝请求的重定向终端。

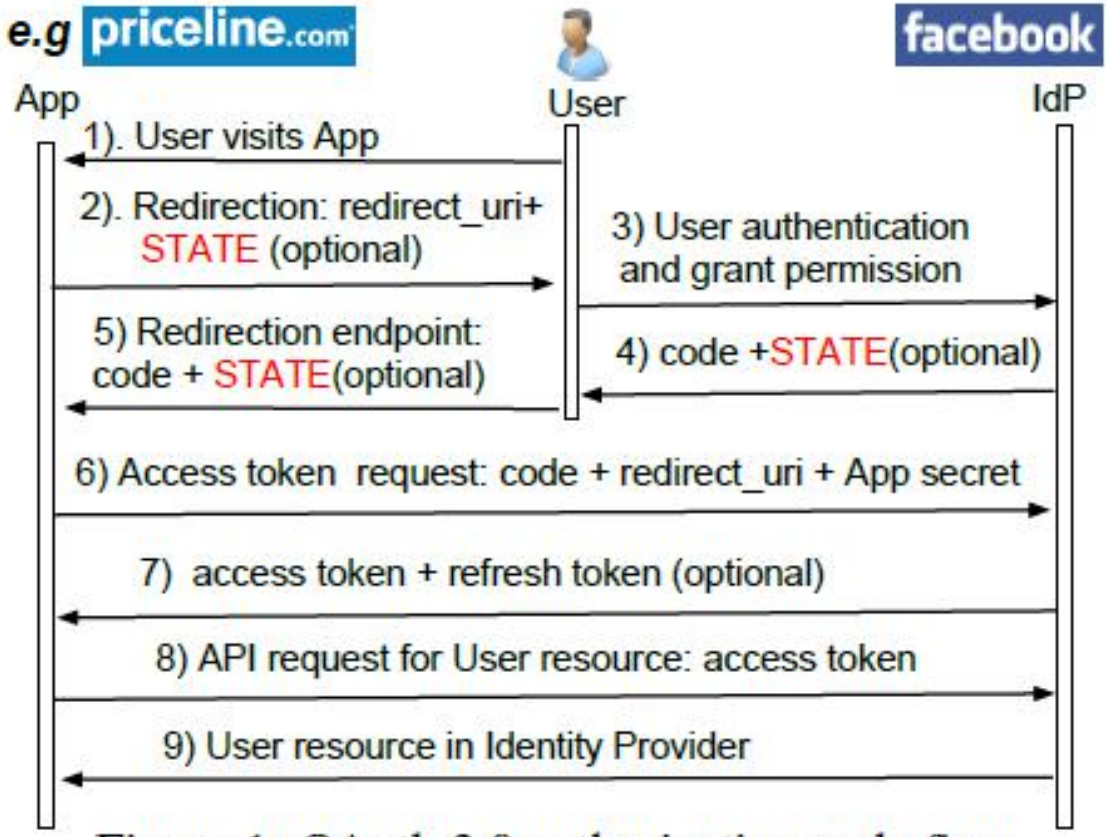


Figure 1: OAuth 2.0 authorization code flow

# OAuth

- 6. App通过将代码及其App秘密发送给IdP来请求访问令牌
- 7. 在检查代码的有效性和应用程序的身份之后，IdP回复一个访问令牌
- 8. App通过访问令牌请求用户数据
- 9. IdP回复用户数据（例如配置文件）给app，以便app可以承担用户的身份并且让用户登录

