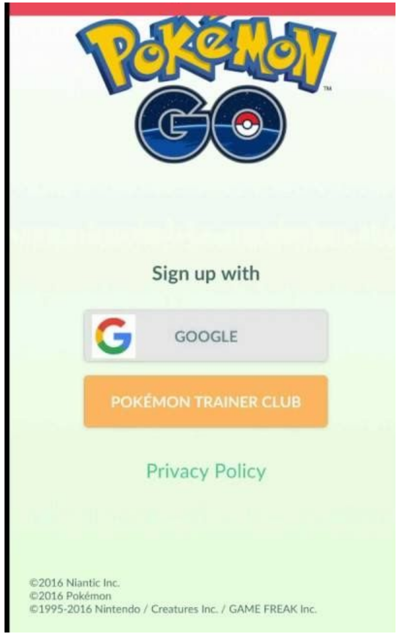
1000 Ways to Die in Mobile OAuth

Eric Chen, **Yutong Pei, Yuan Tian,**
Shuo Chen,Robert Kotcher and Patrick Tague

What is this work about?

- In 2014, Studied OAuth usage in 200 Android/iOS OAuth applications.
  - 60% were implemented incorrectly.
  - **Pinpointed the security-critical portions** in OAuth specs that were **not effectively communicated to mobile app developers**.
- In 2016, these problems are not totally fixed, and new problems are emerging…
- How to do OAuth securely, especially for mobile?

What is OAuth?


Sign up with

GOOGLE

POKÉMON TRAINER CLUB

Privacy Policy

©2016 Niantic Inc.
©2016 Pokémon
©1995-2016 Nintendo / Creatures Inc. / GAME FREAK Inc.

Three parties in OAuth

Resource Owner

Service Provider

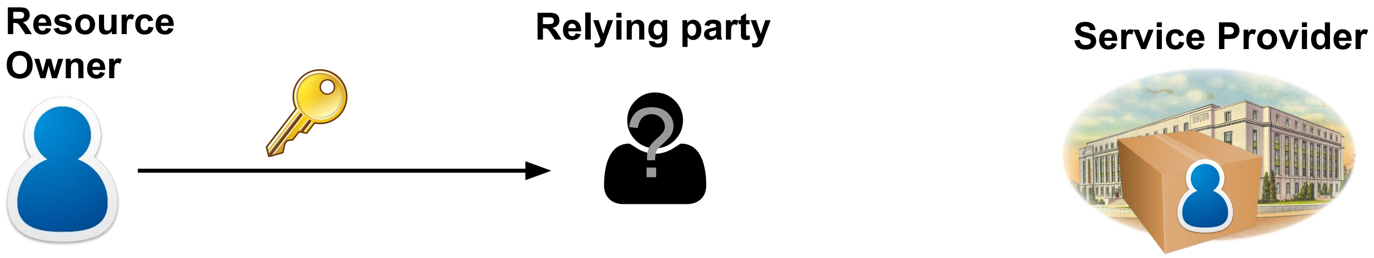
Relying party

Quora

Authorization VS Authentication

Authorization

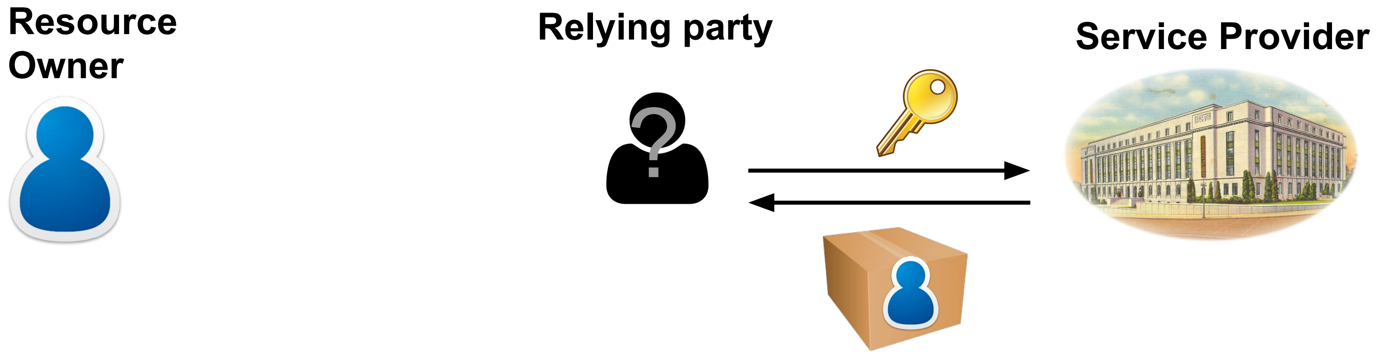
A process for **end-users to grant a third-party website access** to their private resources stored on a service provider.



Authorization VS Authentication

Authorization

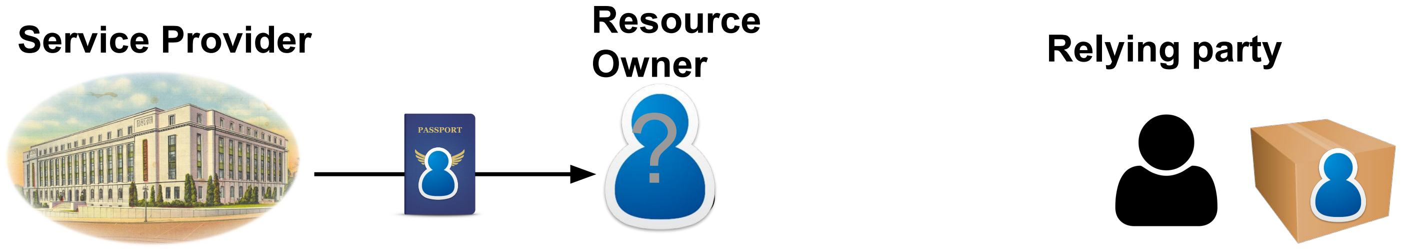
A process for **end-users to grant a third-party website access** to their private resources stored on a service provider.



Authorization VS Authentication

Authentication

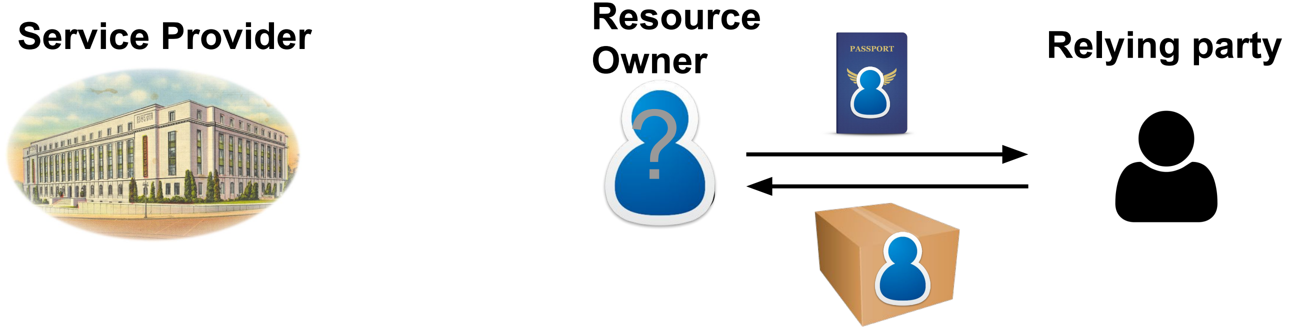
A process for **a user to prove his or her identity to a relying party**, utilizing his or her existing session with the service provider.



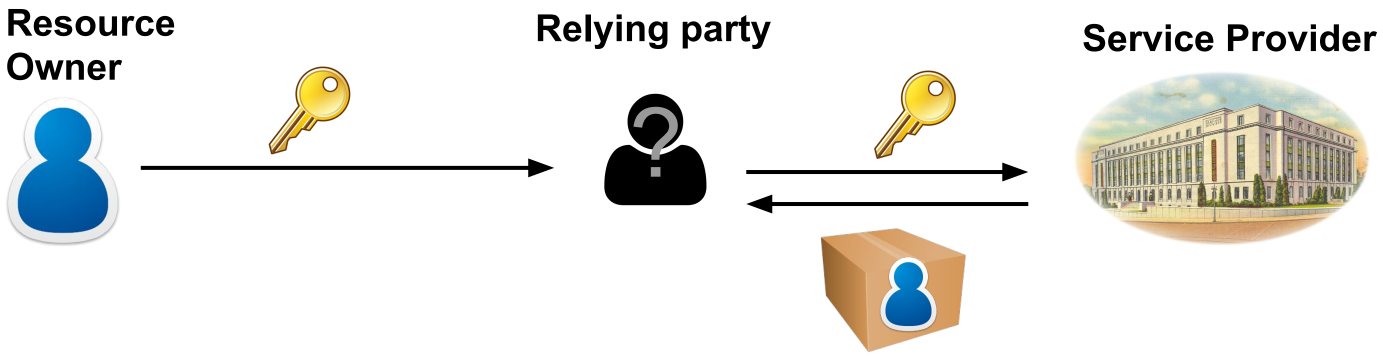
Authorization VS Authentication

Authentication

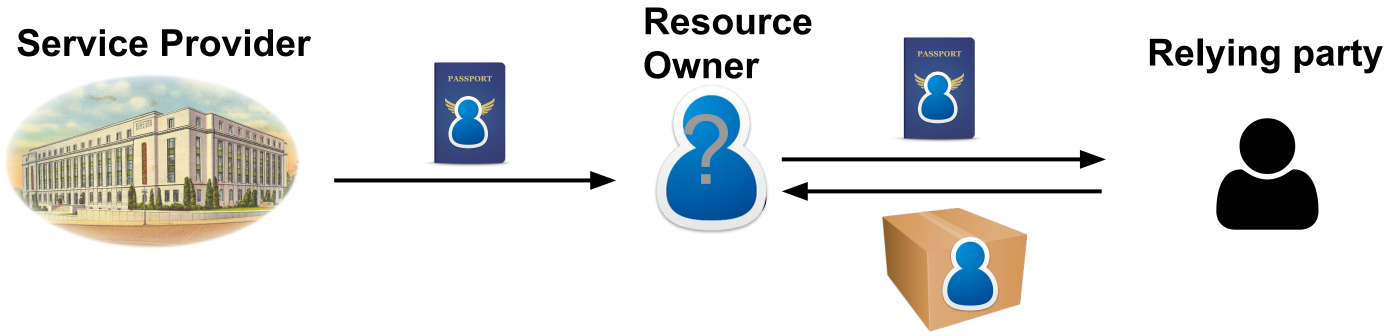
A process for **a user to prove his or her identity to a relying party**, utilizing his or her existing session with the service provider.



Authorization



Authentication



Brief history of OAuth

- (2007) OAuth 1.0
- (2010) 1.0 Standardized through ietf
- (2012) OAuth 2.0 (has 4 “grant types”)
    - Authorization code grant
    - Implicit grant
    - Resource owner password credentials
    - Client credentials

Used by real world mobile apps

- (2007) OAuth 1.0
- (2010) 1.0 Standardized through ietf
- (2012) OAuth 2.0 (has 4 “grant types”)
  - Authorization code grant
  - Implicit grant
  - Resource owner password credentials
  - Client credentials

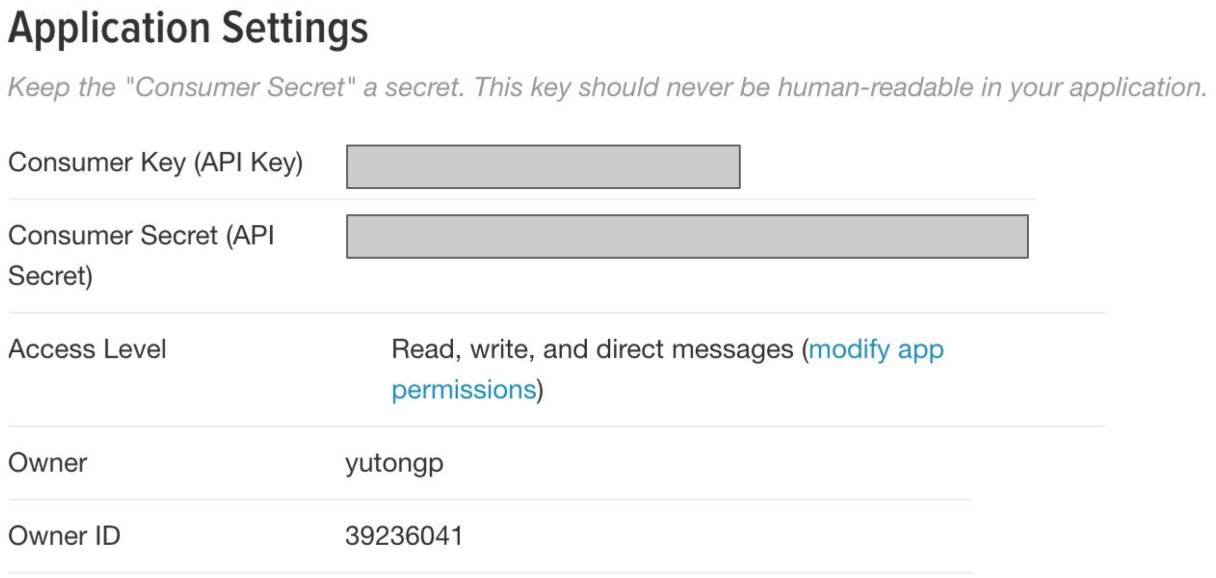
For the rest of this talk

- Study the OAuth specs in terms of their security.
  - Protocol versions: **OAuth 1.0**, **OAuth 2.0 implicit flow**
  - Use cases: **Authorization**, **Authentication**

- Identify parts of the specification that were miscommunicated to mobile developers.

