1000 Ways to Die in Mobile OAuth

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

Demos

Breaking Instagram Authentication
Covertly obtaining all permissions from Tencent
Autocode CSRF
Stealing Facebook cookie

What?

- In 2014, Studied OAuth usage in 200 Android/iOS OAuth applications.
    - **60% were implemented incorrectly.**

Quora

GitHub

- In 2016, these problems are **not fixed**, and there are new attacks.

How bad are the attacks?

- Impersonate a legitimate service (e.g., pinterest)
- Access to all user content on a service (e.g., Instagram)
- Stealing Facebook cookies
- Login CSRF
- **Full account compromise**

Why can’t developers use OAuth securely?

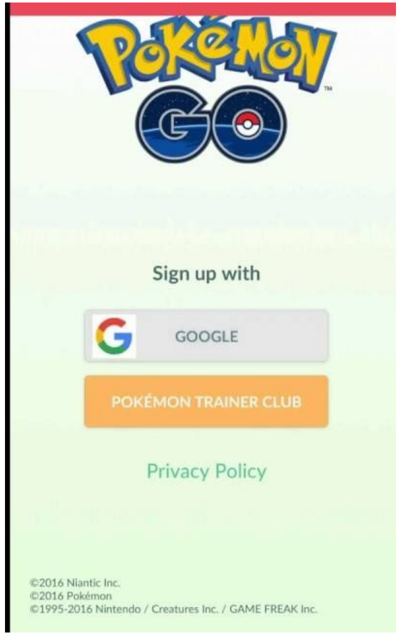
- Confusion between “authorization” and “authentication”
- Don’t know who to trust
  - Is my mobile app trustworthy?
- OAuth spec is too broad and confusing
  - 71 page threat model for OAuth 2.0??
- Requires collaboration from multiple parties
- OAuth spec is not written for mobile apps

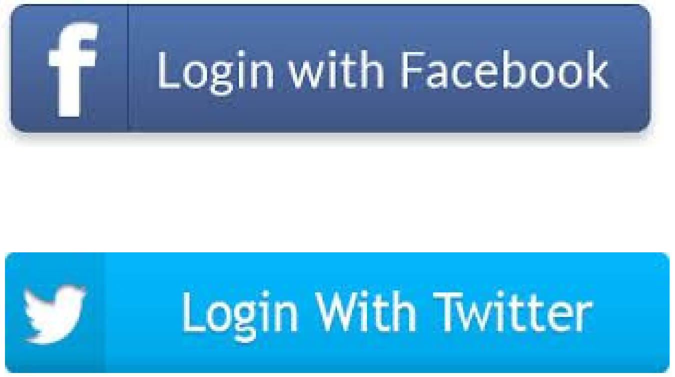
Vulnerabilities in this talk

- Locally stored secrets
- Locally store secrets + Evil redirect URL
- Overwrite Redirect URL in Mobile
- Using OAuth2 Implicit Flow for Authentication
- Provider not verify authorization code
- Lack of Consent Information
- Not using State Token
- WebView Cookies

What is OAuth?

What is OAuth?