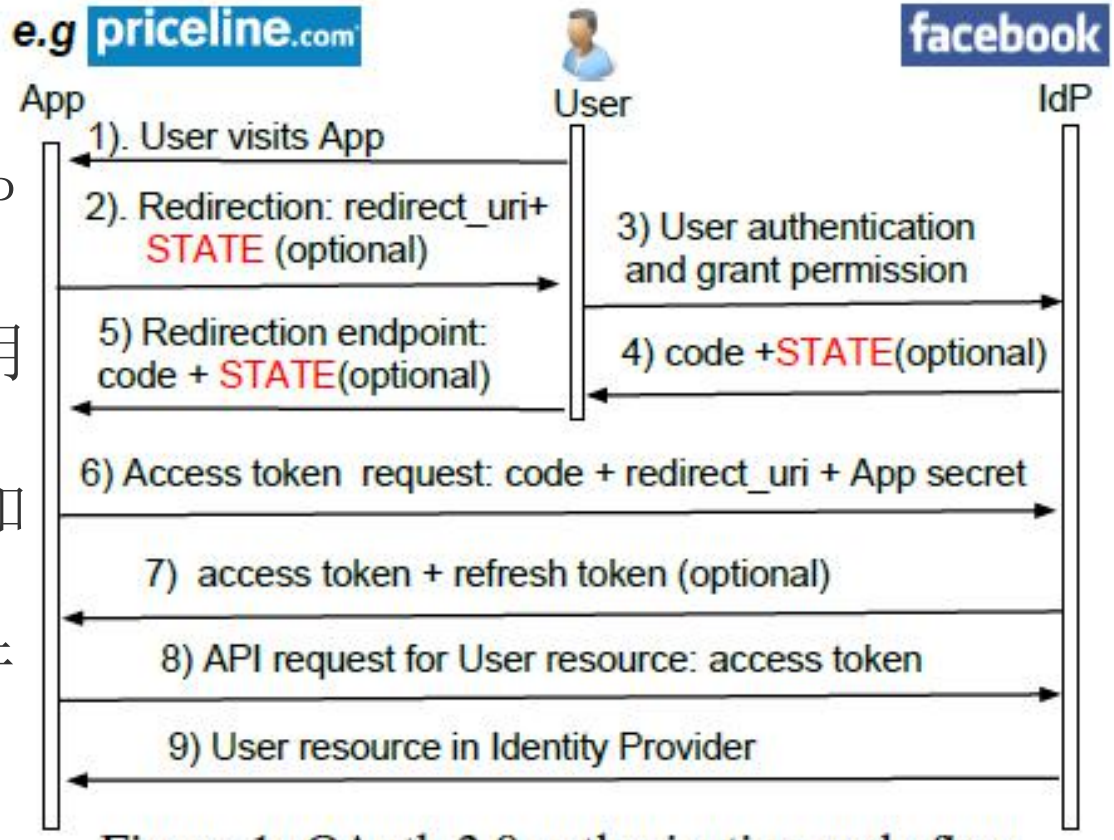


Figure 1: OAuth 2.0 authorization code flow

# 本文摘要

- 本文提出了一种基于自适应模型的测试框架来实现对OAuth 2.0的自动化及大规模安全评估。本文优点在于：
  - 自动识别现有漏洞并发现新漏洞的能力；
  - 对测试覆盖范围进行了改进——将检查模型范围内的所有可能的执行路径；
  - 其在不同的实际OAuth系统/应用中，具有自适应性，使分析者能够在对OAuth进行大范围测试时，减少手工操作负担。
- 本文设计并实施了OAuthTester来实现其提出的框架。

# 威胁模型

- 本文主要关注app和IdP的逻辑缺陷，因此考虑攻击者(Eve)可以窃取正常使用者(Alice)的加密信息与app或IdP交换信息。

# OAuthTester的系统架构

- 1.基于OAuth的规范，本文人为的定义了一个粗粒度系统模型（coarse-grained system model），然后通过从网络跟踪中识别关键参数自动初始化。该模型定义及抽象了IdP和app的正常行为。
- 2.在给定了系统模型之后，OAuthTester由用户输入格式自动生成测试集。
- 3.对测试集进行修改后（修改方式在4.2节中详细描述），Test Harness将可执行的测试集发送给IdP和app。
- 4.OAuthTester收集从IdP和app的响应信息。
- 5.Test Oracle将实际系统响应与系统模型中预定义的预期行为进行比较，并确定响应是否正常。

# OAuthTester的系统架构

- 6. 对于非正常现象，OAuthTester将判断其是否可能导致出现漏洞，如果可能，将测试集列出来，进入步骤8，否则进入步骤7
- 7. 我们需要重新整理或纠正系统模型。
- 8. OAuthTester从收集到的响应中提取出有用的信息，例如访问令牌的值，STATE参数及其安全属性等。然后将该信息加入系统模型的信息池。
- 基于最新得到的信息，我们可以采用特殊的措施来精炼和自定义系统模型，如4.4节中所述。我们继续此迭代过程，直到OAuthTester覆盖了状态机的每个路径。