OAuth 1.0

Register your application on the service provider

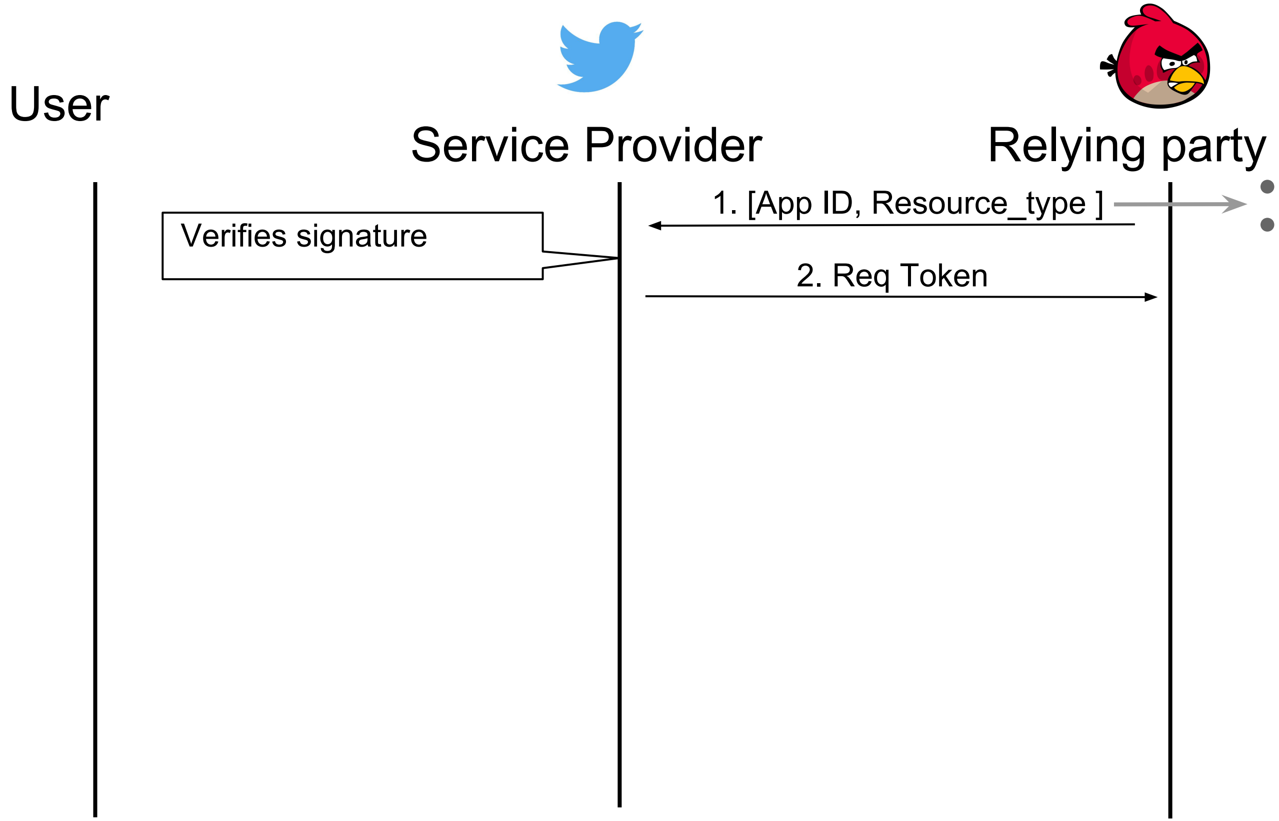


Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

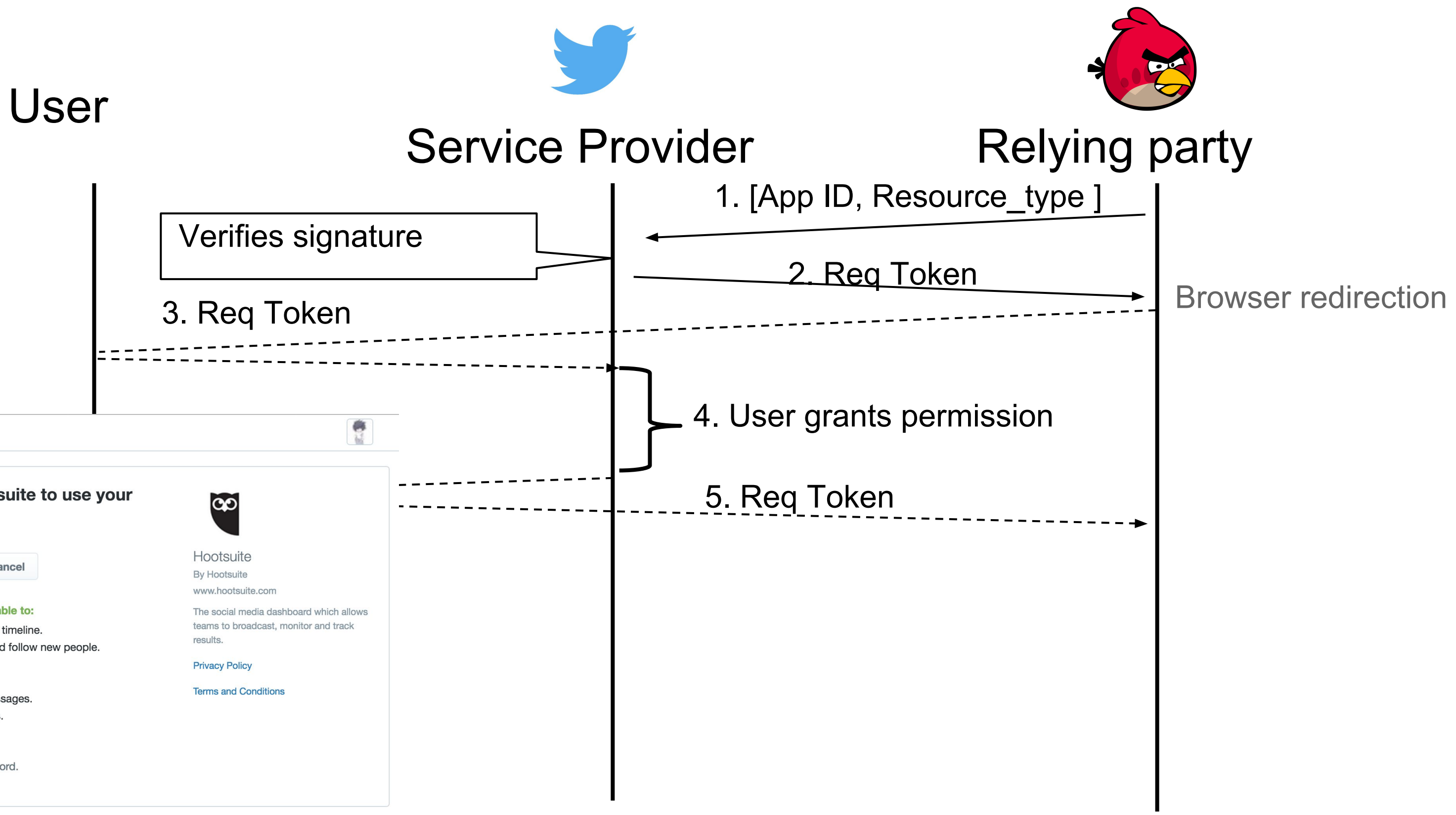
Consumer Key (API Key)	[redacted]
Consumer Secret (API Secret)	[redacted]
Access Level	Read, write, and direct messages (modify app permissions)
Owner	yutongp
Owner ID	39236041

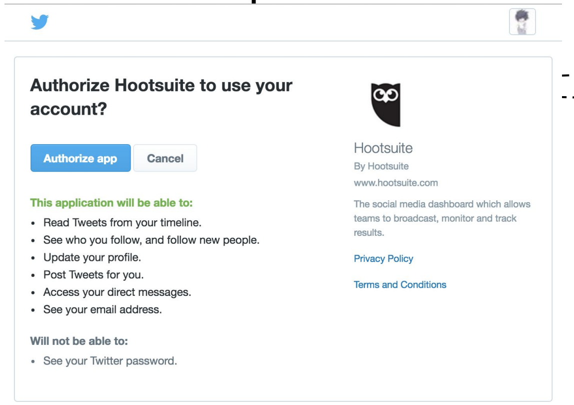
OAuth 1.0



- Signed with app secret
- Resource_type can be: email, user's photos, etc

OAuth 1.0





Authorize Hootsuite to use your account?

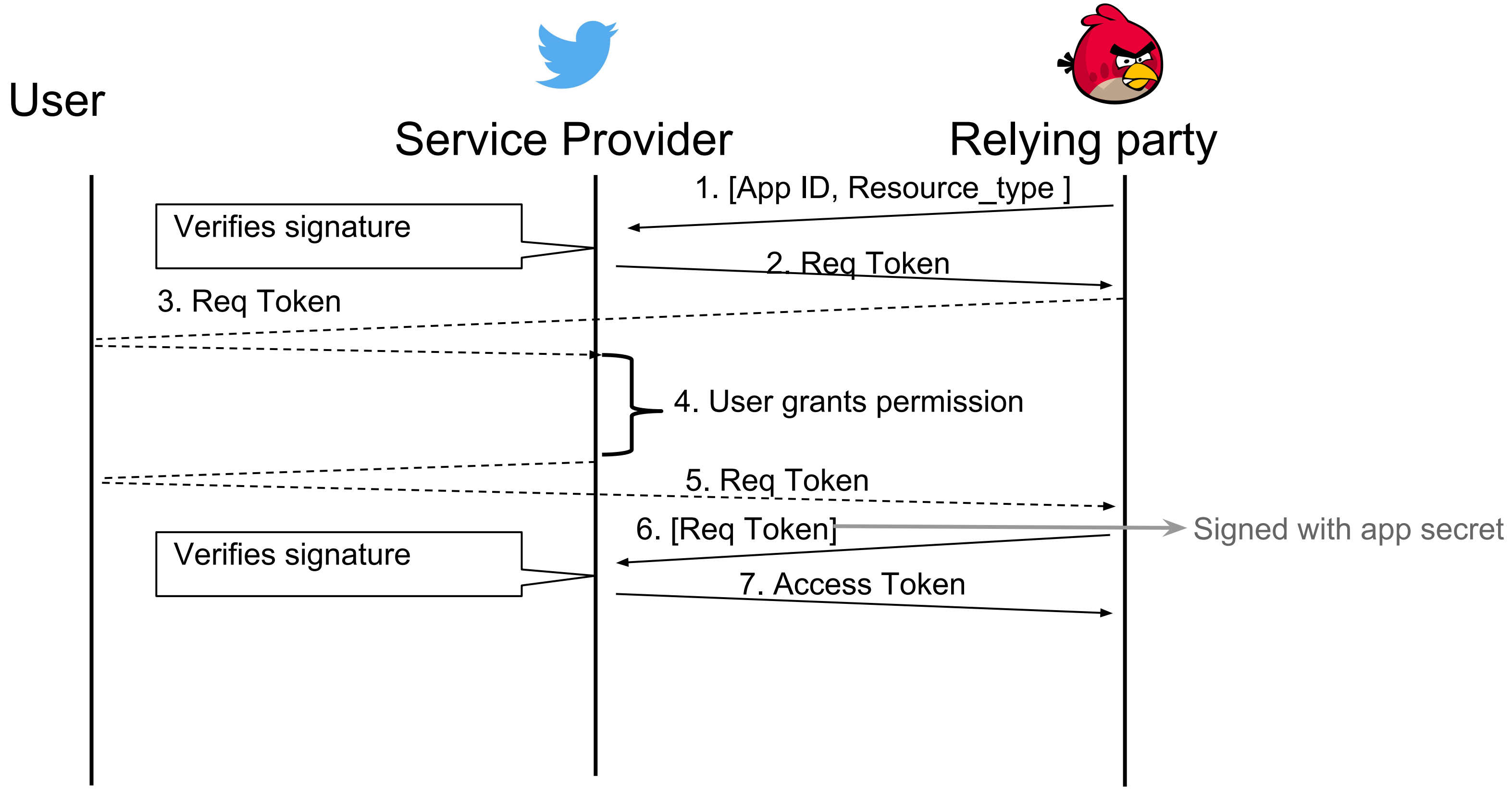
Authorize app | Cancel

Hootsuite
By Hootsuite
www.hootsuite.com
The social media dashboard which allows teams to broadcast, monitor and track results.
Privacy Policy
Terms and Conditions