A Protocol for Authorization

Three parties in OAuth

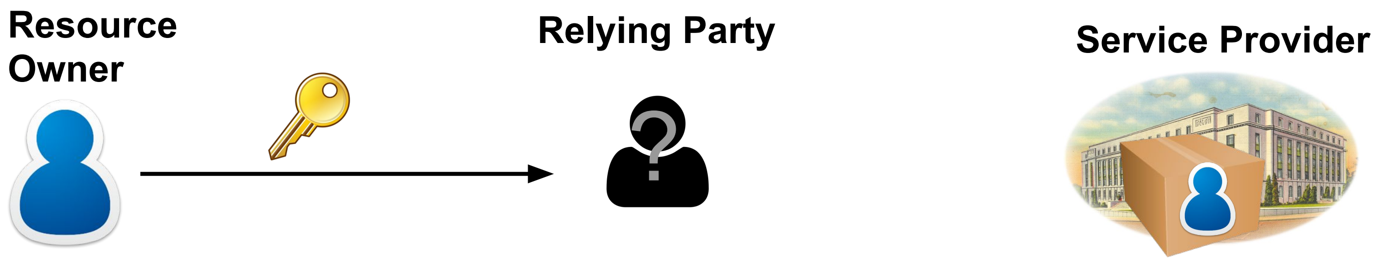
Resource Owner/End User

Service Provider

Relying Party

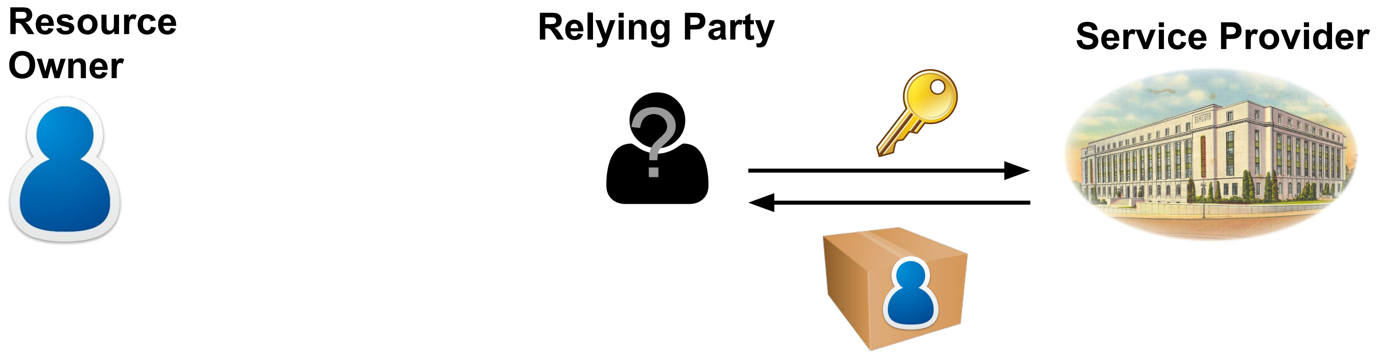
Authorization

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



Authorization

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



Brief history of OAuth

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

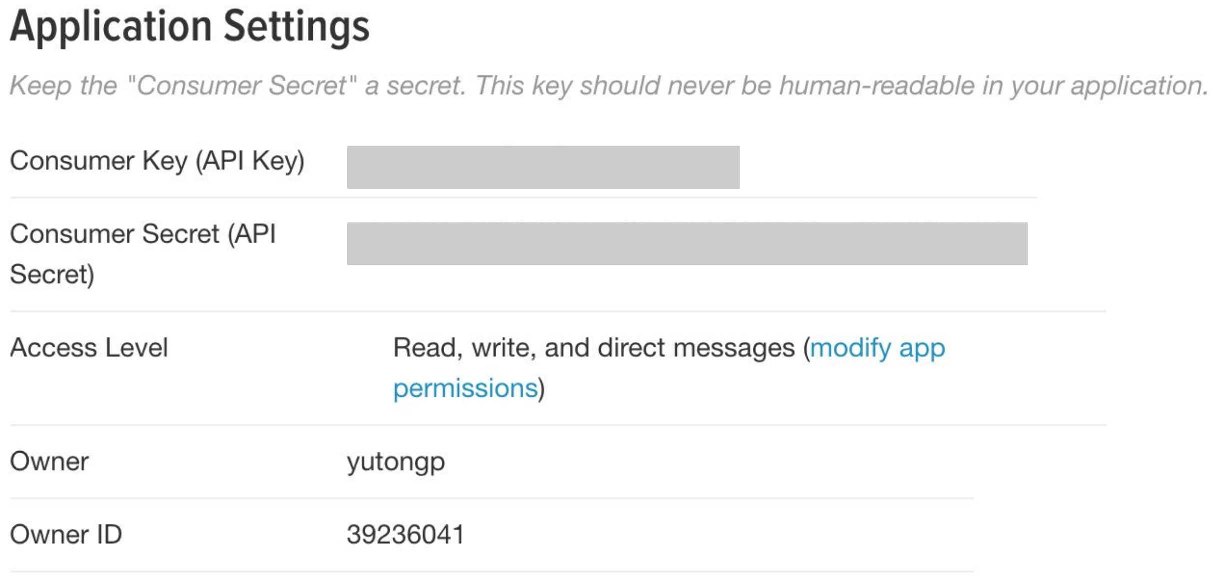
Used by real world mobile apps

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

OAuth 1.0

OAuth 1.0

Register your application on the service provider

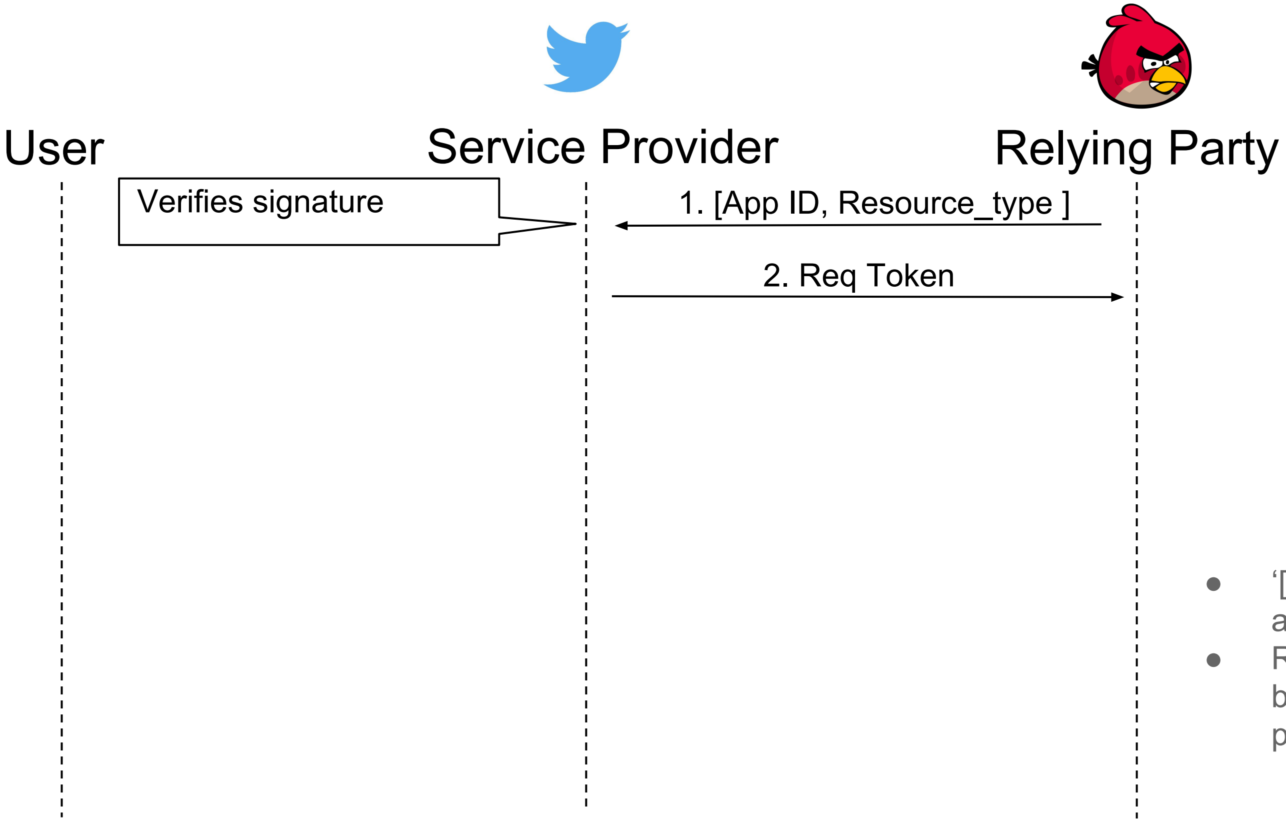


Application Settings

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

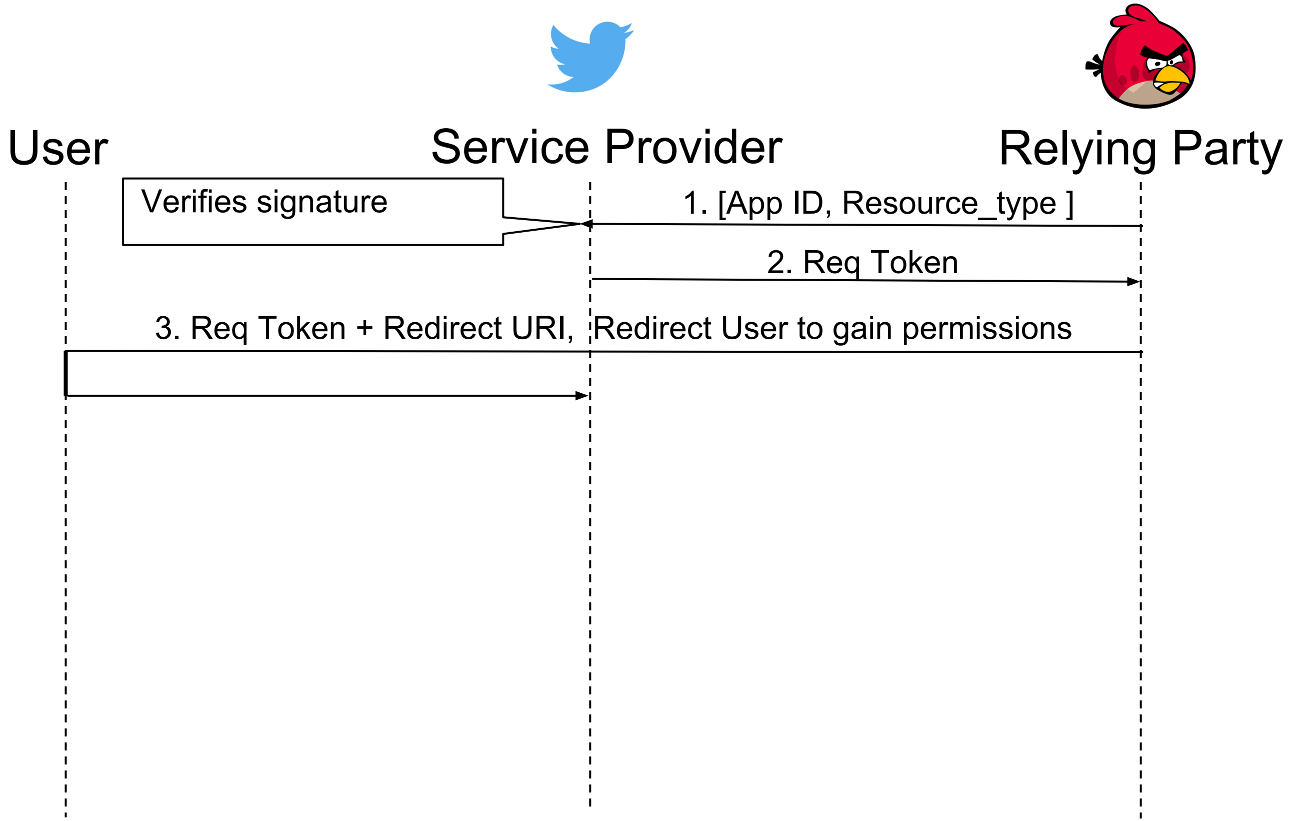
Consumer Key (API Key)	[redacted]
Consumer Secret (API Secret)	[redacted]
Access Level	Read, write, and direct messages (modify app permissions)
Owner	yutongp
Owner ID	39236041

OAuth 1.0

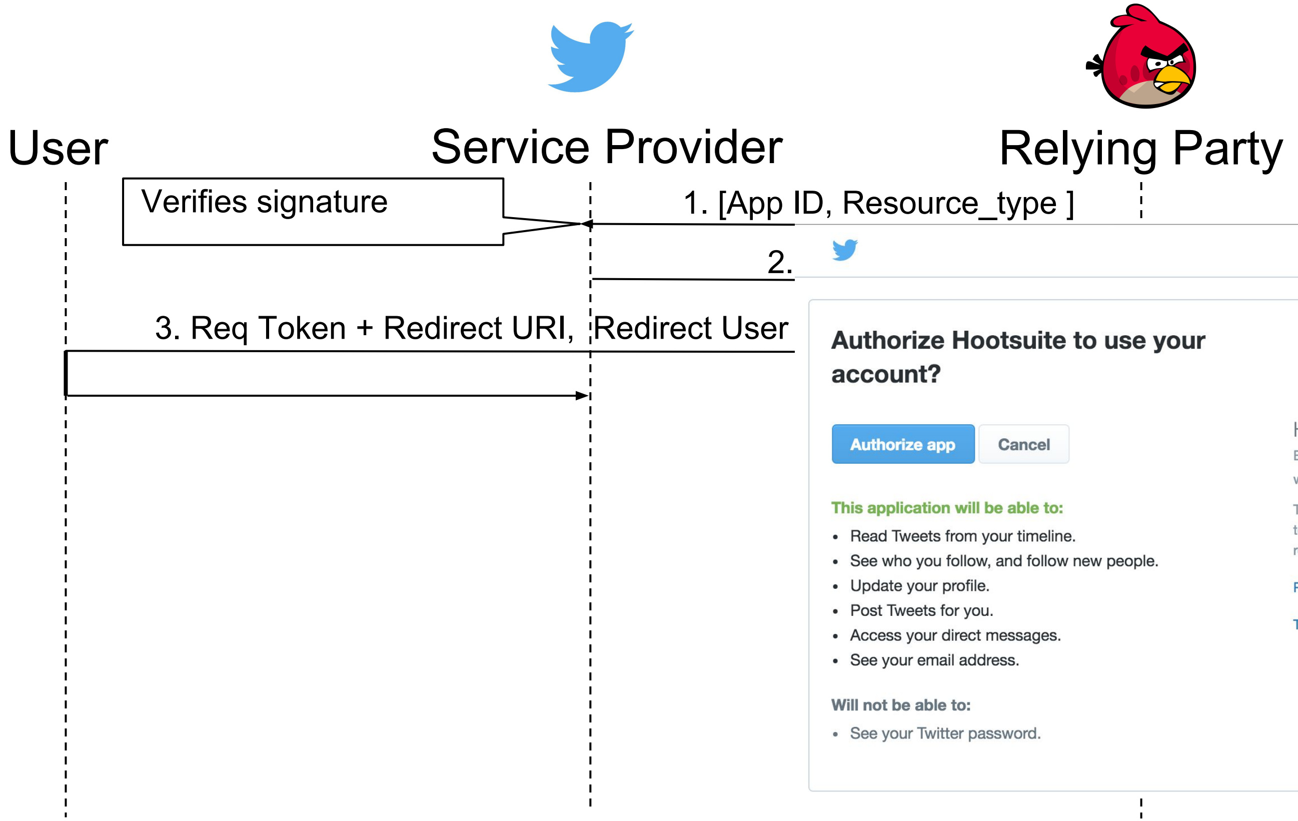


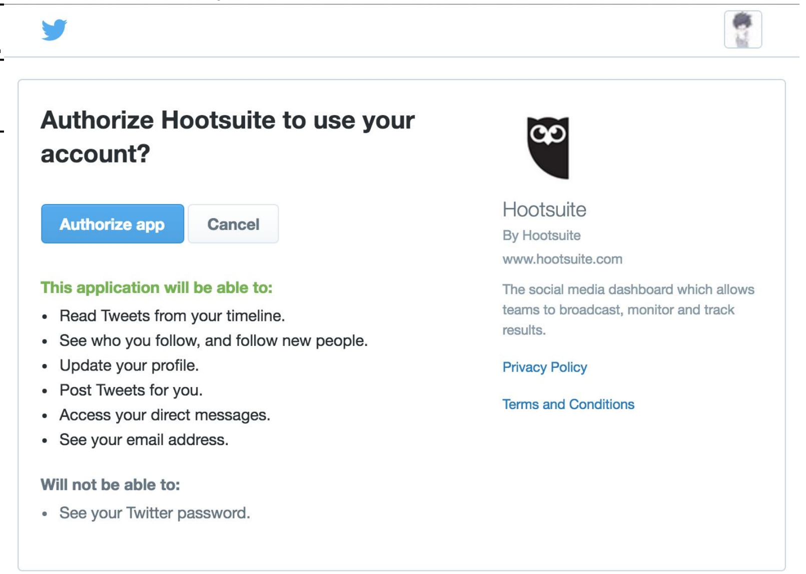
- '[]' means signed with app secret
- Resource_type can be: email, user's photos, etc

OAuth 1.0


User
Service Provider
Relying Party
Verifies signature
1. [App ID, Resource_type]
2. Req Token
3. Req Token + Redirect URI, Redirect User to gain permissions

OAuth 1.0


User
Service Provider
Relying Party
Verifies signature
1. [App ID, Resource_type]
2.
3. Req Token + Redirect URI, Redirect User



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.

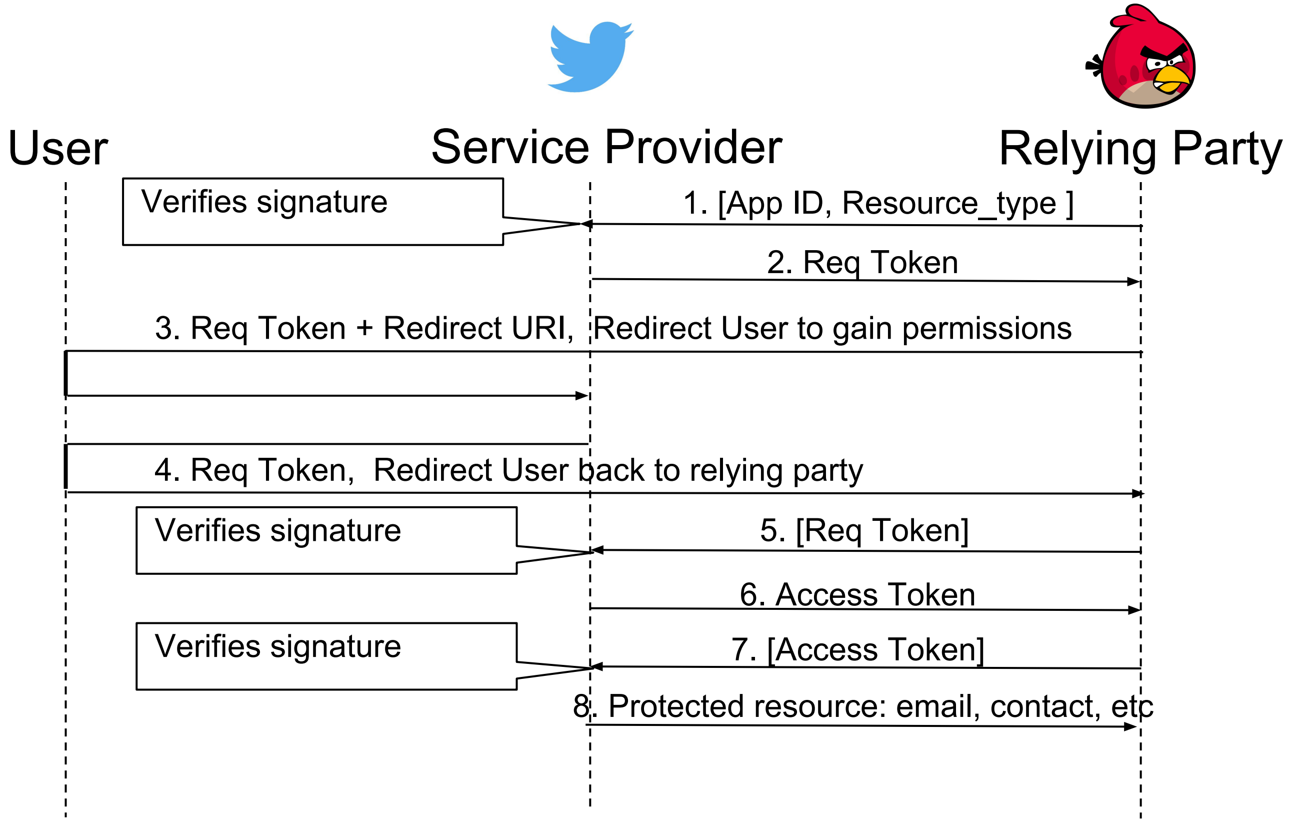
Hootsuite
By Hootsuite
www.hootsuite.com

The social media dashboard which allows teams to broadcast, monitor and track results.