- 本文在OAuthTester中设计并实现了三个模块，System Model,Test Harness and Test Oracle，在图2中表示为A，B，C。
- System Model（模块A）以状态机的形式呈现，其抽象及描述了OAuth的系统表现。基于此模型PyModel可以生成测试集。Test Harness（模块B）修改并执行测试集。最后Test Oracle（模块C）将系统实际表现与预期值进行比较，并确定系统是否存在漏洞。

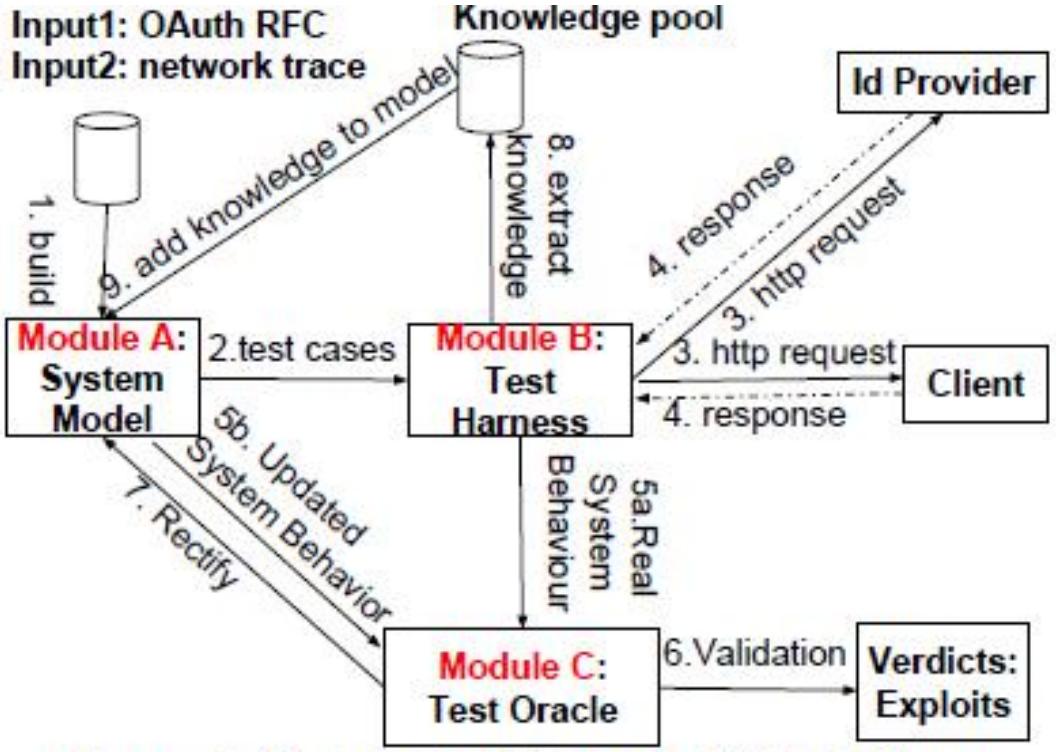


Figure 2: System Architecture of OAuthTester

# Define the State Machine with the Specification

- 本文对各个状态进行了定义，也定义了相应的状态转换来完成状态机。

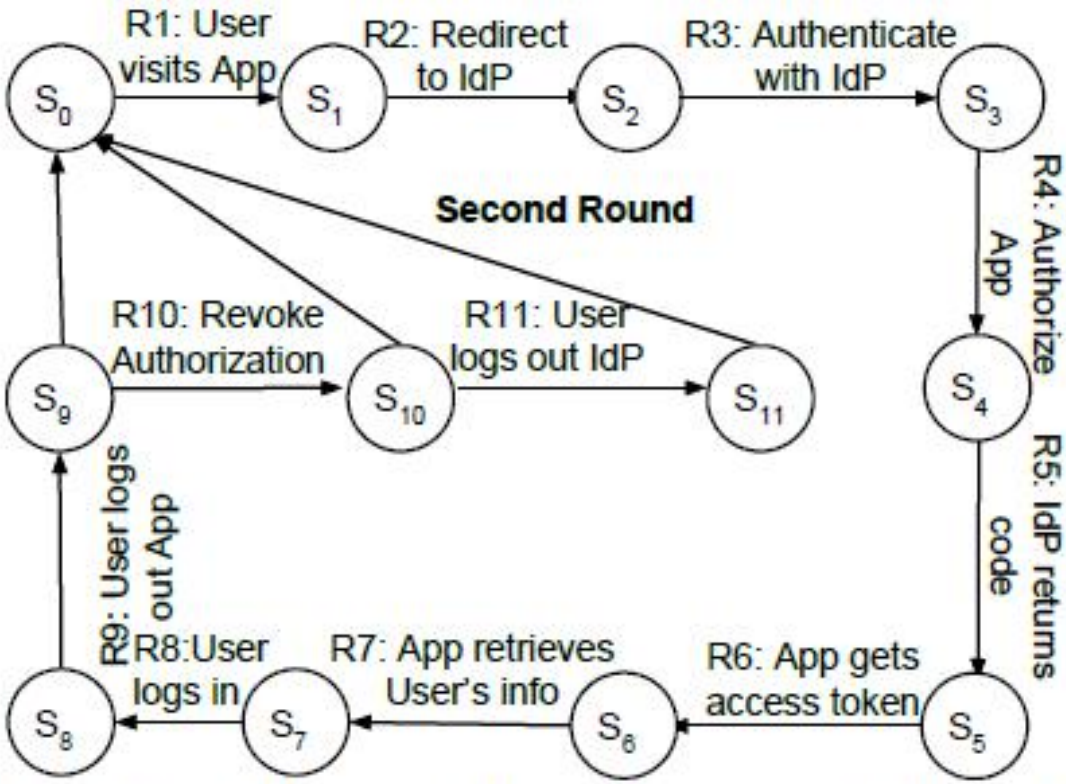


Figure 3: State Machine of Normal Operation of A Single User

- [1]: We do not consider State $S_6, S_7$ afterwards, as they occur in the server side of the application.
- 2: This figure is another representation of Fig. 1. We often exchange these two figures in the rest of this paper.

# Initialize the State Machine with the Network Trace

- 将请求发送到服务器，并将真实响应与预期响应进行比较。
- 红框内的参数可能并不一定会对安全产生影响。本文重新构造此请求，对其进行测试。
- 最终测试结果表明，api_key和next是关键参数