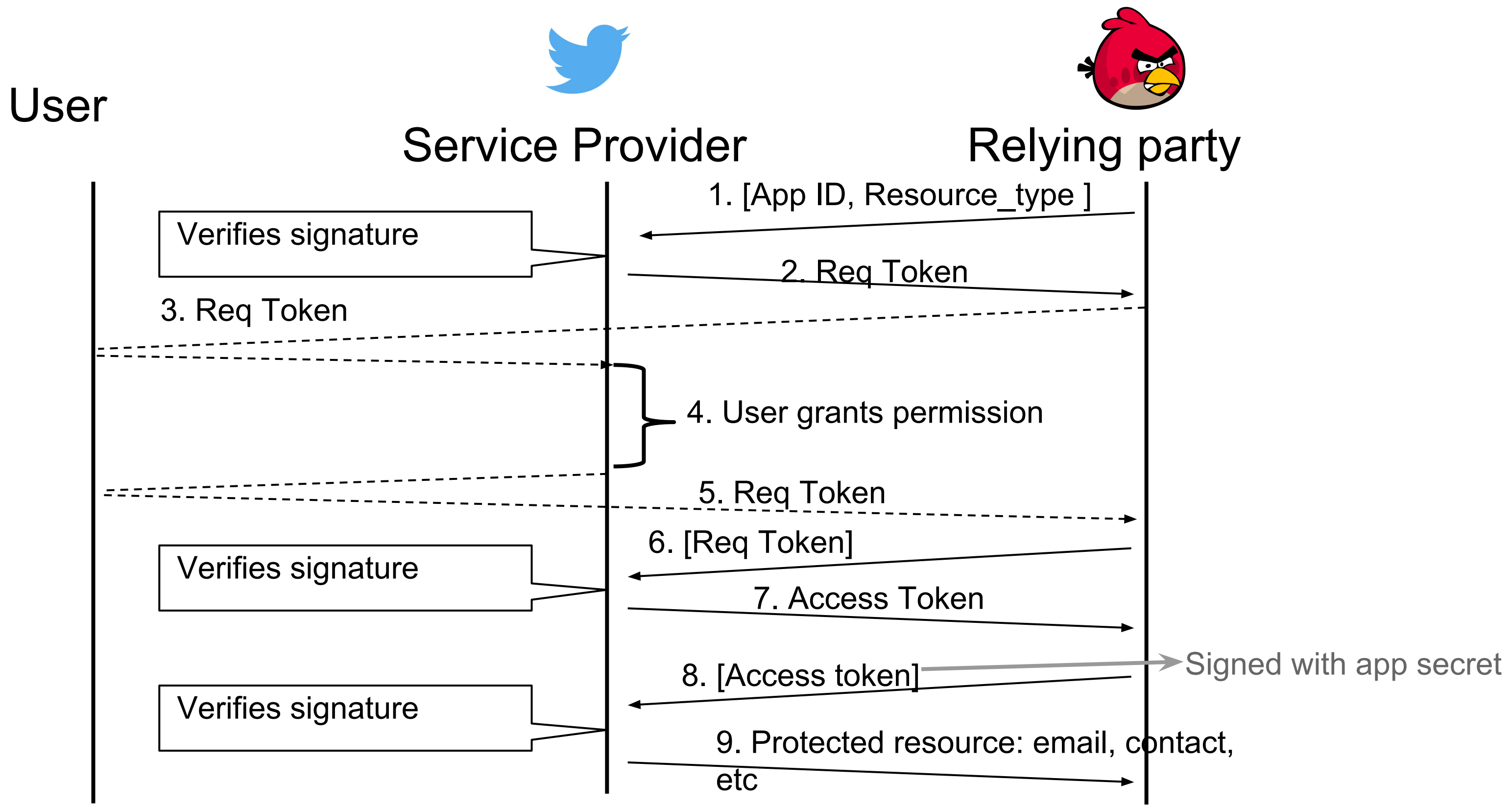
This application will be able to:
- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages.
- See your email address.

Will not be able to:
- See your Twitter password.

OAuth 1.0

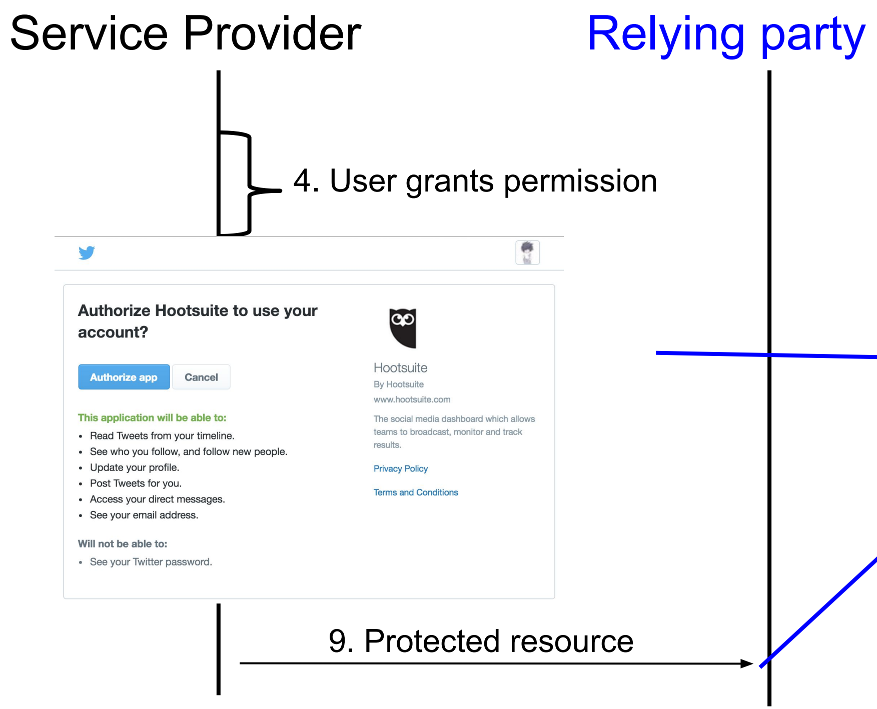

User
Service Provider
Relying party
1. [App ID, Resource_type]
Verifies signature
2. Req Token
3. Req Token
4. User grants permission
5. Req Token
6. [Req Token]
Signed with app secret
Verifies signature
7. Access Token

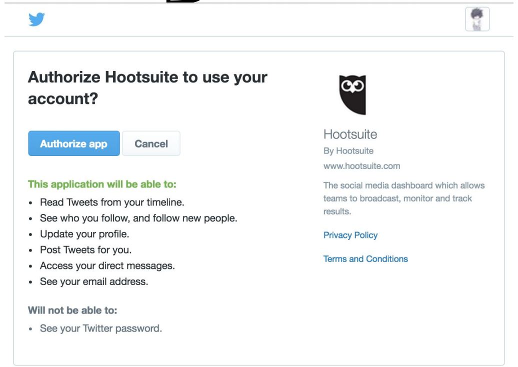
OAuth 1.0



User
Service Provider
Relying party
1. [App ID, Resource_type]
Verifies signature
2. Req Token
3. Req Token
4. User grants permission
5. Req Token
6. [Req Token]
Verifies signature
7. Access Token
8. [Access token]
Signed with app secret
Verifies signature
9. Protected resource: email, contact, etc

OAuth 1.0 Security - Relying Party Identity





Authorize Hootsuite to use your account?

Authorize app | Cancel

This application will be able to:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages.
- See your email address.

Will not be able to:

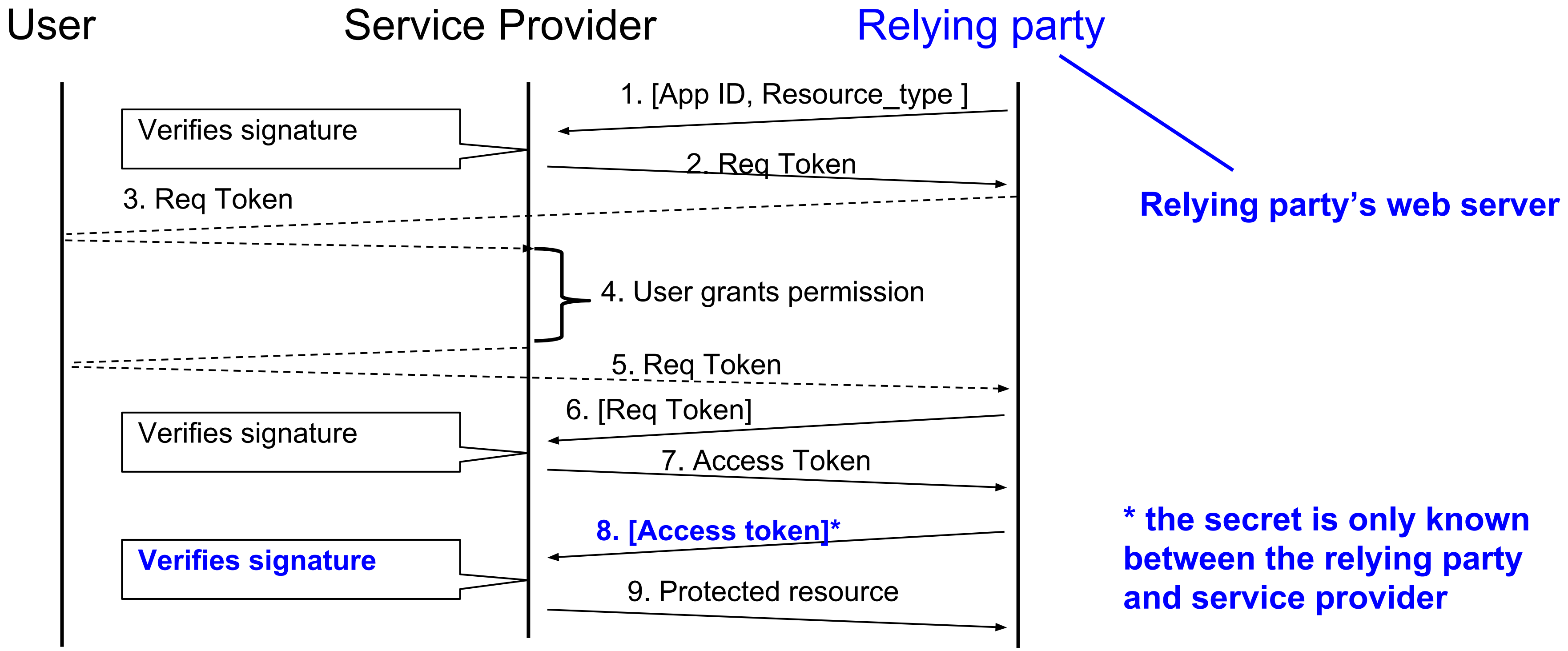
- See your Twitter password.