Privacy Policy

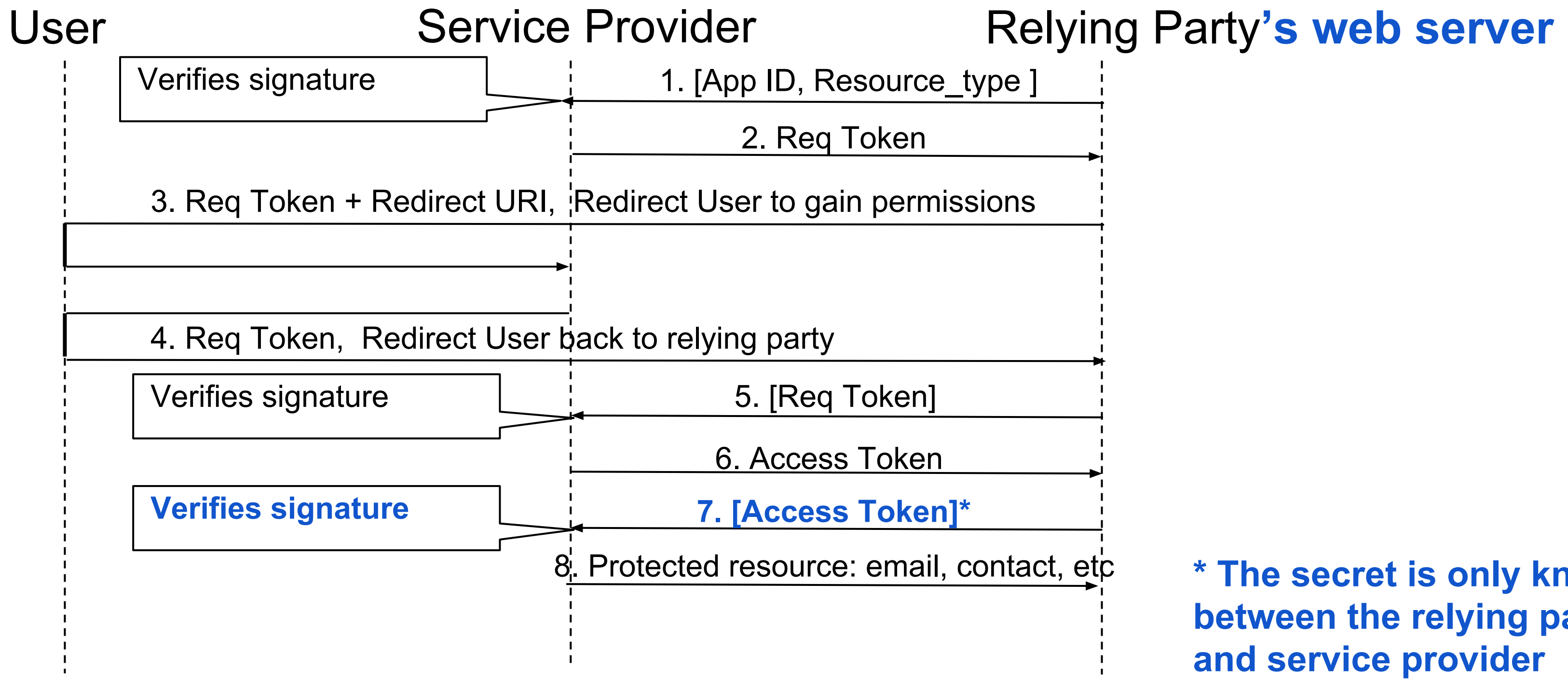
Terms and Conditions

OAuth 1.0


User
Service Provider
Relying Party
Verifies signature
1. [App ID, Resource_type]
2. Req Token
3. Req Token + Redirect URI, Redirect User to gain permissions
4. Req Token, Redirect User back to relying party
Verifies signature
5. [Req Token]
6. Access Token
Verifies signature
7. [Access Token]
8. Protected resource: email, contact, etc

How to know the relying party is the one user grant permissions to?

OAuth 1.0 Security - Relying Party Identity

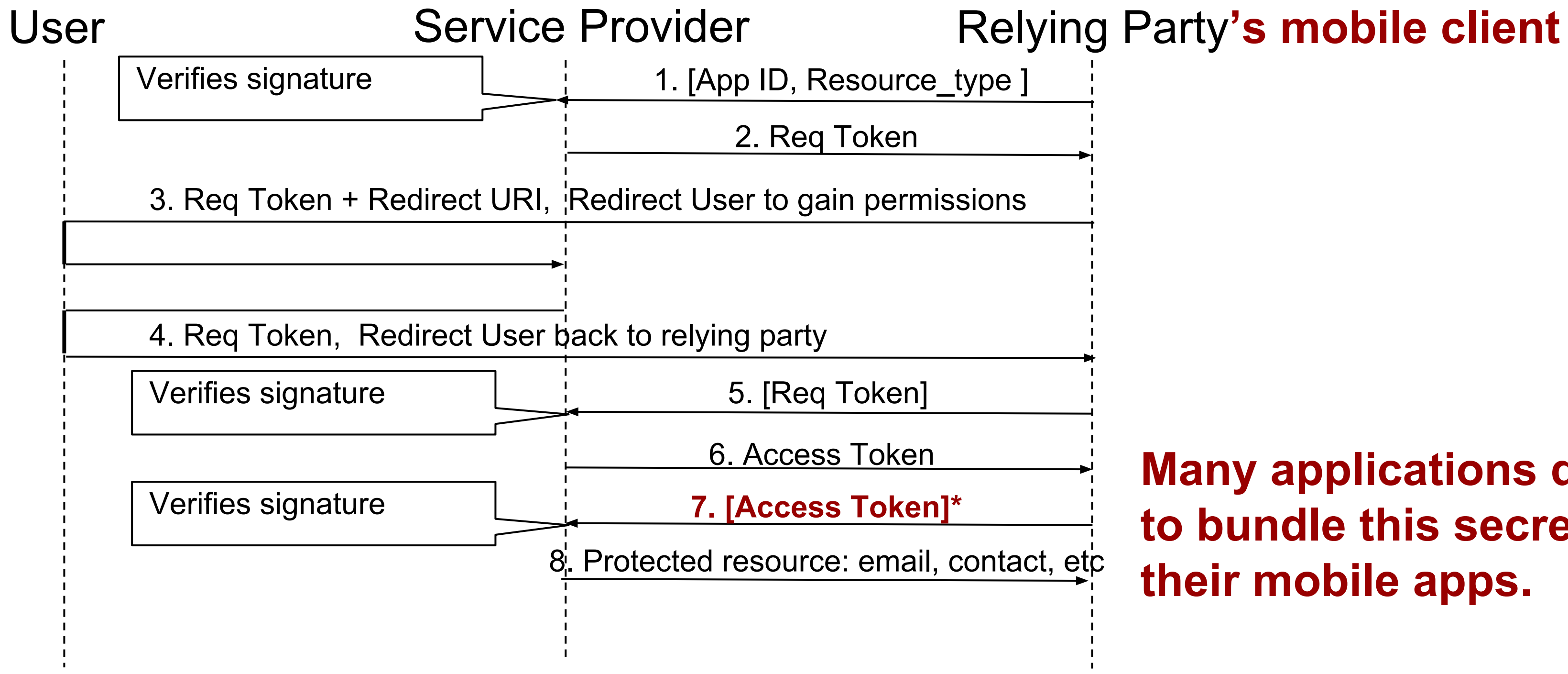

User
Service Provider
Relying Party's web server
Verifies signature
1. [App ID, Resource_type]
2. Req Token
3. Req Token + Redirect URI, Redirect User to gain permissions
4. Req Token, Redirect User back to relying party
Verifies signature
5. [Req Token]
6. Access Token
Verifies signature
7. [Access Token]*
8. Protected resource: email, contact, etc

*** The secret is only known between the relying party and service provider**

Vulnerability I

Locally stored secrets

Vulnerability - Locally stored secrets

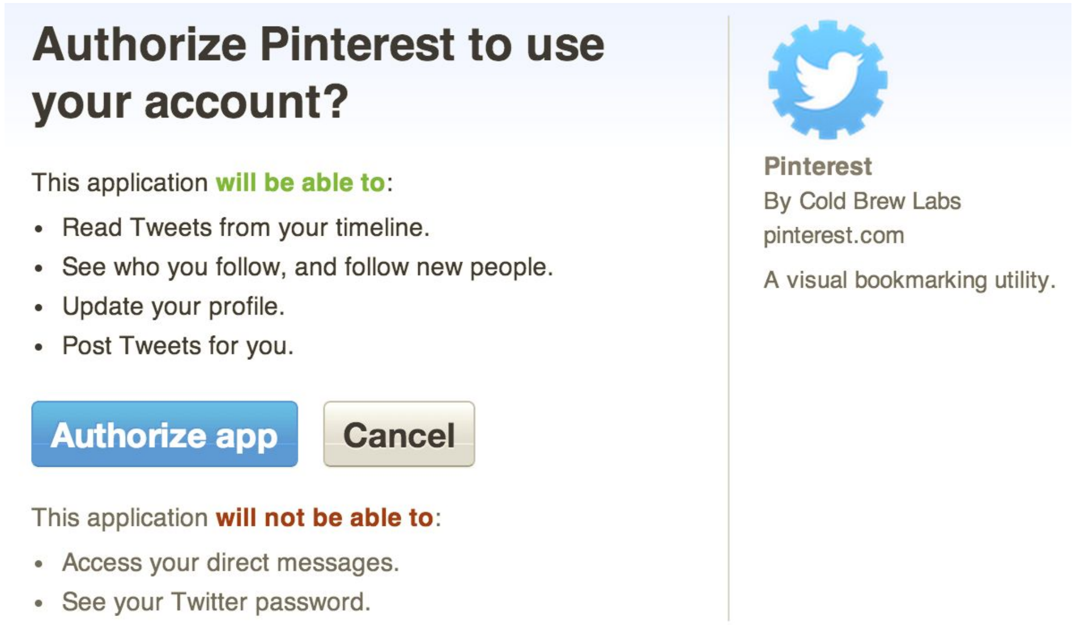

User
Service Provider
Relying Party's mobile client
Verifies signature
1. [App ID, Resource_type]
2. Req Token
3. Req Token + Redirect URI, Redirect User to gain permissions
4. Req Token, Redirect User back to relying party
Verifies signature
5. [Req Token]
6. Access Token
Verifies signature
7. [Access Token]*
8. Protected resource: email, contact, etc

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

Vulnerability - Locally stored secrets

```
src/com/pinterest/activity/signin/TwitterAuthActivity.java
26:          return (new ServiceBuilder()).
provider(org/scribe/builder/api/TwitterApi).
apiKey("Zr6TVkMT2KhKIZwERTB8IQ").
apiSecret("WYmVb7f0a************************X83gNCGQ0").
callback("oauth://twitter").
build();
```

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.