**R3**:https://www.facebook.com/login.php?skip_api_login=1&api_key=127621437303857&signed_next=1&next=https://www.facebook.com/v2.0...&state=71e958b3edc02fb0368c81e4d71917a0#_=_&display=page

**R4**:https://www.facebook.com/v2.0/dialog/oauth?redirect_uri=http://imgur.com/signin/facebook&state=71e958b3edc02fb0368c81e4d71917a0&scope=email&client_id=127621437303857&ret=login&ext=1422535973&hash=AebXSOM95eUr5q5t

**R5**:http://imgur.com/signin/facebook?code=AQD7...7PMO&state=71e958b3edc02fb0368c81e4d71917a0#_=_

Figure 4: Part of traces for the running example

# 测试过程

- 利用上述系统模型，PyModel[1]可以自动生成测试集以覆盖OAuth的每个执行路径。测试集由状态转换（action）和由此操作产生的预期状态（state）表示。例如，图4中，R4的测试集为
  - *action R4=[Alice, https, facebook.com/v2.0/dialog/oauth, GET, redirect_uri, state, scope, client_id];*
  - State $S_4$ = *[Alice_Status, Eve_Status, App_IdP, knowledge_pool].*

[1]an opensource project based on NModel [23], which takes a system model as input and output test cases.