Hootsuite
By Hootsuite
www.hootsuite.com
The social media dashboard which allows teams to broadcast, monitor and track results.
Privacy Policy
Terms and Conditions

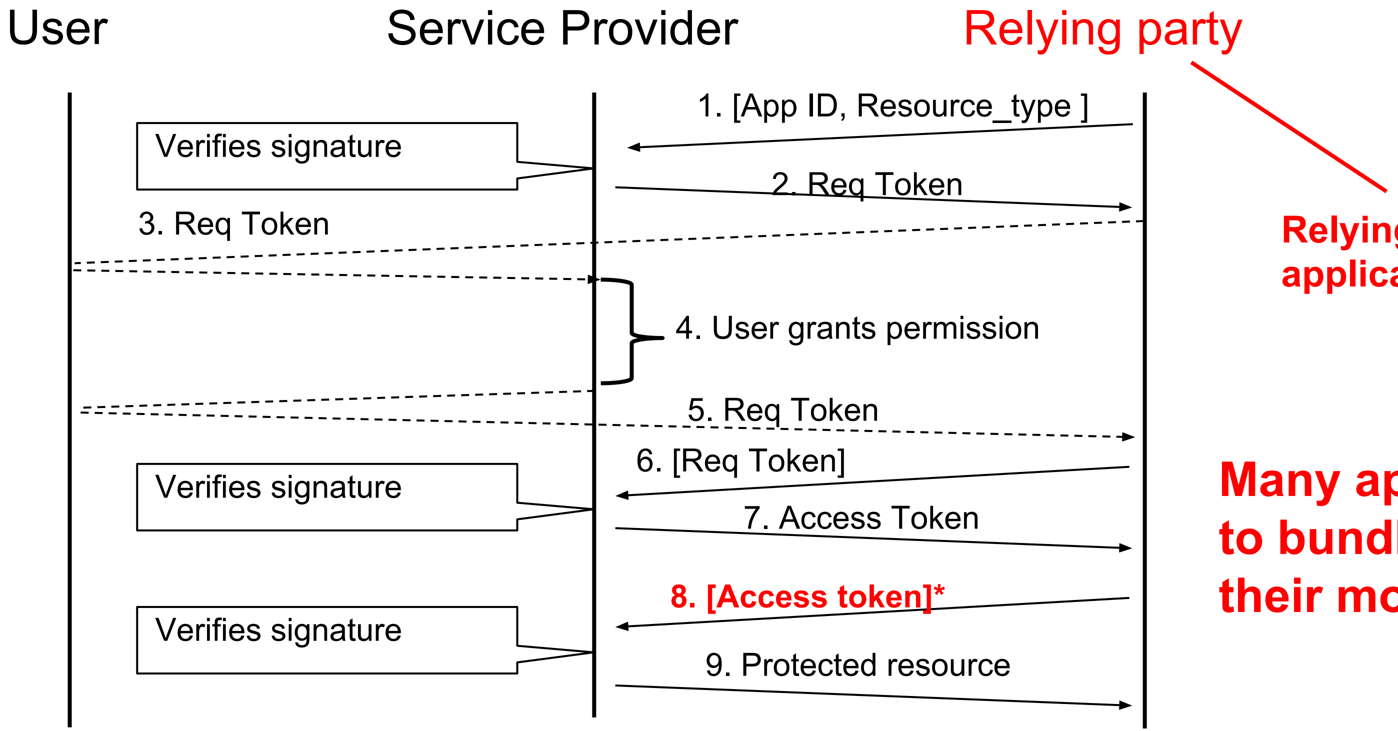
Untrusted

Achieving security:
The receiver of the protected resource must be the same principle that the user granted access to.

OAuth 1.0 Security - Relying Party Identity



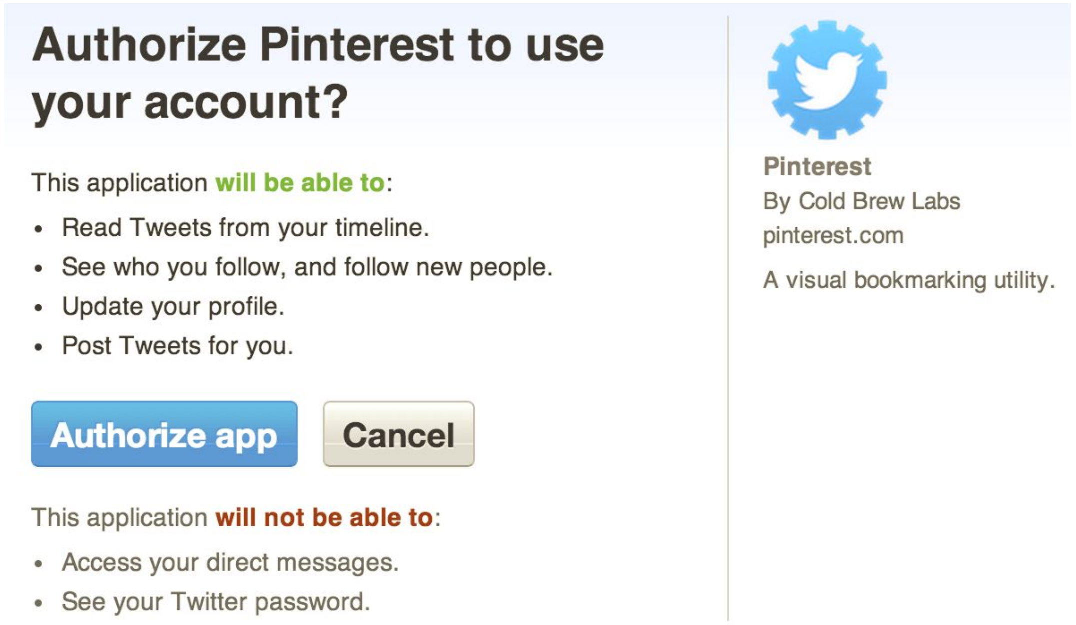
Vulnerability - Locally stored secrets



Relying party's mobile application

Many applications decide to bundle this secret into their mobile apps.

Vulnerability - Locally stored secrets



Authorize Pinterest to use your account?

This application **will be able to**:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

This application **will not be able to**:

- Access your direct messages.
- See your Twitter password.

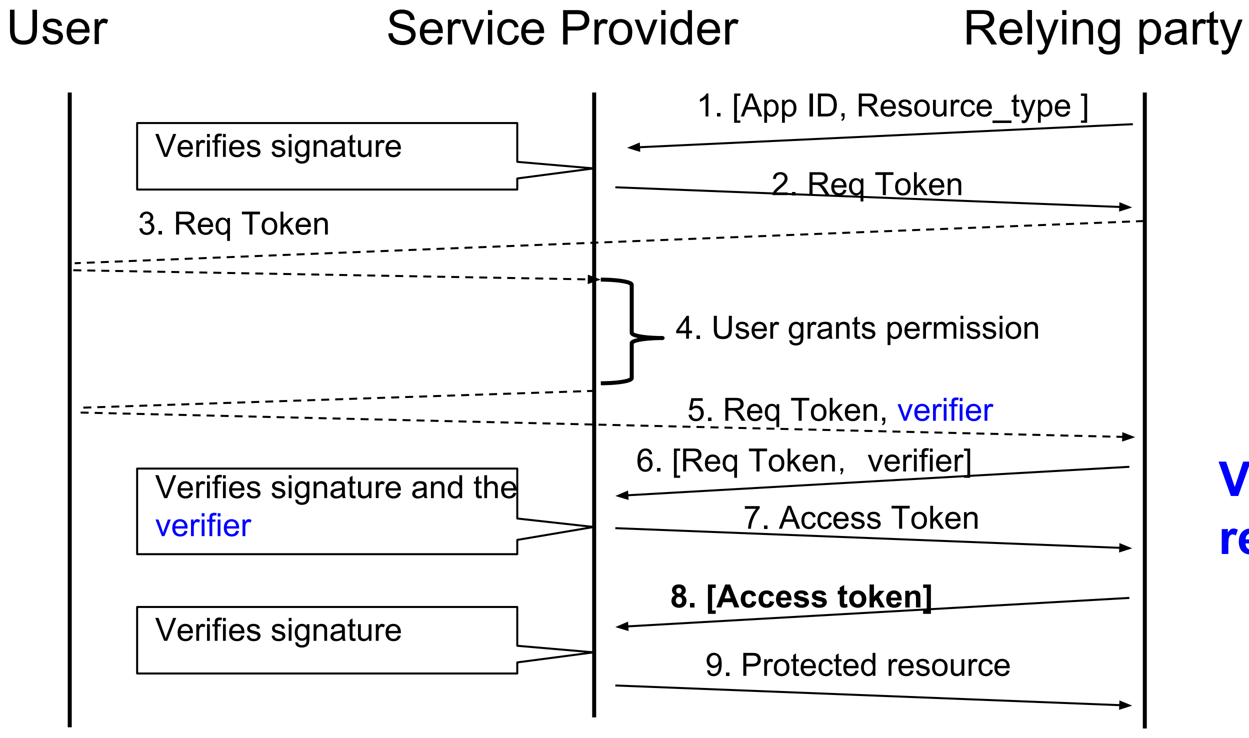
Pinterest
By Cold Brew Labs
pinterest.com

A visual bookmarking utility.

Vulnerability - Locally stored secrets

- After we notified Quora and Pinterest in 2014
  - Both Quora and Pinterest revoked their existing relying party secrets.
  - Quora’s twitter authentication was non-functional after our report.
- Currently both not using twitter login anymore...

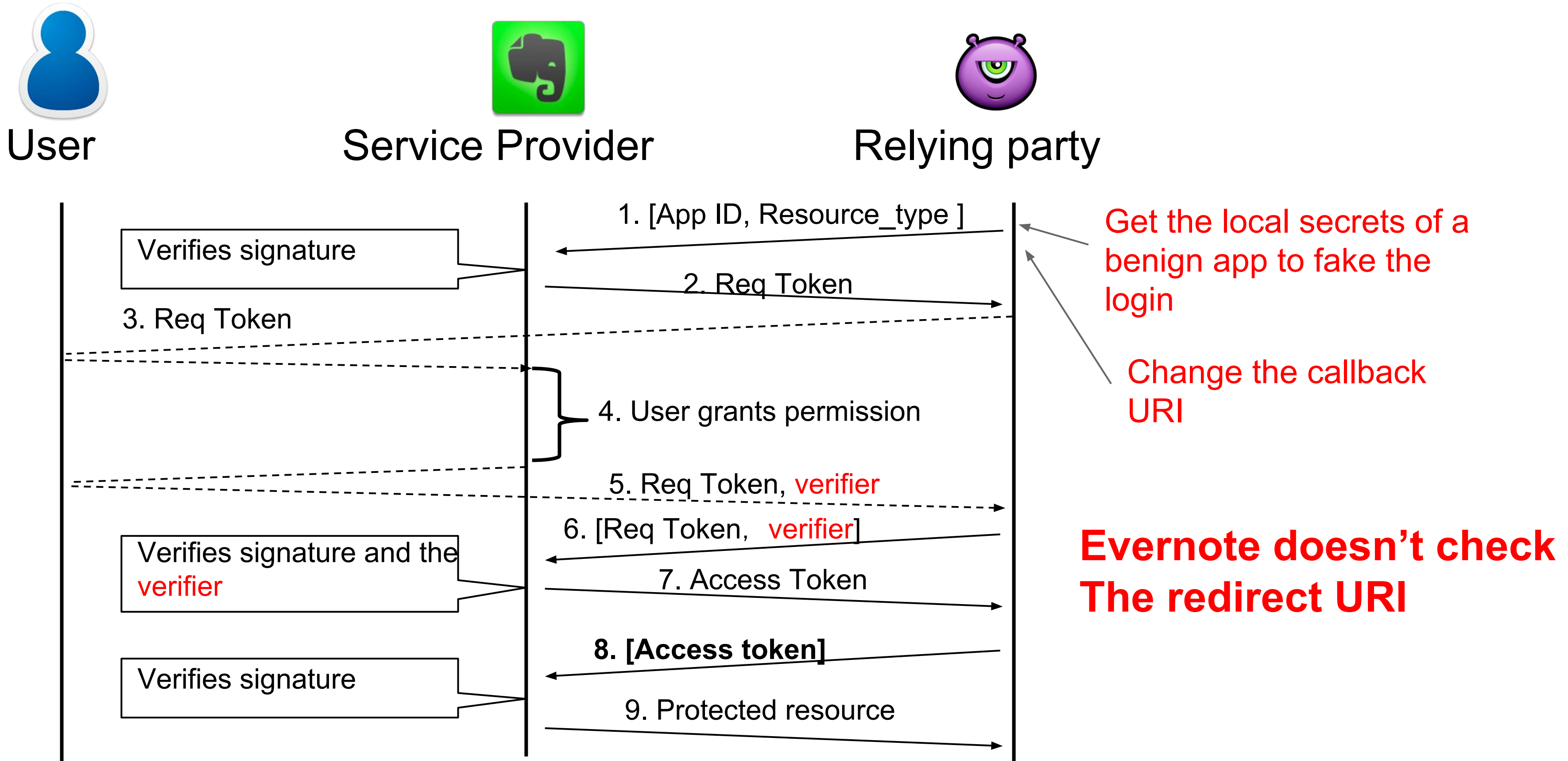
OAuth1.0a improvement



OAuth 1.0a

Verifier is only sent to the registered redirect URL

Locally store secrets + redirect URL



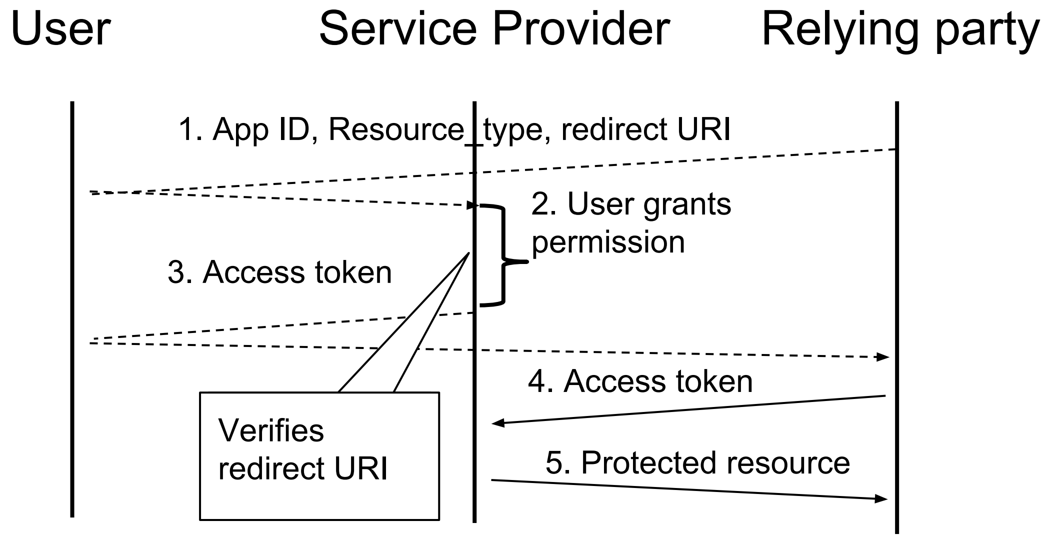
Evernote doesn't check The redirect URI

Security critical design and implementation for OAuth1.0/a

- Relying Party
  - Do not bundle client secret in the client side
- Service Provider
  - Register the redirect URI and check the redirect URI
- Or…