Impacts - Locally stored secrets

- Malicious app can impersonate a benign app
- Break authorization

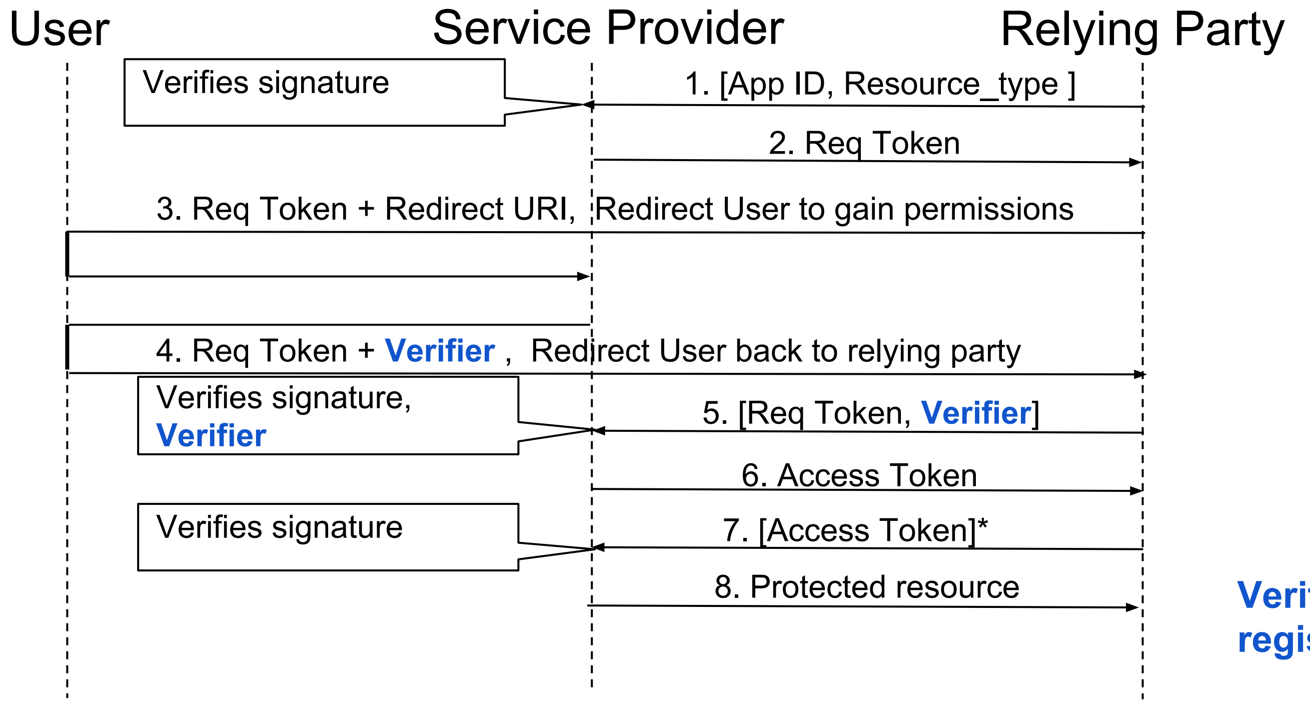
Vulnerability - Locally stored secrets

- 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.
- **Both are not using twitter login anymore...**

Do it right

- Relying Party
    - Do not bundle client secret into the mobile client

OAuth1.0a Security Improvement


User
Service Provider
Relying Party
Verifies signature
1. [App ID, Resource_type]
2. Req Token
3. Req Token + Redirect URI, Redirect User to gain permissions
4. Req Token + **Verifier** , Redirect User back to relying party
Verifies signature, **Verifier**
5. [Req Token, **Verifier**]
6. Access Token
Verifies signature
7. [Access Token]*
8. Protected resource

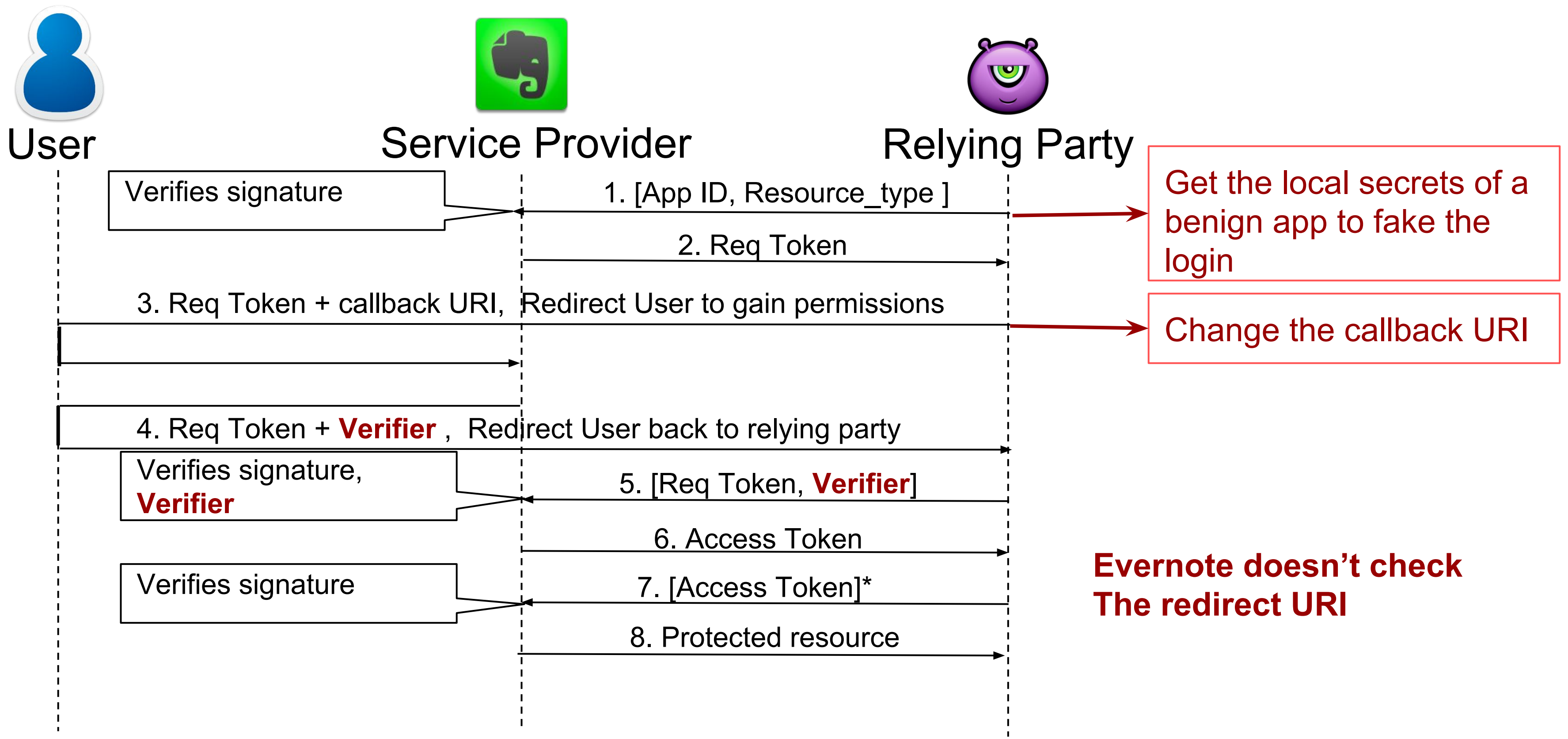
OAuth 1.0a

Verifier is only sent to the registered redirect URL

Vulnerability II

Locally store secrets 1.0a

Vulnerability - Locally store secrets + Evil redirect URL


User
Service Provider
Relying Party
Verifies signature
1. [App ID, Resource_type]
2. Req Token
3. Req Token + callback URI, Redirect User to gain permissions
4. Req Token + **Verifier** , Redirect User back to relying party
Verifies signature, **Verifier**
5. [Req Token, **Verifier**]
6. Access Token
Verifies signature
7. [Access Token]*
8. Protected resource
Get the local secrets of a benign app to fake the login
Change the callback URI

Evernote doesn't check The redirect URI

Do it right

- Service Provider
  - Register the redirect URI and check the redirect URI
- Relying Party
  - Do not bundle client secret into the mobile client

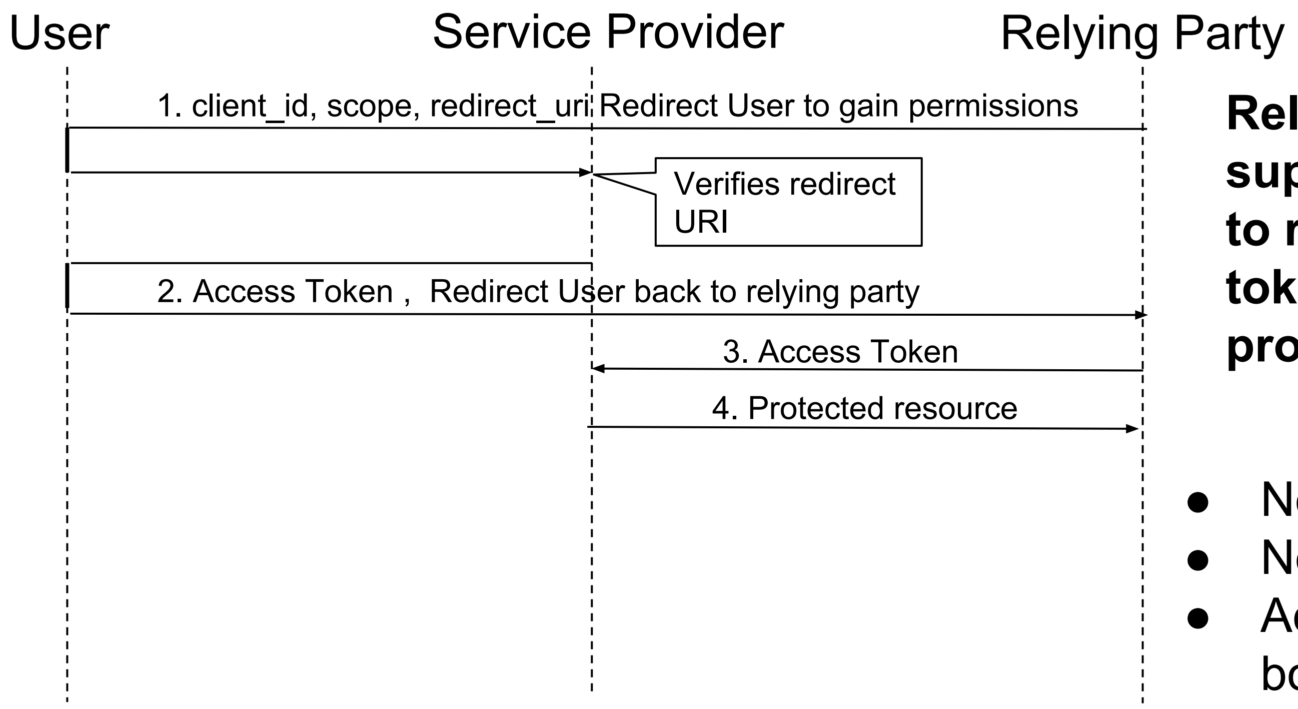
OAuth1.0,
OAuth1.0a, OAuth...?