# 模糊关键参数

- action由PyModel生成，Test Harness首先确定要被模糊化的参数。
- 执行该模糊动作后，系统状态可能维持在S3或者进入到S4阶段，对于前者，Test Harness不断地模糊动作R4的关键参数。对于后者，PyModel将继续生成要模糊的下一状态转换。
- 根据不同参数的安全属性，本文根据下表采用不同的模糊策略。

Table 1: Properties and its Correspondig Fuzzing Scheme of Key Parameters

| Propety | | Fuzzing Scheme |
|---|---|---|
| constant | | Compare the values between different sessions and users |
| variable | mandatory parameter | Remove this parameter and randomize its values |
| | used for once or multiple times | Substitute the value with an used one and compare the response |
| | user-specific | Substitute the value with a fresh one of another user |
| | session-specific | Open a new browser and get a fresh value of this parameter to substitute the existing one |

# 打破执行顺序

- 根据本文的威胁模型，我们允许攻击者无论目前的状态如何，都能发起任何状态转换。

- 攻击者构造不合逻辑的状态转换方式如下：
  - 1）根据当前的状态，获取可能启用的下一状态；
  - 2）当当前状态为waypoint时，通过利用规范的知识将系统回滚到先前的状态；
  - 3）尝试从先前状态到达原来启用的下一状态。