Use OAuth2

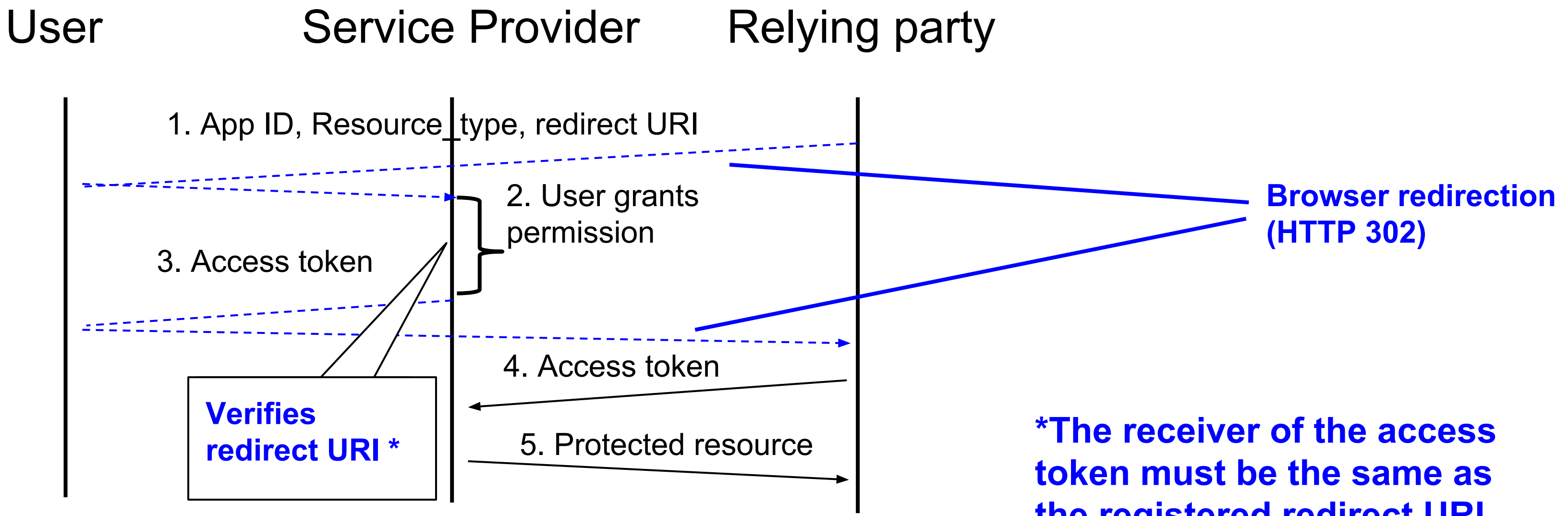
OAuth 2.0 implicit flow



Relying party must supply a "redirect URI" to receive access tokens from the service provider

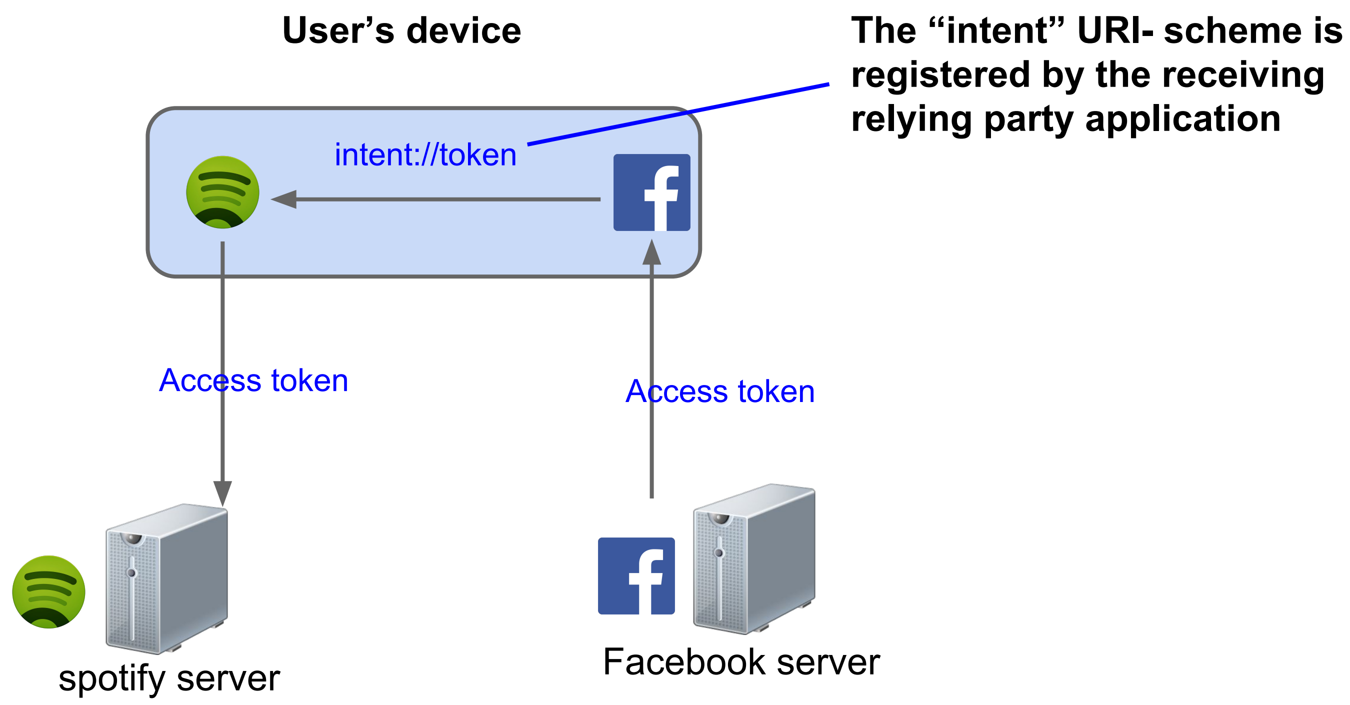
1. No relying party secret!
2. No signature/encryption.
3. Access token is not bound to a RP

Handling redirection in Implicit flow

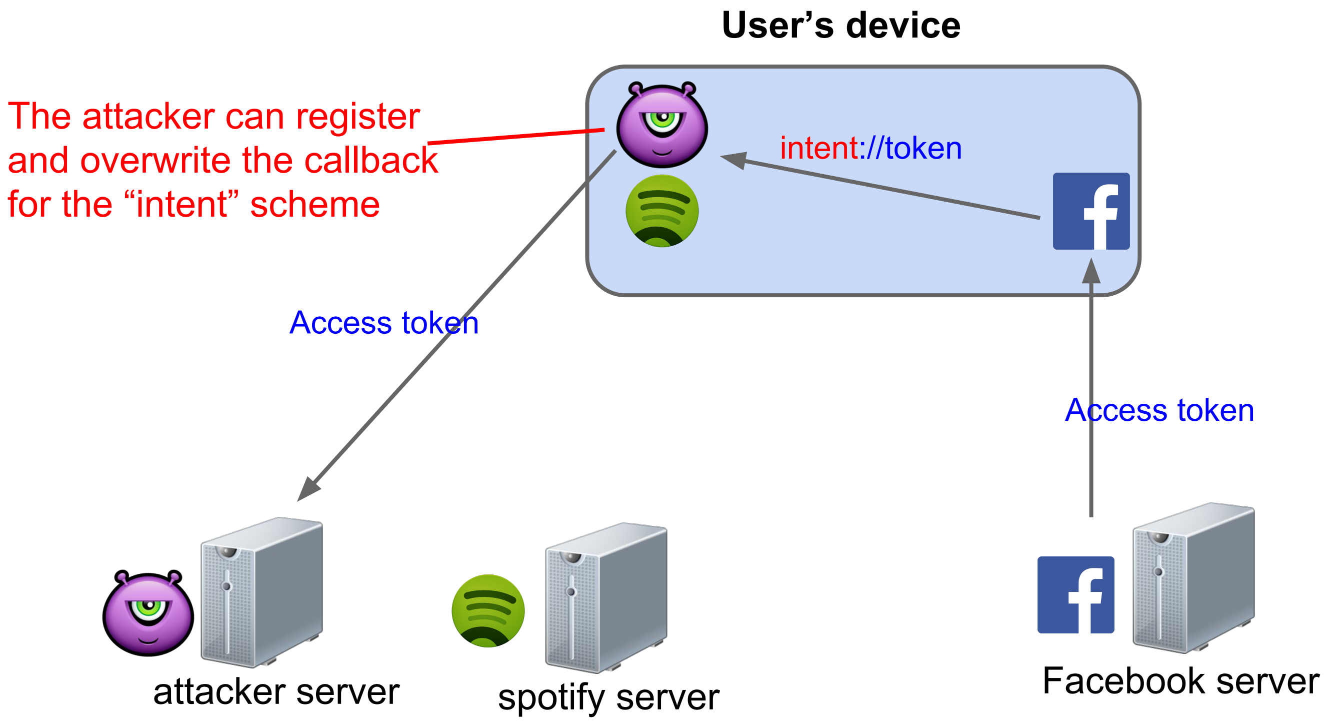


*The receiver of the access token must be the same as the registered redirect URI

Handling redirection in mobile applications


User's device
intent://token
The "intent" URI- scheme is registered by the receiving relying party application
Access token
Access token
spotify server
Facebook server

Handling redirection in mobile applications

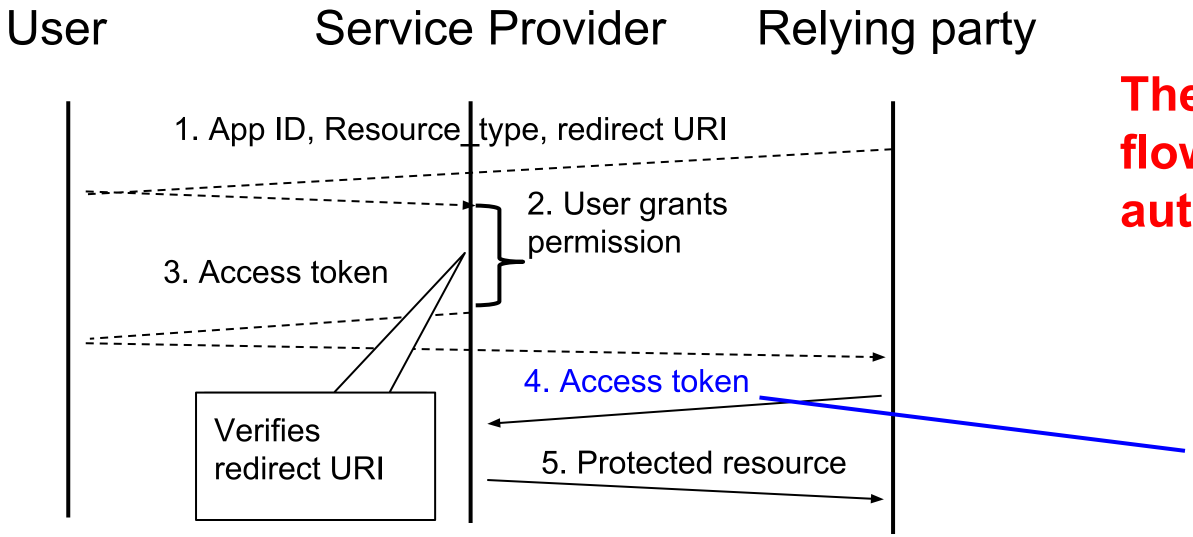


Handling redirection in mobile applications

- Secure redirection using Android Intents:
  - Each application is signed using a developer key.
  - We can check the developer’s key hash of the intent receiver.

```
relying_party =  Activity.getCallingPackage();
dev_key_hash = getPackageManager().
          getPackageInfo(relying_party, PackageManager.GET_SIGNATURES);
```