OAuth 2

OAuth 2.0

- **Implicit grant**
- **Authorization code grant**
- Resource owner password credentials
- Client credentials

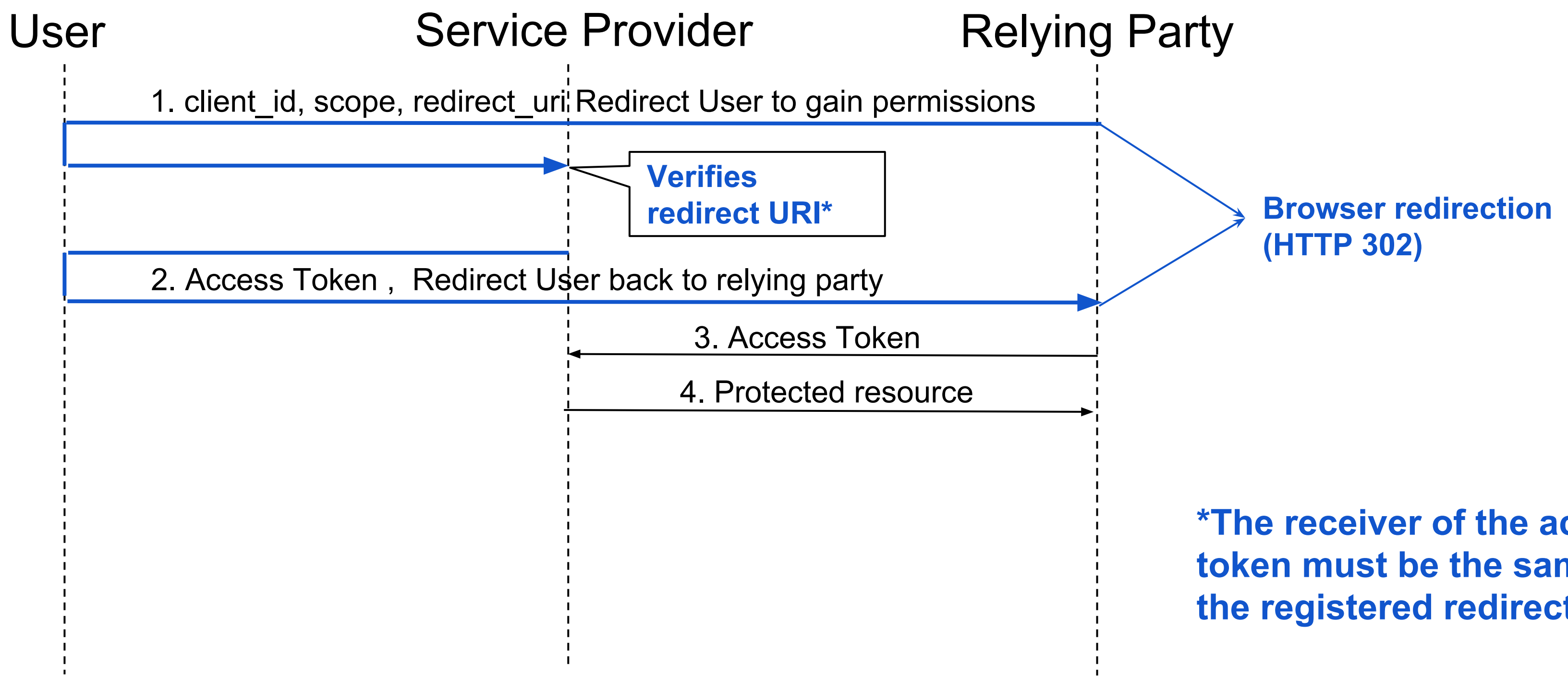
OAuth 2.0 implicit flow


User
Service Provider
Relying Party
1. client_id, scope, redirect_uri Redirect User to gain permissions
Verifies redirect URI
2. Access Token , Redirect User back to relying party
3. Access Token
4. Protected resource

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

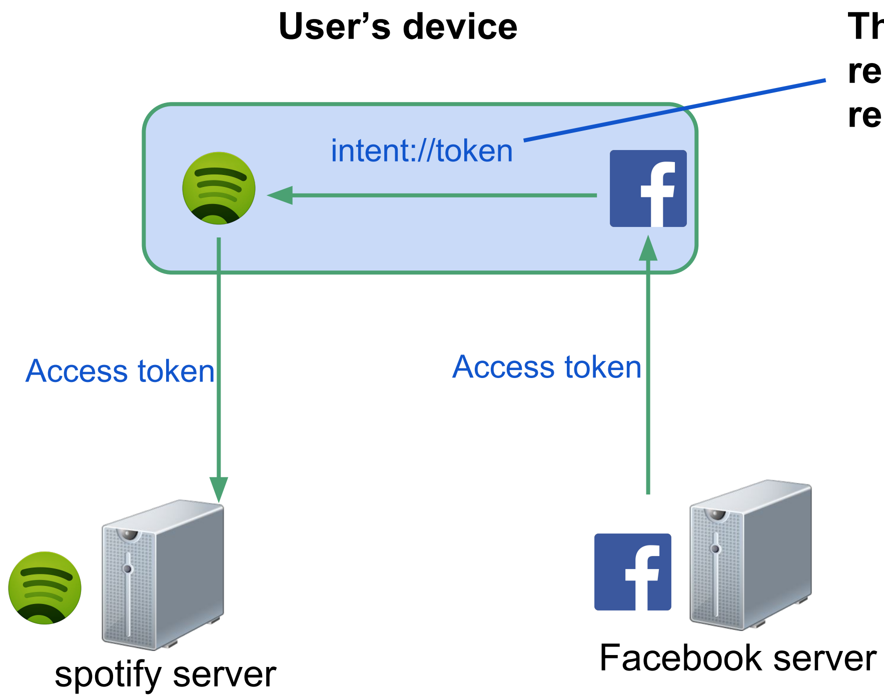
- No relying party secret!
- No signature/encryption
- Access token is not bound to a RP

OAuth 2.0 implicit flow Security - Handling redirection


User
Service Provider
Relying Party
1. client_id, scope, redirect_uri Redirect User to gain permissions
Verifies redirect URI*
Browser redirection (HTTP 302)
2. Access Token , Redirect User back to relying party
3. Access Token
4. Protected resource

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

OAuth 2.0 implicit flow Security - Handling redirection

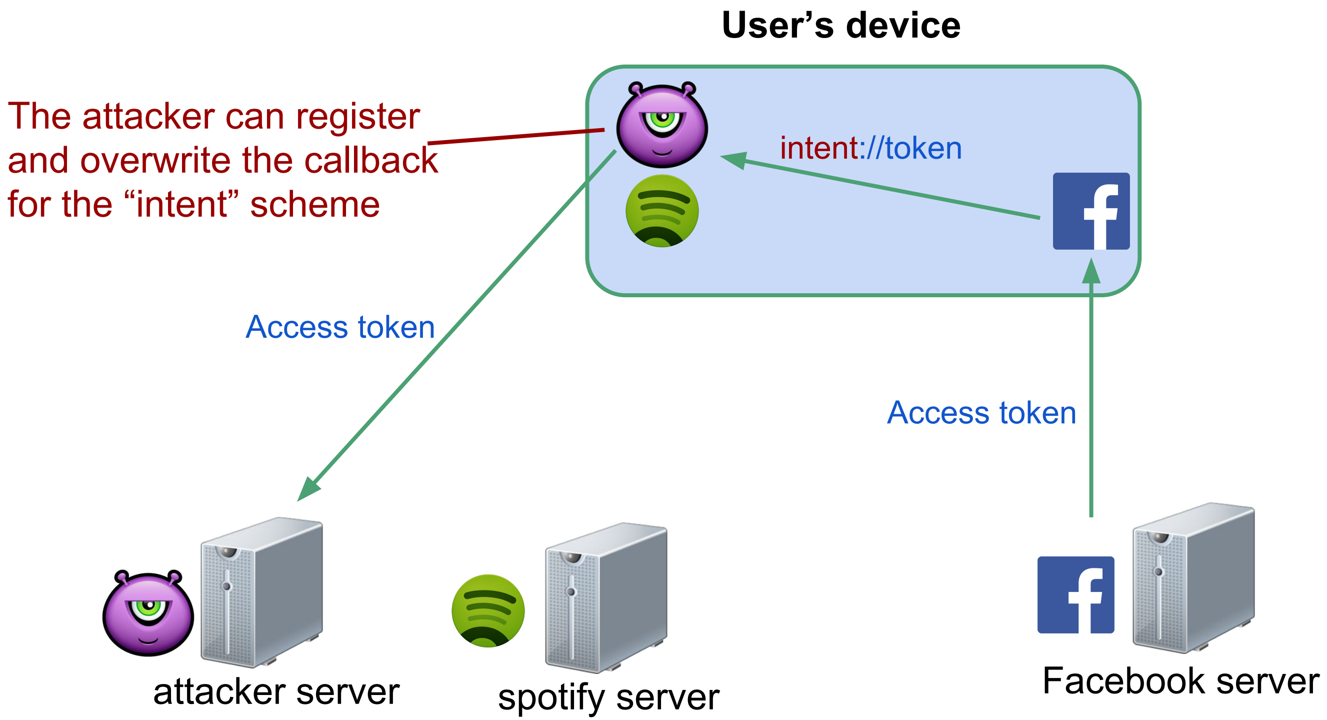


The “intent” URI- scheme is registered by the receiving relying party application

Vulnerability III

Overwrite Redirect URL in Mobile

Vulnerability - Overwrite Redirect URL in Mobile



Impact - Overwrite Redirect URL

- Attacker can access user’s Facebook data without consent.

Do it right

- 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.

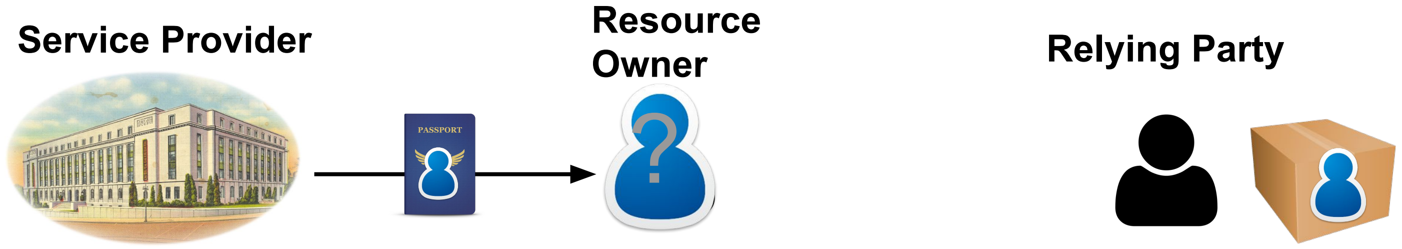
```
relying_party =  Activity.getCallingPackage();
dev_key_hash = getPackageManager().
          getPackageInfo(relying_party, PackageManager.GET_SIGNATURES);
```

Authorization VS Authentication

Authorization VS Authentication

Authentication

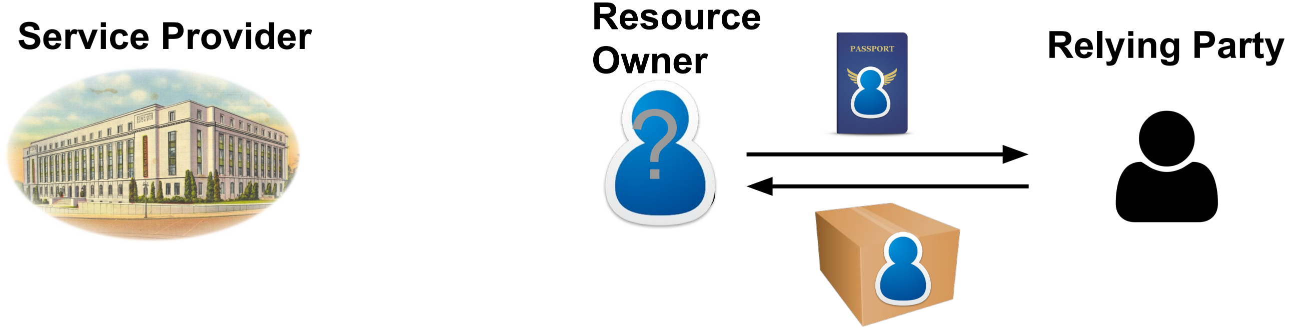
A process for **a user to prove his or her identity to a relying party**, utilizing his or her existing session with the service provider.