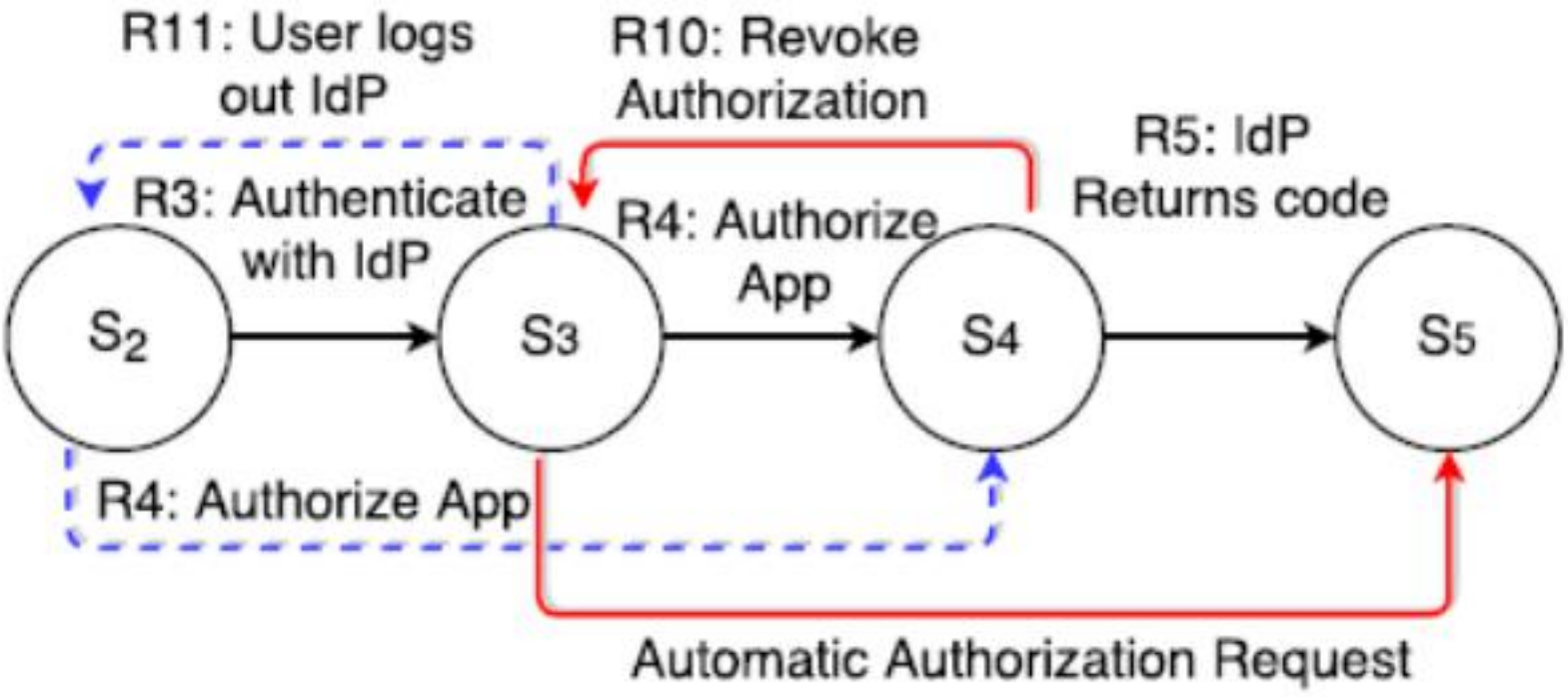


Figure 5: Design Request Sequence to Break Execution Ordering

- 应用程序在S4状态已经被授权，因此IdP可以向App发出一个代码作为授权结果。但是，攻击者通过撤销Request R10的授权将系统回滚到先前的状态S3。正常情况下，从S3到S5是不可能的。然而，此时，攻击者将被自动授权，并可尝试在未经用户许可的情况下到达状态S5。

# Test Oracle

- 通过比较实际和预期的系统行为（即预期状态），测试人员可以确定系统是否正常。若出现偏差，则意味着系统模型存在漏洞或不正确的定义。为了验证这种不符合是否会导致系统不安全，我们检查以下三个安全属性：
  - 认证：攻击者可以获取诸如访问令牌，代码或会话ID等信息，让App/IdP认为他是受害者。或者他可以像正常用户一样登录应用程序。
  - 授权：攻击者可以绕过授权机制或者和得到授权阶段一样，能获取任何想要的信息并且进行任何操作。
  - 关联：OAuth的目标是正确绑定三条数据：用户的身份，用户的权限和会话的身份。这个关联具体是什么，取决于应用程序对用户的识别和用户的权限。

# Iteratively Refine the System Model

- 系统精炼由两个主要组成部分组成：1.更新关键参数的属性；2.纠正IdP的实施。（1. update the properties of key parameters; 2. rectify the implementation of IdP）

- 首先，OAuthTester可以通过观察OAuth entities的响应来自动更新关键参数的属性。

- 对于后者，由于不同的IdP会为自己的业务和服务添加新功能，而不考虑RFC6749的定义。则对于同一个参数可能存在不同的属性，OAuthTester可以在该参数的属性更改后提示警告。

# 实测效率

- 由于每次点击按钮或发送HTTP请求都涉及到具有IdP/App的高延迟往返功能，所以减少测试用例数量，使得在合理的时间内完成测试是很重要的。因此，我们利用以下启发式方法在有限的时间内发现尽可能多的潜在安全漏洞：

- （1）对于在应用程序之间共享的那些功能，我们只测试一次（例如只有一个应用程序）。
- （2）我们首先检查应用程序以前已知的漏洞，以尽快找到应用程序的所有潜在漏洞。
- （3）我们主要关注waypoint来构造无序请求。（We mainly focus on the waoit to construct out-of-order requests.）

# 检测精准

- 一般来说，OAuthTester不会报告任何误报，因为我们将检查每个状态转换后系统是否转换到预期状态，并重新验证任何异常行为的安全属性。但也是存在出错的可能性的，因为：

- 1.我们只能保证由状态机定义的所有路径，而不是真实的情况，如果模型太粗糙，可能会漏掉真实系统中的一些漏洞。
- 2.我们只关心SSO流程，而不管被测应用程序被授权后的后续操作。
- 3.OAuthTester仅查找关键参数的原始格式或其值，一旦关键参数被进一步模糊化，就无法发现任何漏洞了。

# 实验结果

- 本文使用OAuthTester检测了中美500个排名最高的网站，包括Facebook，新浪，人人网和腾讯微博。通过OAuthTester，本文成功地检测到许多现有的安全漏洞。

# 发现的漏洞1：滥用STATE参数

- 滥用案例如下：
- 缺乏STATE验证：STATE参数未被App验证。
- 宽松的STATE验证：如果有一个STATE参数，应用程序可以正确验证它。但若没有，App也接受该请求。
- STATE不绑定到用户：应用程序假定自己生成的所有STATE参数是有效的，并且无法检查此参数是否与用户绑定在一起。因此，攻击者可以用自己伪造的请求替换受害者的STATE参数。