Using Implicit Flow for Authentication

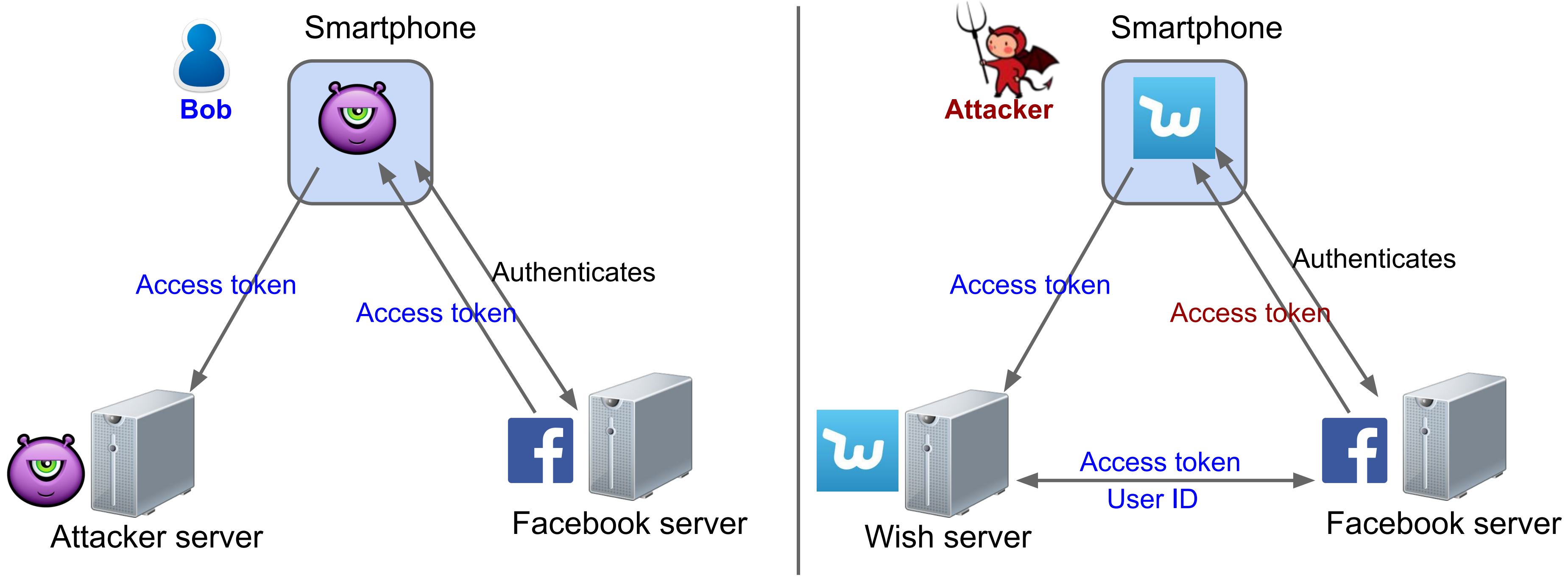


The OAuth 2.0 implicit flow is not secure for authentication

The access token is not bound to a relying party

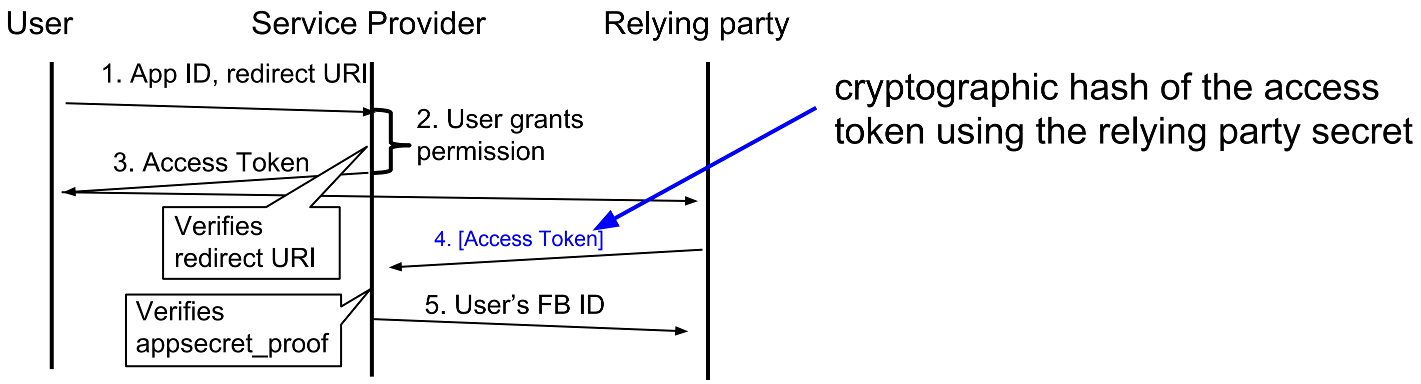
Vulnerability - Using authorization flow for authentication

- Vulnerability in Wish's Android application using FB login:



Vulnerability - Using authorization flow for authentication

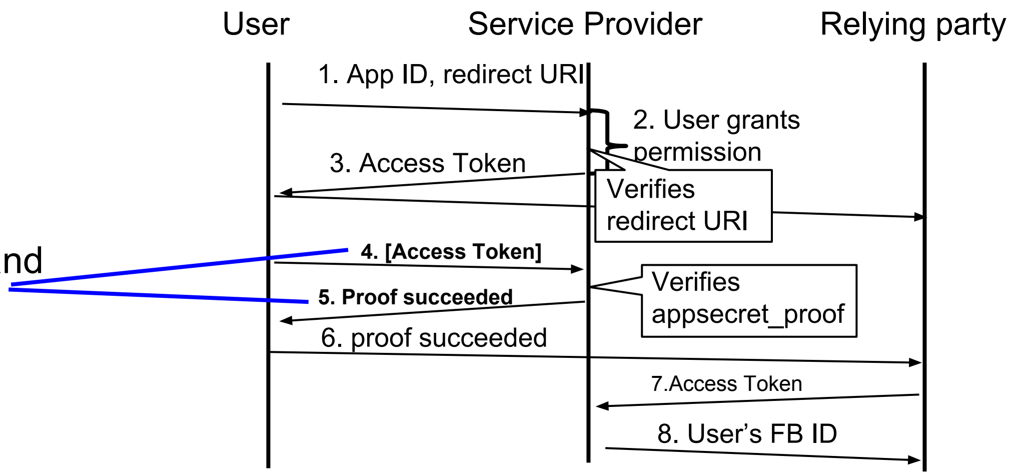
- Facebook also supports a modified implicit flow for authentication.



Vulnerability - Using authorization flow for authentication

- Keek's (vine-like app with 65 million users) "appsecret_proof" flow

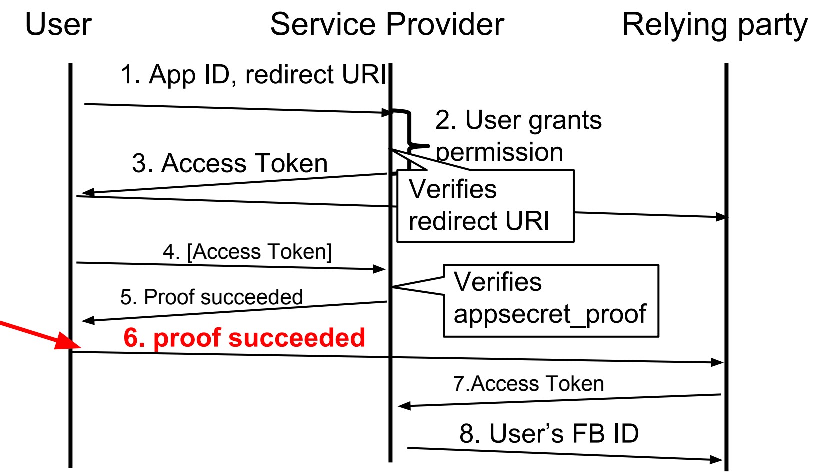
Sent between the user's device and the service provider



Vulnerability - Using authorization flow for authentication

- Keek’s “appsecret_proof” flow

Sent from the client’s device



Vulnerability - Using authorization flow for authentication

Distribution of authentication methods for Facebook relying parties:

- Using unmodified implicit flow: 84.7%
- Using wrongly implemented app_secret proof 1.3%
- Using correctly implemented app_secret proof 14%

- Bounty reward from Instagram (Facebook)

OpenID Connect

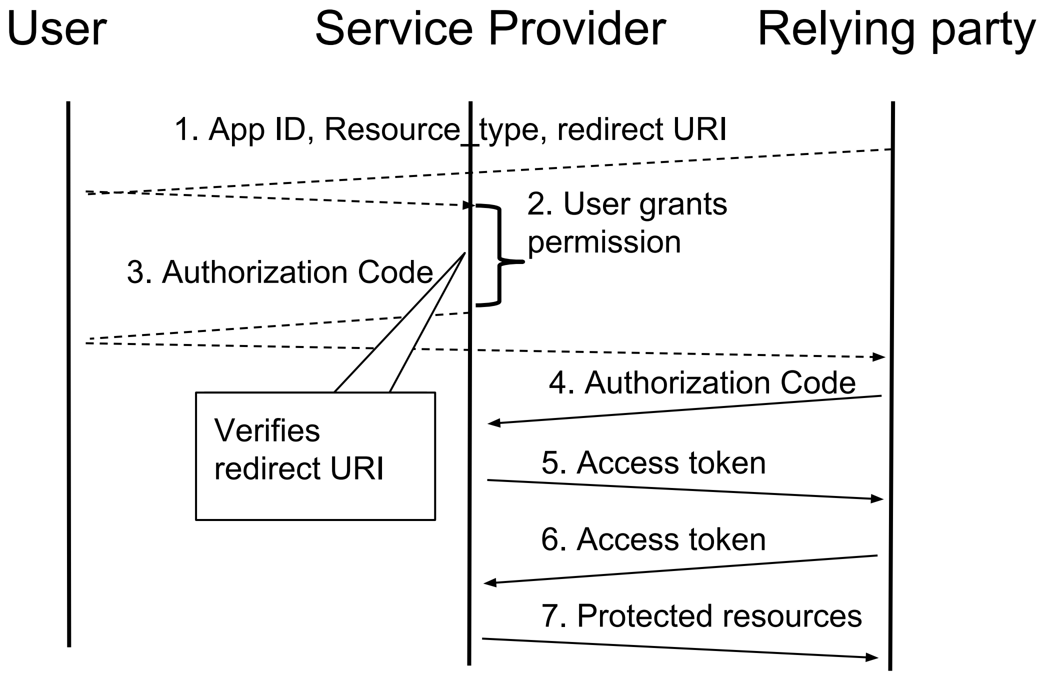
ID token - signed JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3MiOiJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwic3ViIjoiMjQ4Mjg5NzYxMDAxIiwiYXVkIjoiczZCaGRSa
3F0MyIsIm5vbmNlIjoibi0wUzZfV3pBMk1qIiwiZXhwIjoxMzExMjgxOTcwLCJpYXQiOjEzMTEyODA5NzAsImF0X2hhc2
giOiI3N1FtVVB0alBmeld0RjJBbnBLOVJRIn0.VW_s1XIAkhlFTfx90VjofHjbRqM5MEtMA5mlctc7dCE
```

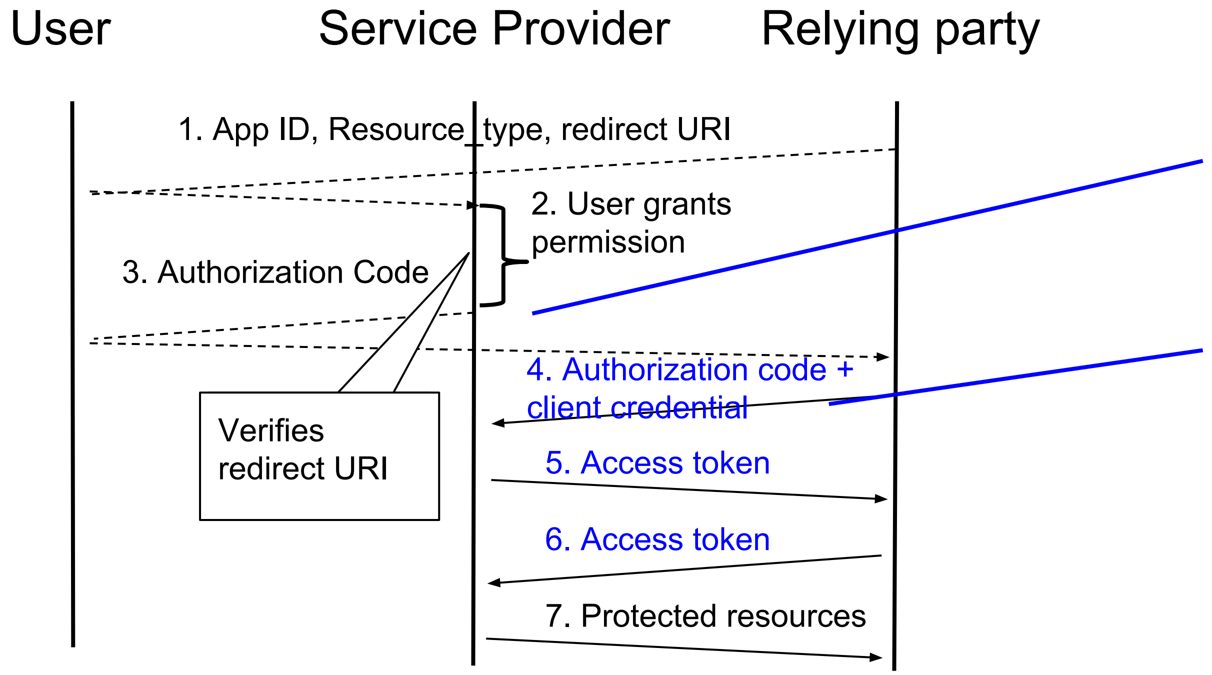
Payload:

```
{
   "iss": "http://server.example.com",
   "sub": "248289761001",
   "aud": "s6BhdRkqt3",
   "nonce": "n-0S6_WzA2Mj",
   "exp": 1311281970,
   "iat": 1311280970,
   "at_hash": "77QmUPtjPfzWtF2AnpK9RQ"
}
```

OAuth 2 Code Authorization Flow



Code Authorization Flow- verify the code

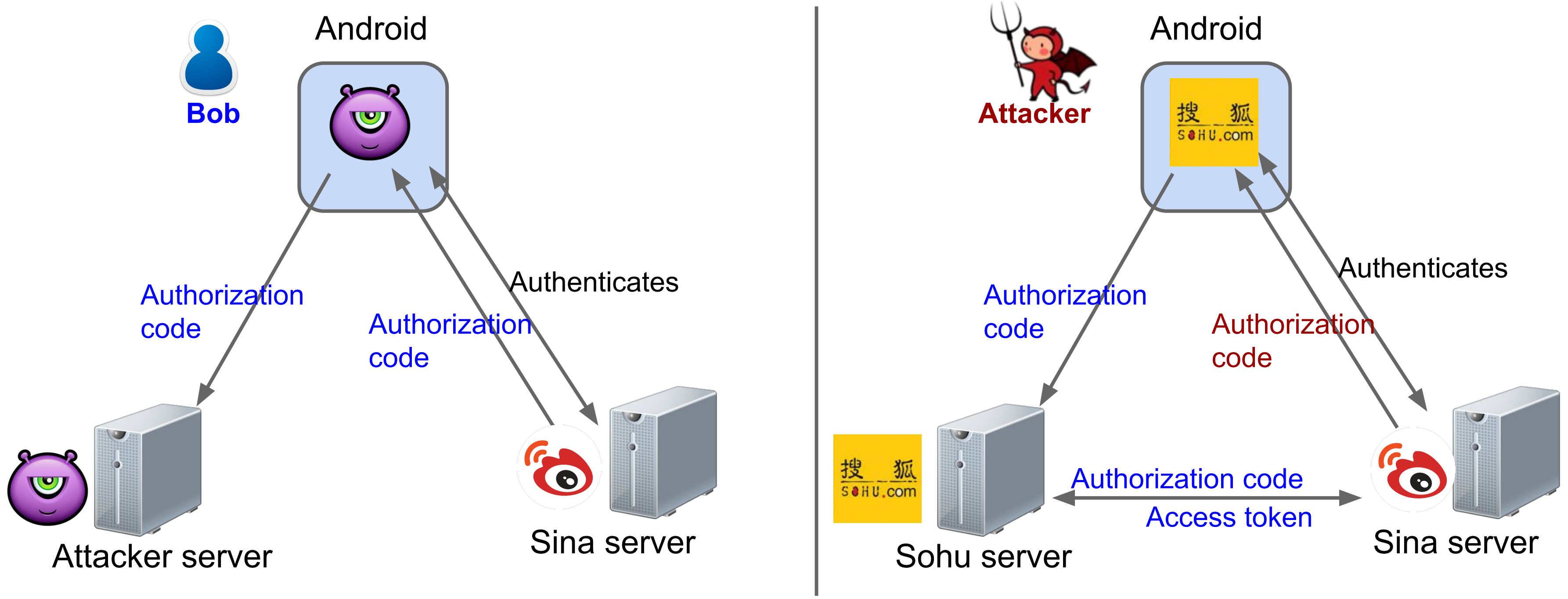


The authorization code should be one-time password

The service provider should verify that the authorization code belongs to the same relying party

Vulnerabilities- not verifying the code

Vulnerability in Sohu news app with Sina login:

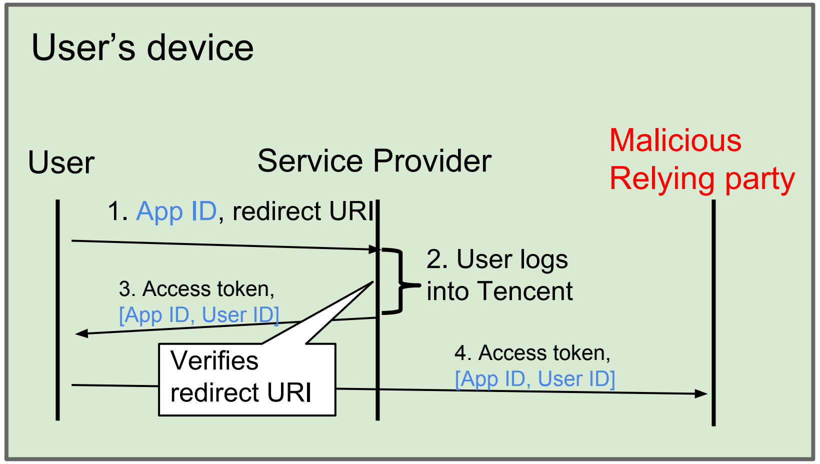


Security critical design and implementation for OAuth2

- Do security checks in the server side
- Verify the receiver and sender of security-critical content such as code and token

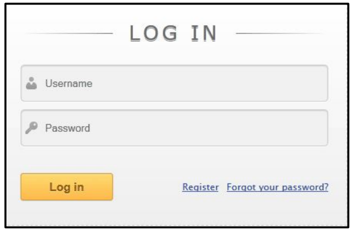
Lake of Consent Information- Tencent

- No information about relying party for Tencent mobile UI



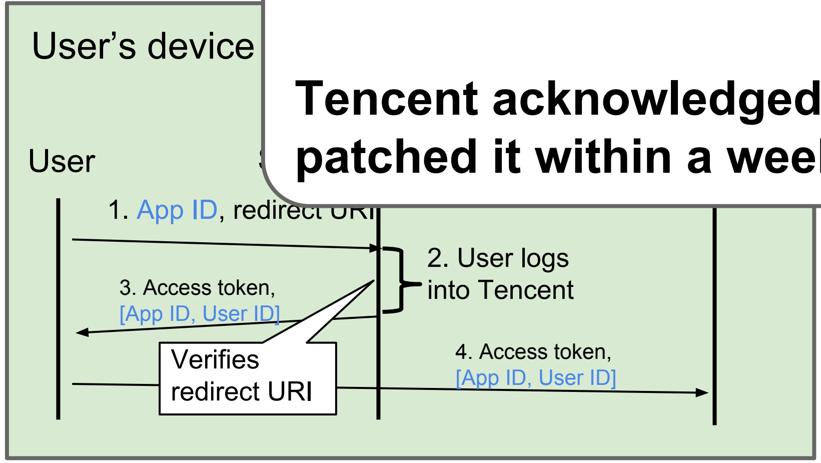
App ID is public information

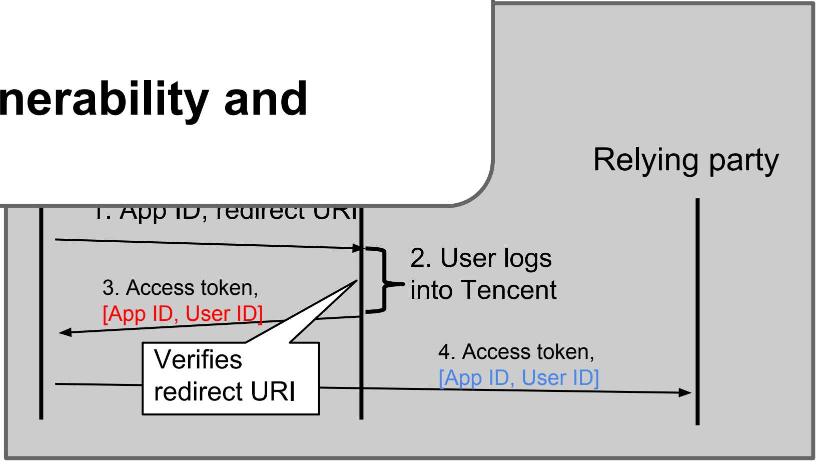
The user sees the same Tencent login-dialog for all relying parties



Impacts

- No information about relying party for Tencent mobile UI


User's device
User
1. App ID, redirect URI
2. User logs into Tencent
3. Access token, [App ID, User ID]
Verifies redirect URI
4. Access token, [App ID, User ID]

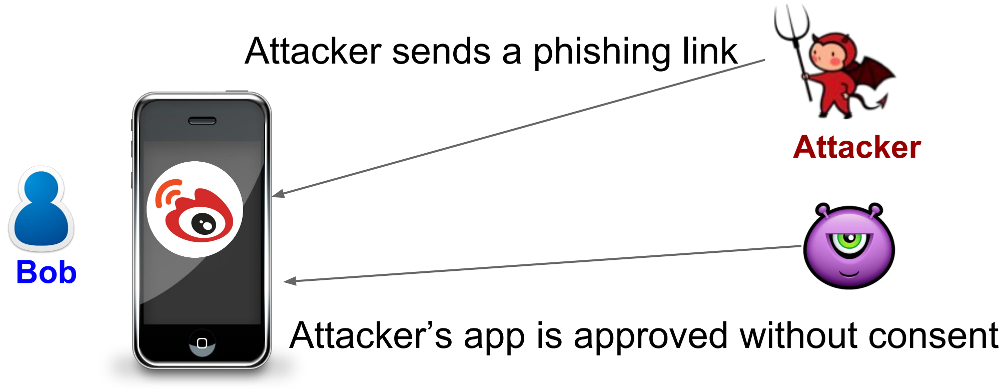

Relying party
1. App ID, redirect URI
2. User logs into Tencent
3. Access token, [App ID, User ID]
Verifies redirect URI
4. Access token, [App ID, User ID]

~700 million users affected.

Tencent acknowledged the vulnerability and patched it within a week.

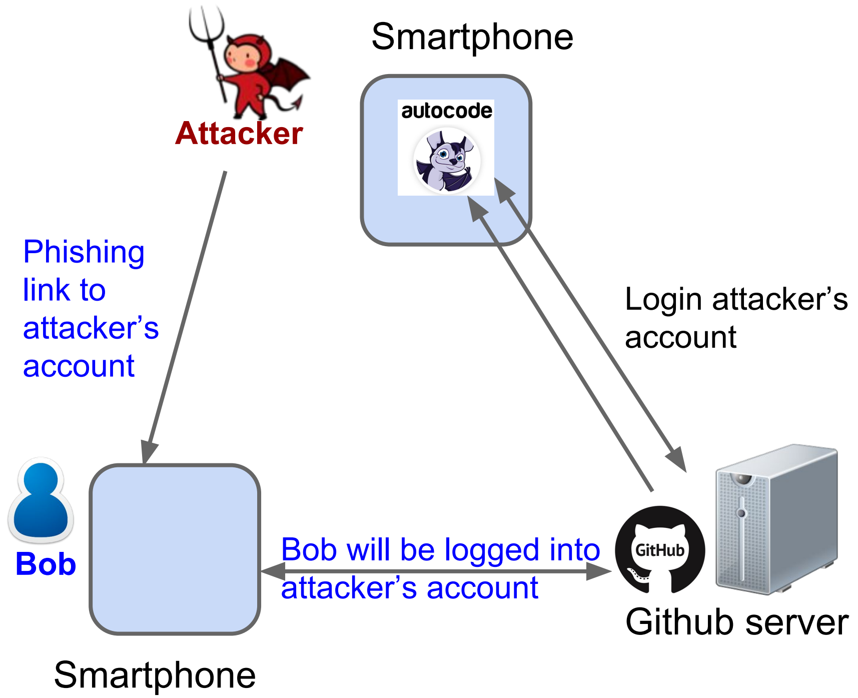
No Consent Page- Sina

Sina doesn't show consent page if the user logs in her Sina account


Attacker sends a phishing link
Attacker
Bob
Attacker's app is approved without consent

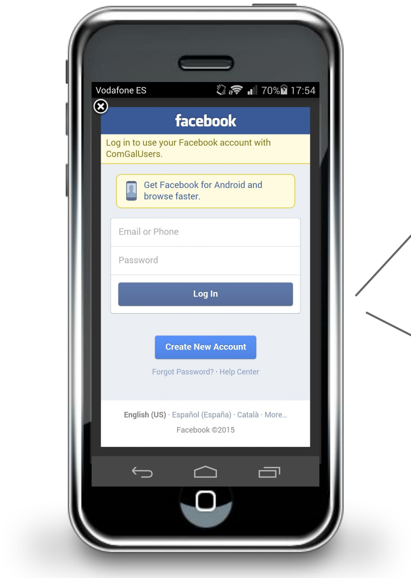
No State Token

Relying party should use state token to identify the login session



Mobile Webview

Service provider should not provide long term cookie in the webview login



Log in to use your Facebook account with ComGalUsers.

Get Facebook for Android and browse faster.

Webview provides the feature that app can get the cookies from the webview it embeds

Facebook uses long term cookie even inside webview, and attacker can reuse the cookie to log in as the user.

How to use mobile OAuth securely?

- Service provider
  - Verify the Identity of the token/code receiver
  - Consent page
  - Set short term cookie for webview
  - Adopt OpenID connect for authentication
- Relying party
  - Do not trust the client
    - Do not store content locally
    - Perform security checks on the server
  - Choose the right flow and follow the flow

Summary

- Studied OAuth usage in 200 Android/iOS OAuth applications.
  - 60% were implemented incorrectly.