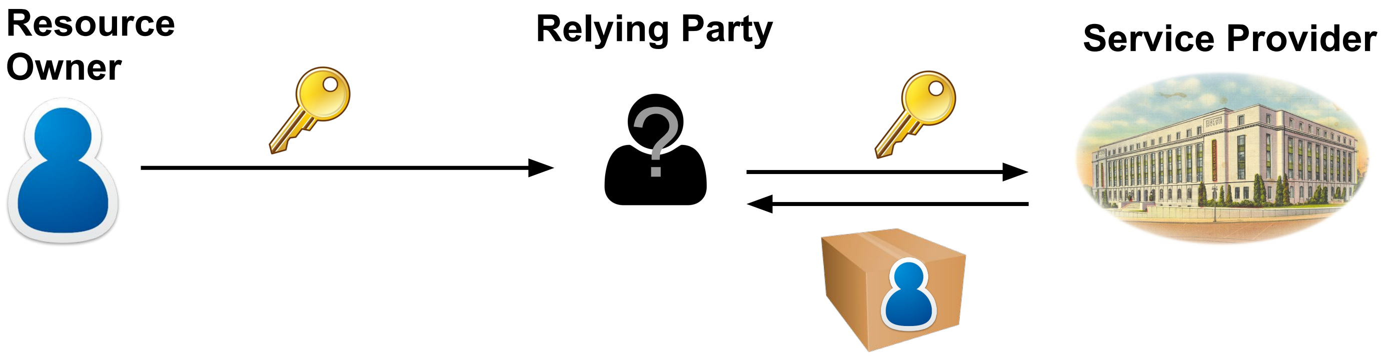
Authorization VS Authentication

<u>Authentication</u>

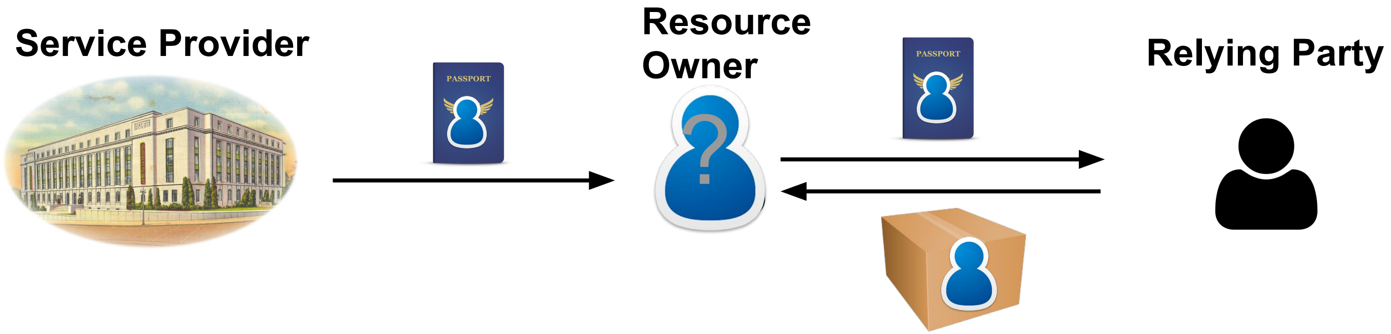
A process for **a user to prove his or her identity to a relying party**, utilizing his or her existing session with the service provider.



Authorization



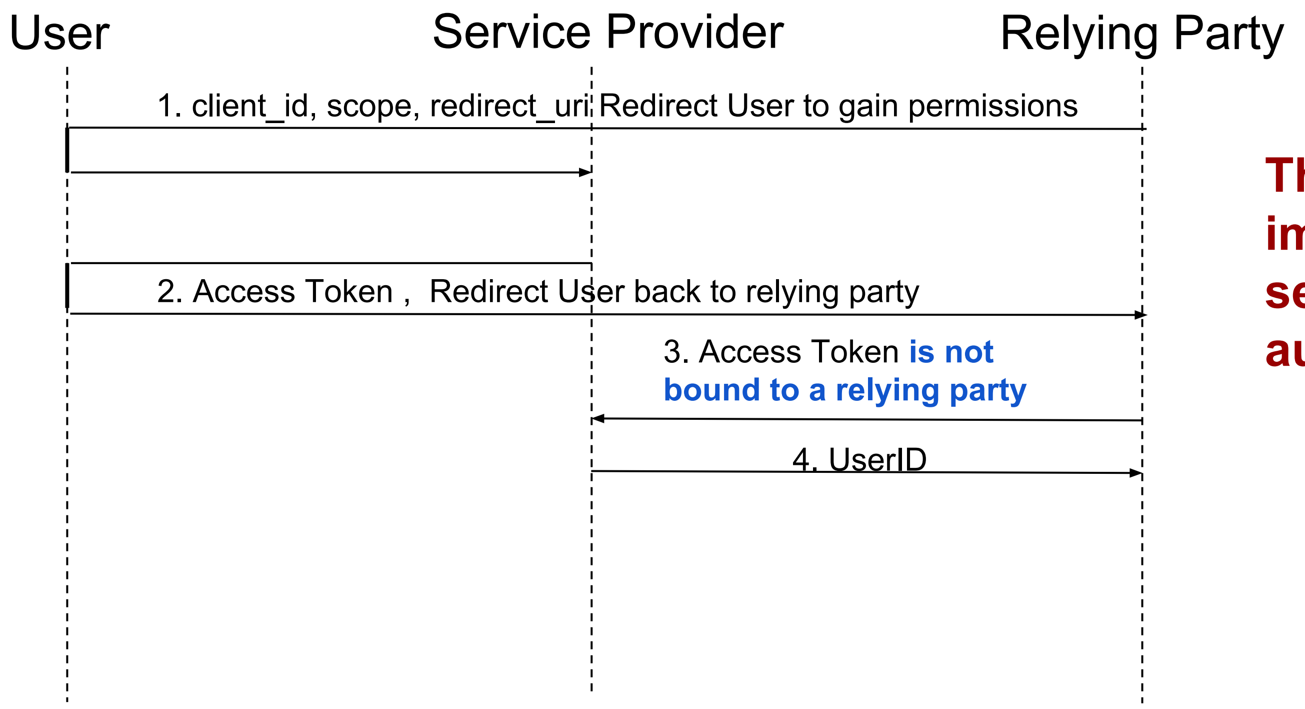
Authentication



Vulnerability IV

Using OAuth2 Implicit Flow for Authentication

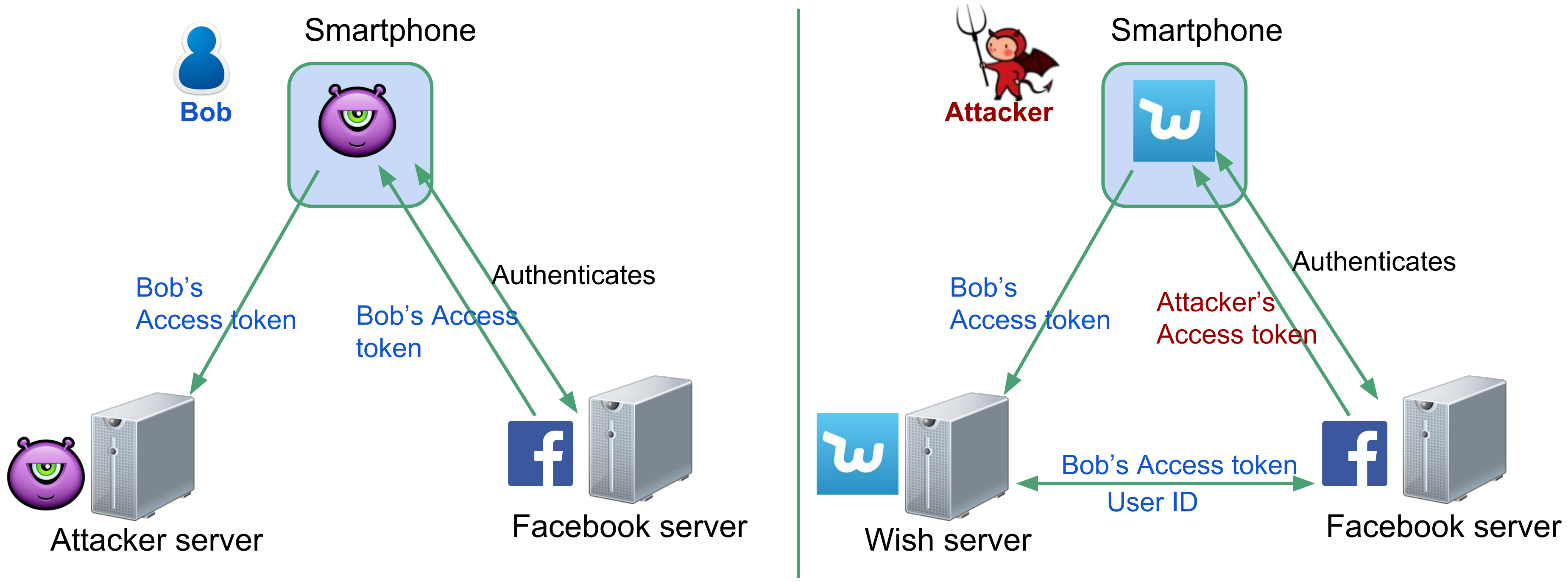
Vulnerability - Using authorization flow for authentication


User
Service Provider
Relying Party
1. client_id, scope, redirect_uri Redirect User to gain permissions
2. Access Token , Redirect User back to relying party
3. Access Token **is not bound to a relying party**
4. UserID

The OAuth 2.0 implicit flow is not secure for authentication

Vulnerability - Using authorization flow for authentication

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



Vulnerability - Using authorization flow for authentication

Response From Facebook

```
<script
type="text/javascript">window.location.href="fbconnect:\/\/succe
ss#granted_scopes=user_birthday\u00252Cuser_hometown\u00252Cuser
_location\u00252Cuser_likes\u00252Cuser_friends\u00252Cemail\u00
252Ccontact_email\u00252Cpublic_profile&denied_scopes=&access_to
ken=XXXXXXXXXXX&expires_in=5182633";</script>
```

Request from Wish APP

```
GET /v2.2/me?access_token=XXXXXXXXXXX&format=json&sdk=android
...

{
        "id": "100007872092560",
        "birthday": "11\/25\/1989",
        "email": "yutong\u0040lockie.io",
        "first_name": "Yutong",
        "gender": "male",
        "last_name": "Pei",
        "link":
"https:\/\/www.facebook.com\/app_scoped_user_id\/100007872092560
\/",
        "locale": "en_US",
        "name": "Yutong Pei",
        "timezone": -7,
        "updated_time": "2014-02-22T02:45:44+0000",
        "verified": false
}
```

Impact - Using OAuth2 Implicit Flow for Authentication

- Full account compromise
  - Instagram in 2014

How to do better Authentication?