- STATE重播：STATE参数可以重复使用多次。根据STATE参数的有效性，存在三种情况。
  - 多次使用直到下一次登录：STATE参数一直保持有效，直到受害者再次尝试登录应用程序时，才会产生一个新的STATE值。
  - 一个浏览器的相同状态：只要用户使用相同的浏览器，相同的用户使用相同的STATE值。
  - 常量STATE参数：参数在不同会话和用户之间保持不变

Table 2: Statistics of STATE Parameter Misuse[1] [2]

| IdPs (NO. of Applications) | Lack STATE Validation | Lenient STATE Validation | STATE Not Bind to User | STATE Replay | | | Summary |
|---|---|---|---|---|---|---|---|
| | | | | Multiple Use Until Next Login | Same STATE for One Browser | Constant STATE Parameter | |
| Facebook (79) | 12.50% | 14.29% | 21.42% | 5.37% | 14.29% | 7.14% | 35.72% |
| Sina (182) | 13.32% | 13.32% | 59.97% | 38.66% | 10.68% | 32.01% | 74.67% |
| Renren (68) | 18.17% | 9.09% | 30.31% | 18.19% | 12.12% | 24.23% | 65.13% |
| Tencent Weibo[3] (36) | 12.50% | 12.50% | 12.50% | 37.50% | 12.50% | 0 | 50.00% |
| Average (405) | 13.09% | 11.85% | 37.53% | 18.52% | 11.36% | 20.00% | 55.31% |

- [1] We divide the number of App with specific problem by the number of tested applications which use the STATE parameter.
- [2] There may be overlaps. For example, applications lacking of validation may also be susceptible to lenient STATE validation.
- [3] Only 8 applications use the STATE parameter, thus the distribution is skewed.

# 发现的漏洞2：通过双重角色IdP进行放大攻击

- 双重角色IdP是指，该IdP在为它自身应用提供基于OAuth认证的同时，还使用其他IdP提供的OAuth服务。本文发现，如果双重角色的IdP无法为其领导的IdP正确地实现OAuth，则双重角色IdP平台上的所有应用程序，无论其实现措施是否安全，都会受到影响。

- 因此，攻击者可以通过在双重角色的IdP平台中利用上述漏洞进行攻击，而不是攻击个人第三方应用程序，从而显着放大其攻击的影响。

- 此外，双重角色IdP通常支持多个IdP作为其提供商。

# 发现的漏洞3：未能撤销授权

- 如第4.2.2节所述，我们将系统回滚到先前的状态，然后尝试访问下一个状态。通过这种方式，我们会发现与正常执行顺序与异常行为有所偏差，这可能导致违规授权，因为攻击者可以绕过授权步骤，换句话说，只要该应用程序已被授权一次（但现在已被撤销），IdP将在接收到授权请求（图3中的R4）时自动发出未授权应用程序的代码（状态S5）。