- Dissected OAuth specifications for security.
  - Initially designed for **authorization**, not authentication!
  - Initially designed for **web**, not mobile!
- **The OAuth Working group should provide clear usage guidelines for mobile platforms**

Thank you

What is this work about?

Why so many vulnerable applications?

- Specifications were written for **authorization**, not authentication.

- Specifications were written for **web applications**, not mobile applications.

Our study

- Field study of 200 Android/iOS applications
  - 133 were taken from top 600 ranked applications in app stores
  - 16 were manually selected (Quora, Weibo)
  - 16.8% service providers, 84.5% relying parties, 1.3% both
- 59.7% of these applications were vulnerable to attacks

Differences between web and mobile platforms

1. Different redirection mechanisms
   - HTTP 302 Vs. iOS custom schemes or intents
2. Lack of application identity
   - No concept of “Origin” for mobile applications
3. Client-side heavy protocol logic
   - Observation: mobile apps have heavier clients

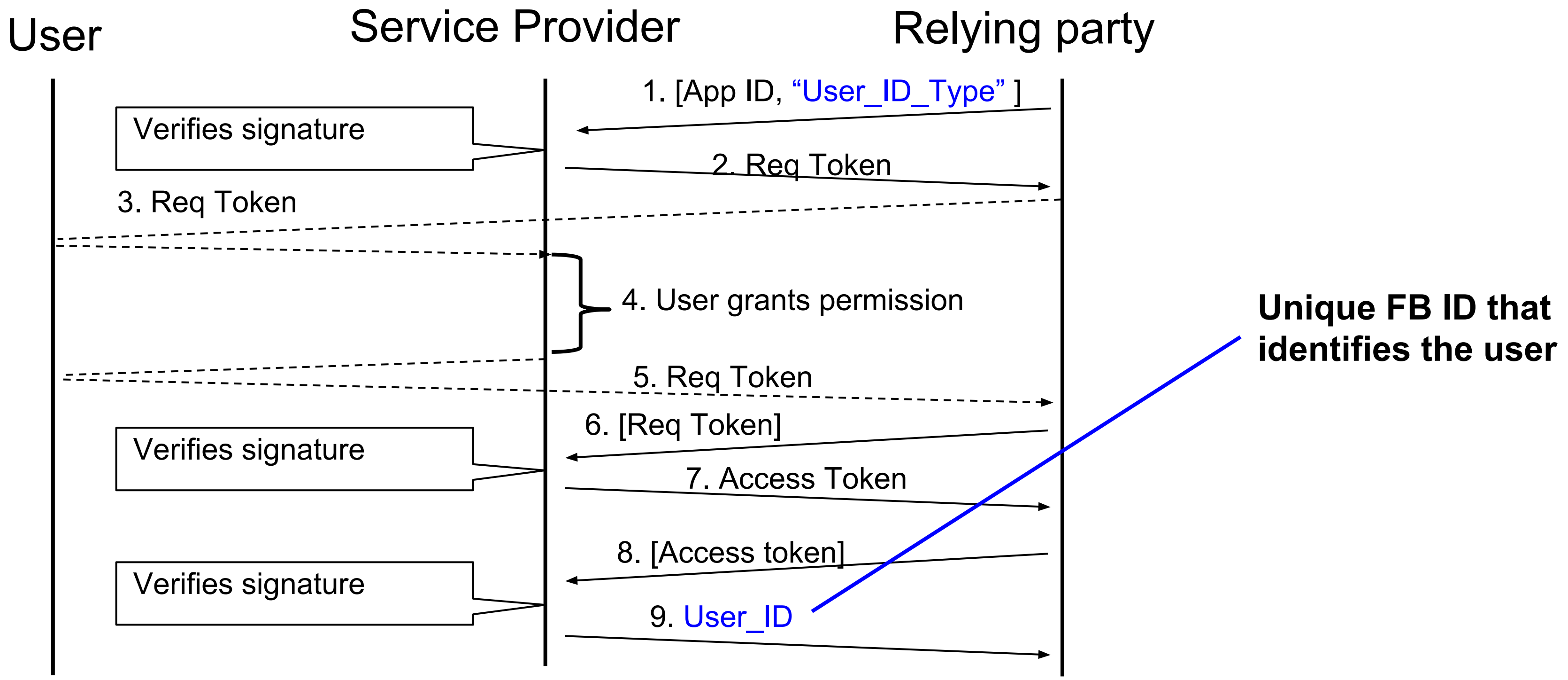
Differences between web and mobile platforms

1. Different redirection mechanisms
   - HTTP 302 Vs. iOS custom schemes or intents
2. Lack of application identity
   - No concept of “Origin” for mobile applications
3. Client-side heavy protocol logic
   - Observation: mobile apps have heavier clients

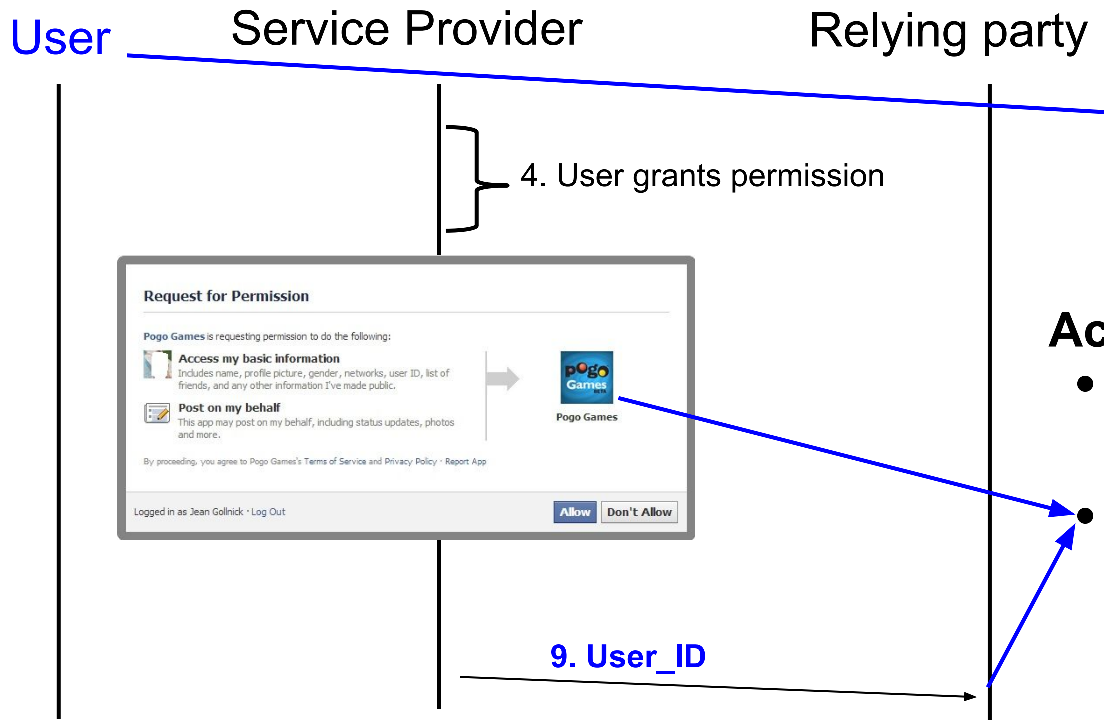
Motivation

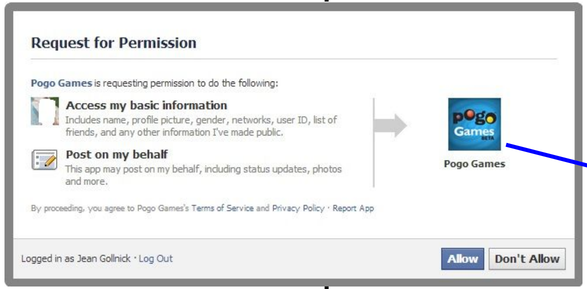
Oauth is designed for authorization, but is used for authentication

OAuth 1.0 - Authentication



OAuth 1.0 authentication security




Request for Permission

Pogo Games is requesting permission to do the following:

Access my basic information
Includes name, profile picture, gender, networks, user ID, list of friends, and any other information I've made public.

Post on my behalf
This app may post on my behalf, including status updates, photos and more.

By proceeding, you agree to Pogo Games's Terms of Service and Privacy Policy · Report App

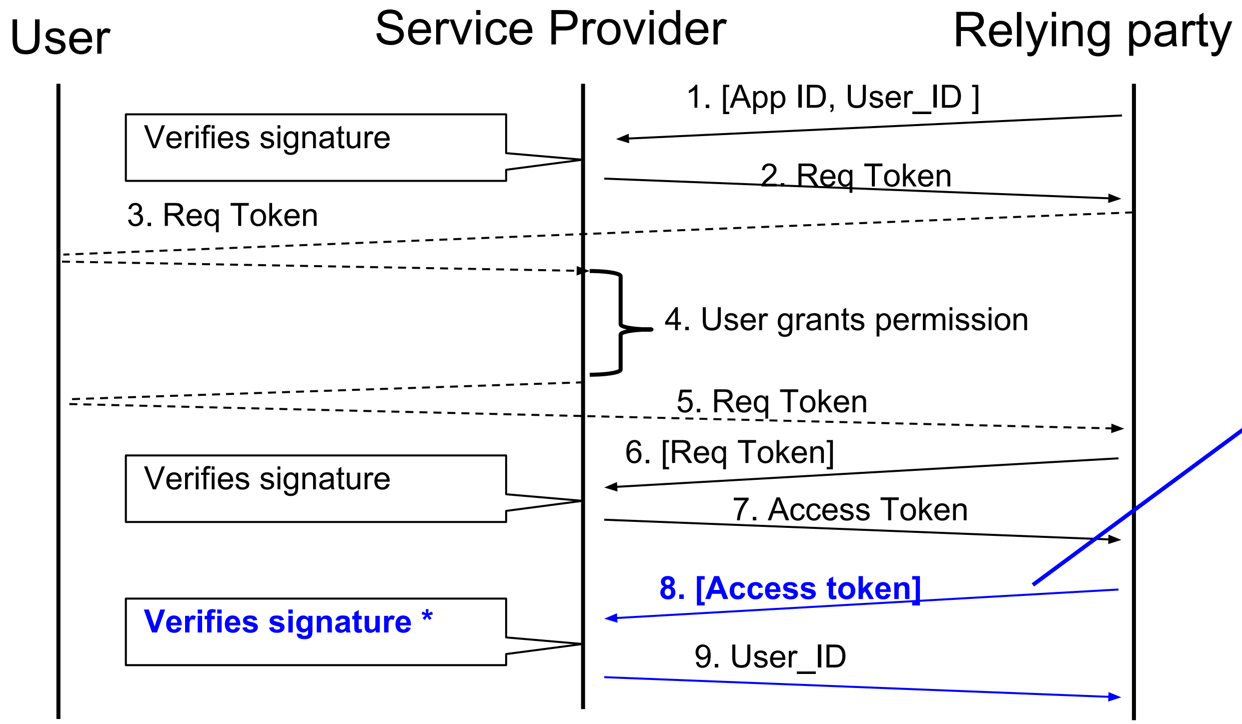
Logged in as Jean Gollnick · Log Out

User is untrusted

Achieving security:

- Sent from the service provider and **cannot be tampered** by the user.
- **Only the approved relying party** can receive the user's ID.

OAuth 1.0 authentication security



Cannot be tampered by the attacker

* Signature verification binds an access token to the issued party

General OAuth Security

- User’s consent
    - Need to display the app’s name and icon, and scopes that we be granted
- Session identifications
    - Need to use state token to identify the session, or else attackers might trick users to log in on behalf of them

Take a way

- Service provider
  - Checking logic on the server side
  - Consent page
  - Set short term cookie for webview
- Relying party