OpenID Connect

OpenID Connect

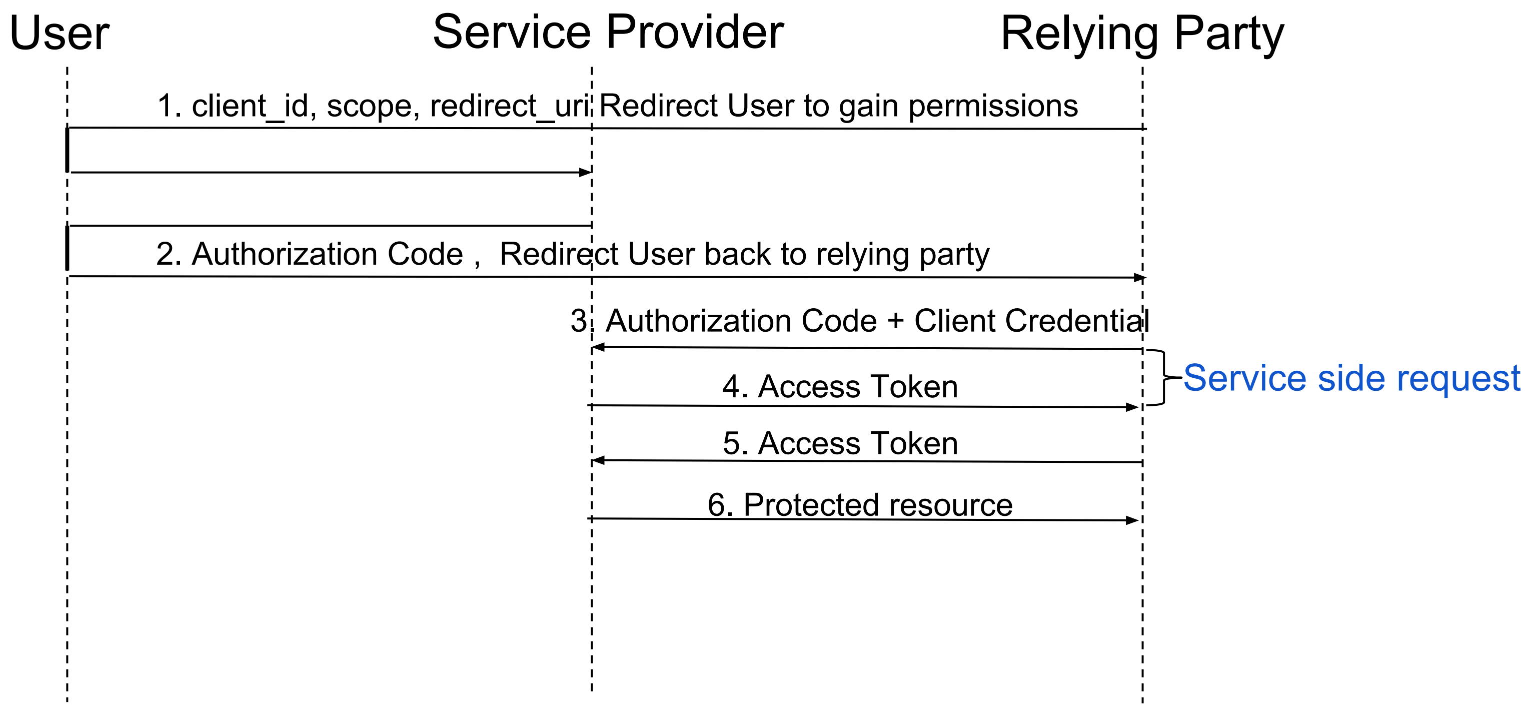
ID token - signed JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwic3ViIjoiMjQ4
Mjg5NzYxMDAxIiwiYXVkIjoiczZCaGRSa3F0MyIsIm5vbmNlIjoibi0wUzZfV3pBMk1qIiwiZXhwIjoxMzExMjgxOTcwLCJpY
XQiOjEzMTEyODA5NzAsImF0X2hhc2giOiI3N1FtVVB0alBmeld0RjJBbnBLOVJRIn0.VW_s1XIAkhlFTfx90VjofHjbRqM5ME
tMA5mlctc7dCE
```

Payload:

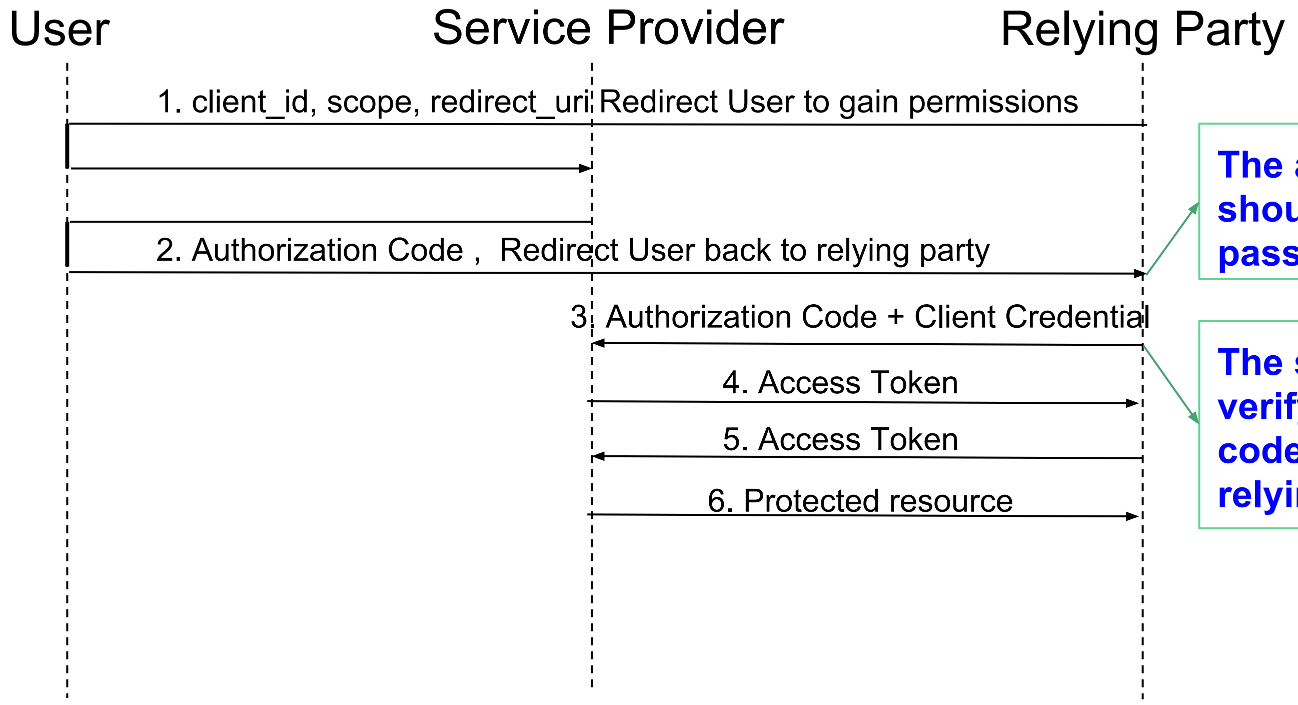
```
{
  "iss": "http://server.example.com",
  "sub": "248289761001",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "at_hash": "77QmUPtjPfzWtF2AnpK9RQ"
}
```

OAuth2 Code Authorization Flow



User
Service Provider
Relying Party
1. client_id, scope, redirect_uri Redirect User to gain permissions
2. Authorization Code , Redirect User back to relying party
3. Authorization Code + Client Credential
4. Access Token
Service side request
5. Access Token
6. Protected resource

Code Authorization Flow- Verify authorization code


User
Service Provider
Relying Party
1. client_id, scope, redirect_uri Redirect User to gain permissions
2. Authorization Code , Redirect User back to relying party
3. Authorization Code + Client Credential
4. Access Token
5. Access Token
6. Protected resource

The authorization code should be one-time password

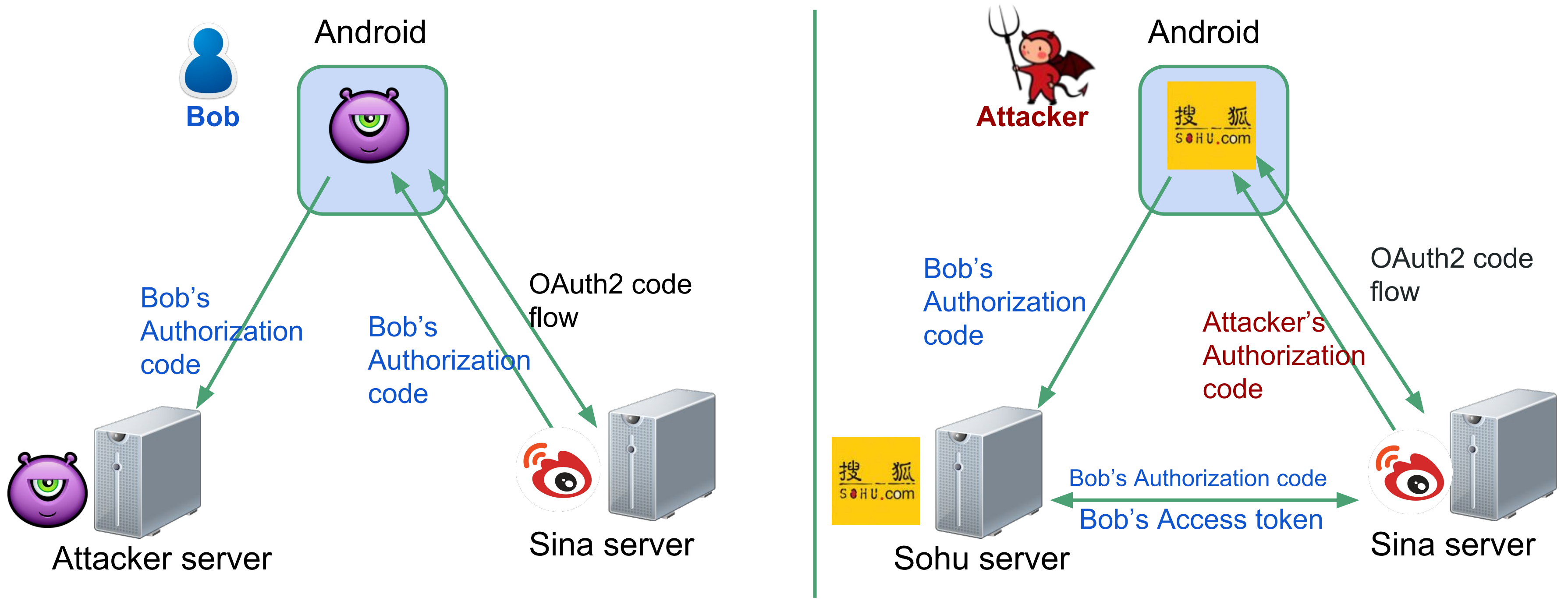
The service provider should verify that the authorization code belongs to the same relying party

Vulnerability V

Provider not verify authorization code

Vulnerabilities- Not verifying authorization code

Vulnerability in Sohu news app with Sina login:



Impact- Provider not verify authorization code

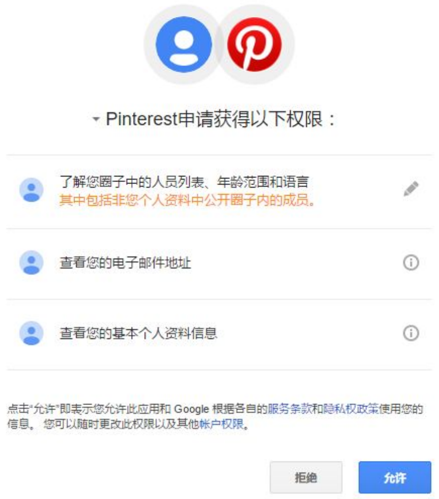
- Full account compromise

Do it right

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

Little bit more about Consent Page

A page that describes what the app requests from the user, and allows the user to approve or reject.



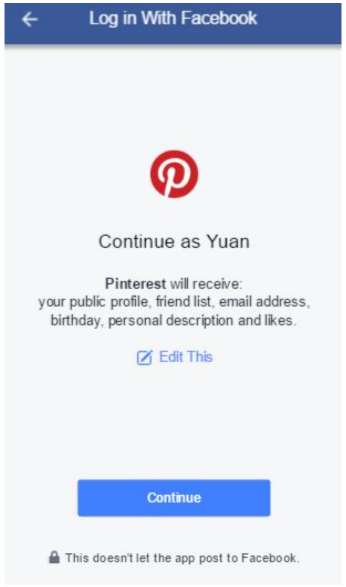
Pinterest申请获得以下权限：

了解您圈子中的人员列表、年龄范围和语言
其中包括非您个人资料中公开圈子内的成员。

查看您的电子邮件地址

查看您的基本个人资料信息

点击"允许"即表示您允许此应用和 Google 根据各自的服务条款和隐私权政策使用您的信息。 您可以随时更改此权限以及其他帐户权限。



Continue as Yuan

Pinterest will receive:
your public profile, friend list, email address, birthday, personal description and likes.

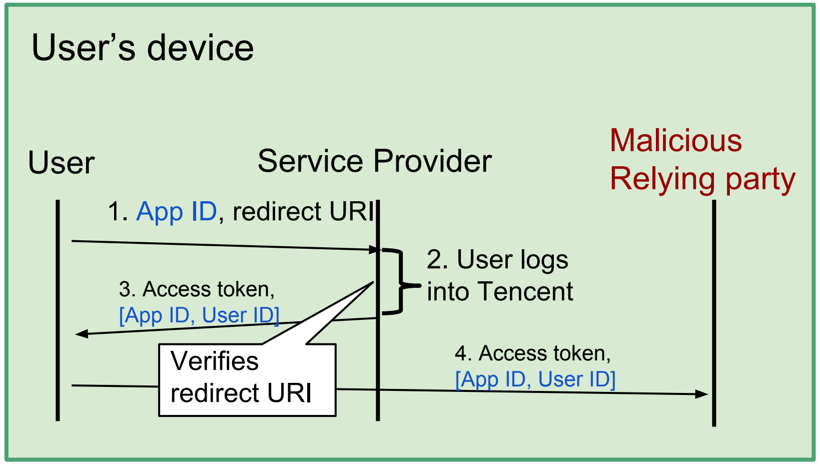
This doesn't let the app post to Facebook.

Vulnerability VI

Lack of Consent Information

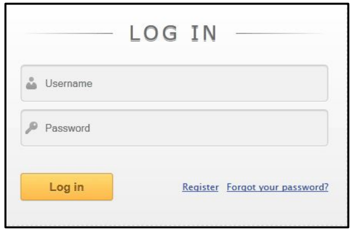
Vulnerability - Lack of Consent

- No information about relying party for Tencent mobile UI



App ID is public information

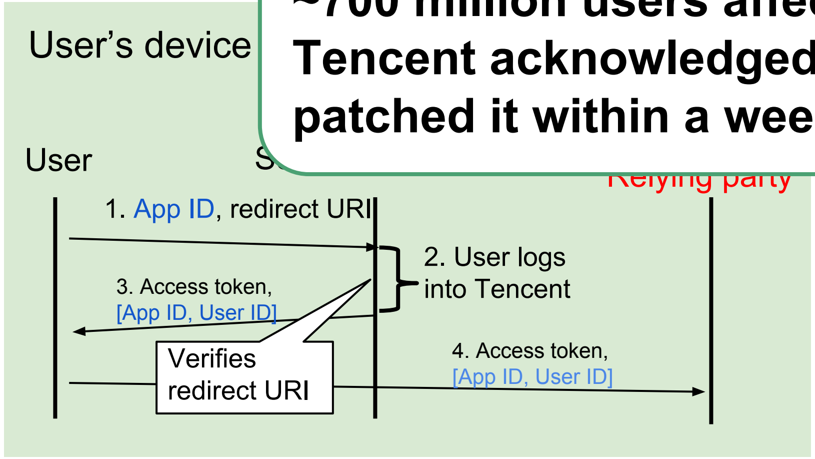
The user sees the same Tencent login-dialog for all relying parties

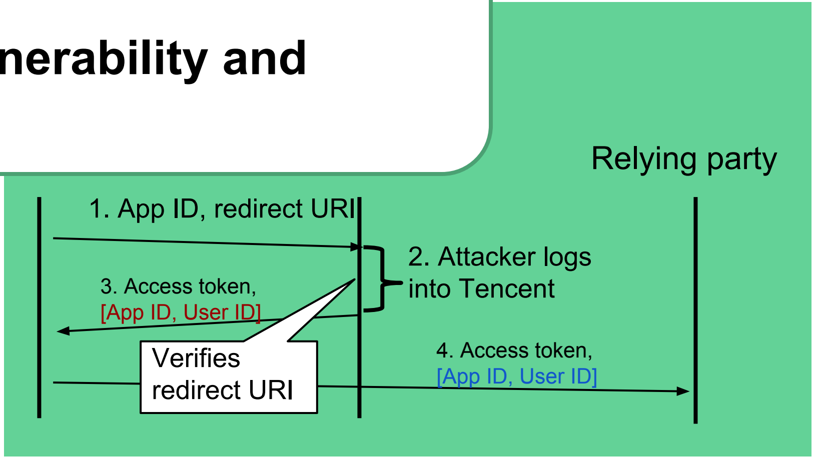


Vulnerability - Lack of Consent

- No information about relying party for T