# 关于未采用TLS保护的新发现

- OAuth的安全性主要取决于TLS的使用。而没有部署TLS方案的情况如下，没有TLS意味着攻击者可以窃取密码或访问令牌：

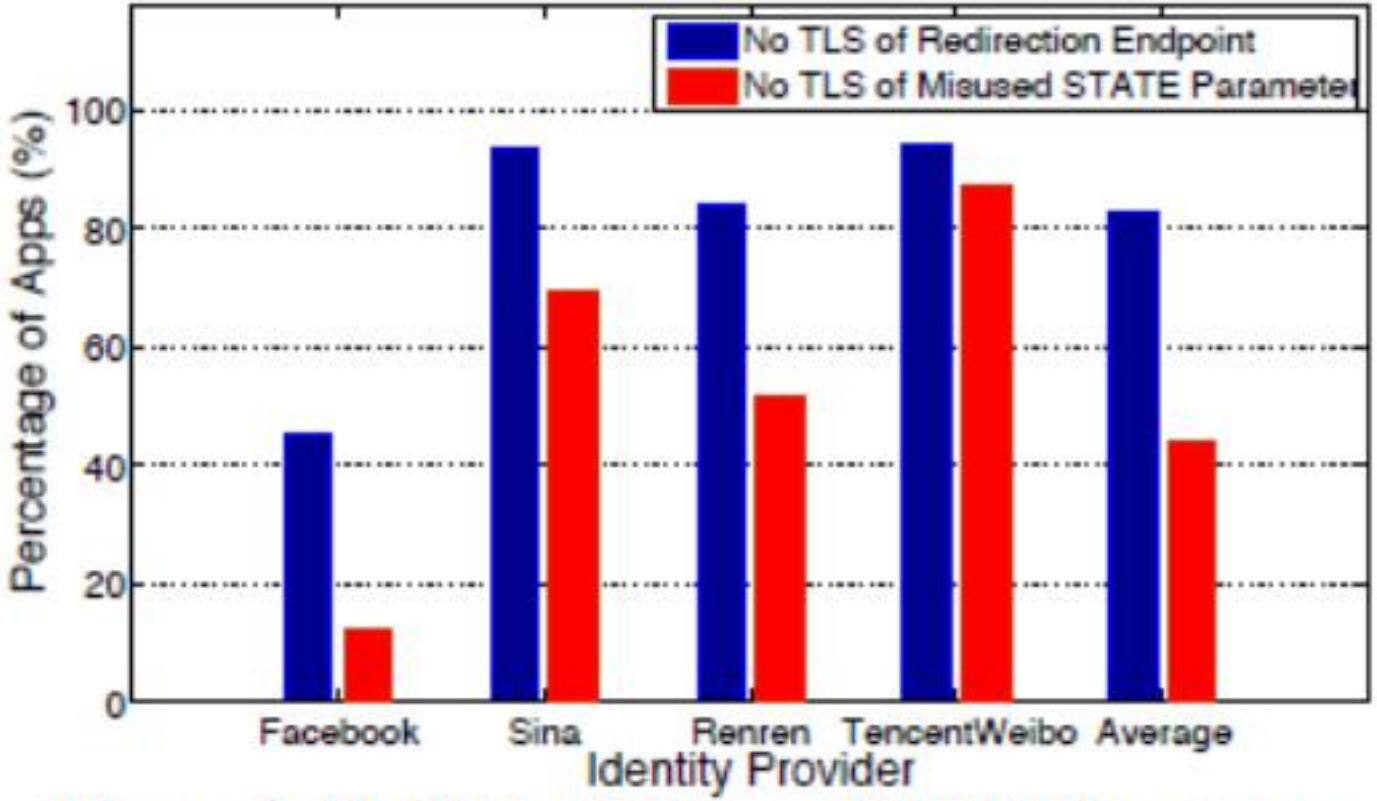


Figure 6: No TLS of Misused STATE Parameter

# 再次发现已存在的漏洞

- 除了发现上述新的漏洞和漏洞之外，OAuthTester还成功重新发现了现有的OAuth安全漏洞。这种现有的漏洞包括由SOSOS [4]发现的4个漏洞，[4]发现的4个漏洞，[37]中提出的4个漏洞，App Impersonation攻击[21]和隐蔽重定向[24]发现的漏洞。

# 讨论

- 本文可以进一步完善OAuthTester，其功能取决于系统模型的精细度。本文讨论了一些潜在的安全漏洞，这些缺陷目前还没有得到很好的研究：
- 子系统不一致：应用程序可能包含一个主系统和多个子系统。而一旦用户登录到子系统中，用户将自动登录主系统。因此，如果子系统错误地实施OAuth，攻击者可能会瞄准子系统来破坏主系统。
- 重新验证应用程序失败：根据RFC6749的要求，IdP应在刷新访问令牌期间通过app_secret验证应用程序，但某些IdP（例如，腾讯微博）根本就不认证他们的应用。因此，一旦他以某种方式获得刷新令牌，攻击者就可以获得访问令牌。
- 预期权限：当用户授权应用程序时，IdP将显示应用程序请求的权限列表，以便用户可以重置并收回特权和不必要的权限。但是，用户从来不知道IdP是否真的发出/撤销了预期的权限。

# 结论

- 本文提出了一种基于自适应模型的测试工具OAuthTester来系统地评估OAuth的实现。
- 本文使用OAuthTester对500个顶级的美国和中国网站进行了测试。
- 本文利用OAuthTester发现了三个新的漏洞。

谢谢各位老师！