Impact:
~700 million users affected.
Tencent acknowledged the vulnerability and patched it within a week.


User's device
User
S
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]


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

What should be included in Consent Page?

- User Name
- User Profile Image
- Client Name
- Client Icon
- Authorizating Permissions

OAuth Security - State Token

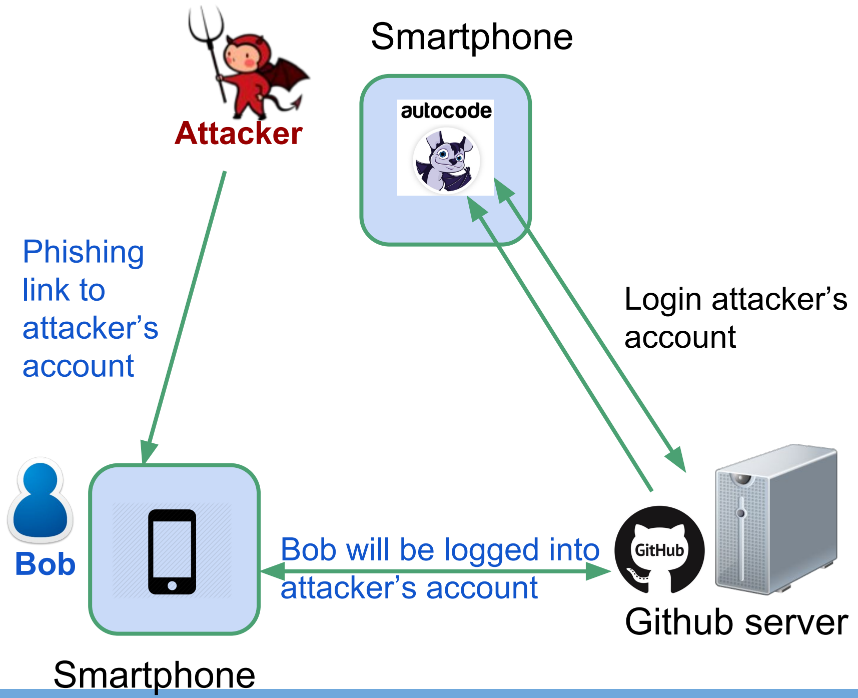
- Similar to CSRF token
- Ensure OAuth flow session integrity

Vulnerability VII

Not using State Token

Vulnerability - No State Token

Relying party should use state token to identify the login session



Vulnerability - Not using State Token

- Autocode attack

Attacker starts the OAuth flow on his machine:

https://github.com/login/oauth/authorize?client_id=2722d7d1c25dca9b3559&redirect_uri=https://app.autocode.run&scope=user:email,public_repo

Tricks the user into rendering this iframe:

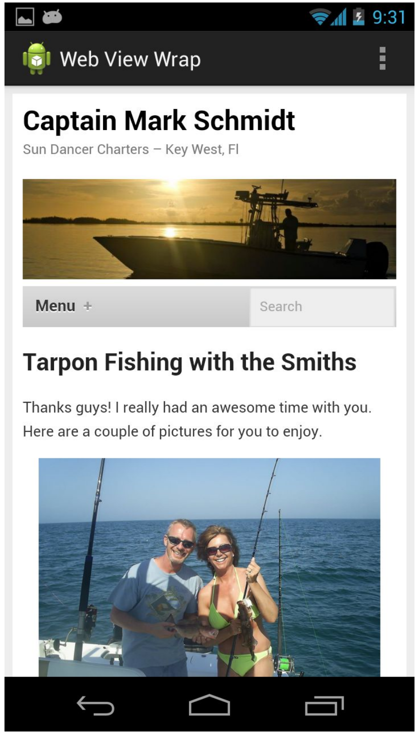
```
<iframe src="https://app.autocode.run/?code=f3ec63e21bb4841d01f9"
style="visibility:hidden;display:none"></iframe>
```

Do it right

- Service Provider
    - Support State Token
- Relying Party
    - Pass State Token to provider
    - Verify State Token when get back from provider

What is a WebView?

- Webview is a browser that is bundled into a mobile app.
- Useful for hybrid apps and embedding content
- Powerful, the app can control the website embedded in the webview (e.g., get cookies)



Captain Mark Schmidt

Sun Dancer Charters – Key West, Fl

Tarpon Fishing with the Smiths

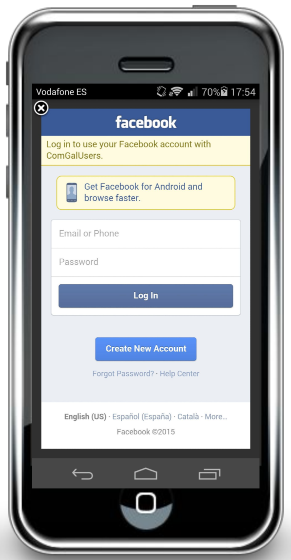
Thanks guys! I really had an awesome time with you. Here are a couple of pictures for you to enjoy.

Vulnerability VIII

WebView

Vulnerability- Cookie in Webview

Service provider set long term cookies in the webview, which allows attacker to log into user’s account



Log in to use your Facebook account with ComGalUsers.

Get Facebook for Android and browse faster.

Email or Phone
Password
Log In
Create New Account
Forgot Password? · Help Center
English (US) · Español (España) · Català · More...
Facebook ©2015

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.

Impact- Webview

- Full account compromise
  - **Currently no fix exists**

Summary

How to use mobile OAuth securely?

It’s very very hard

But is there anything we can do?

- Service Provider
  - Verify the Identity of the token/code receiver
  - Informative Consent page
  - Adopt OpenID connect for authentication

But is there anything we can do?

- 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 spec
  - Use SDK

Thank you

{eric.chen,yuan.tian,patrick.tague}@sv.cmu,edu
shuochen@microsoft.com
yutong@uber.com