OAuth
App Impersonation Attack

HOW TO LEAK A 100-MILLION-NODE SOCIAL GRAPH IN JUST ONE WEEK? - A REFLECTION ON OAUTH AND API DESIGN IN ONLINE SOCIAL NETWORKS

Pili Hu & Prof. Wing Cheong Lau
The Chinese University of Hong Kong

Aug, 2014

black hat
USA 2014

OAuth App Impersonation Outline

- Short version
- Long version
  - OAuth Background
  - Previous Attacks Based on Misuse
  - App Impersonation Attack
    - Forged-implicit-grant-flow Attack
    - Forged-bearer-token Attack
    - Executive Summary
  - Case Study
    - Massive leakage of user data
    - Other sample exploits
  - Immediate Fixes & Reflections

Three System Participants in Online Social Network

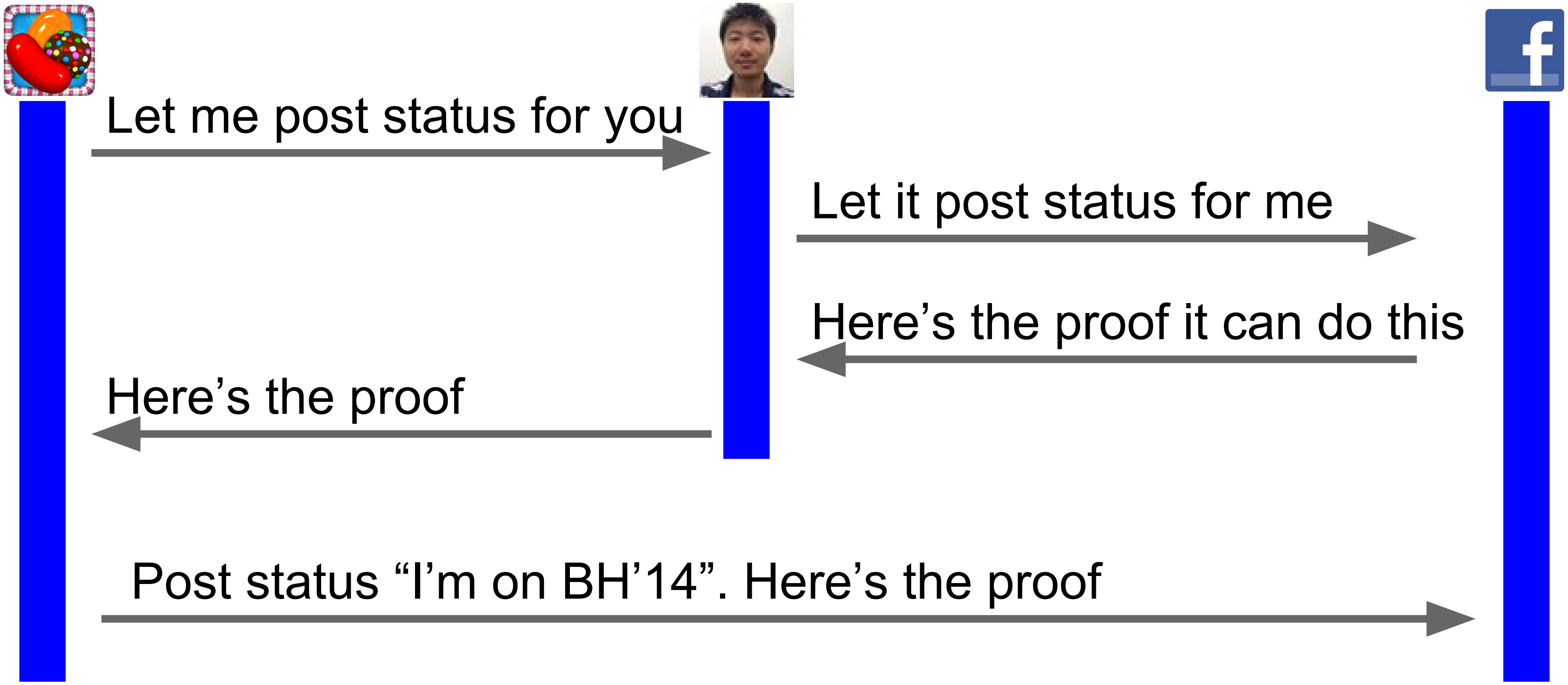
- Provider (e.g.)

  ……

- User (e.g.)
  - Register user account on Provider
  - Operate various data objects
- App (e.g.)
  - Register developer account on Provider
  - Get data objects access permission from
    - Provider: via application/ approval
    - User: via OAuth
  - AppID, **AppSecret**

[Short version]

Basic Interaction among App, User and Provider



Let me post status for you

Let it post status for me

Here’s the proof it can do this

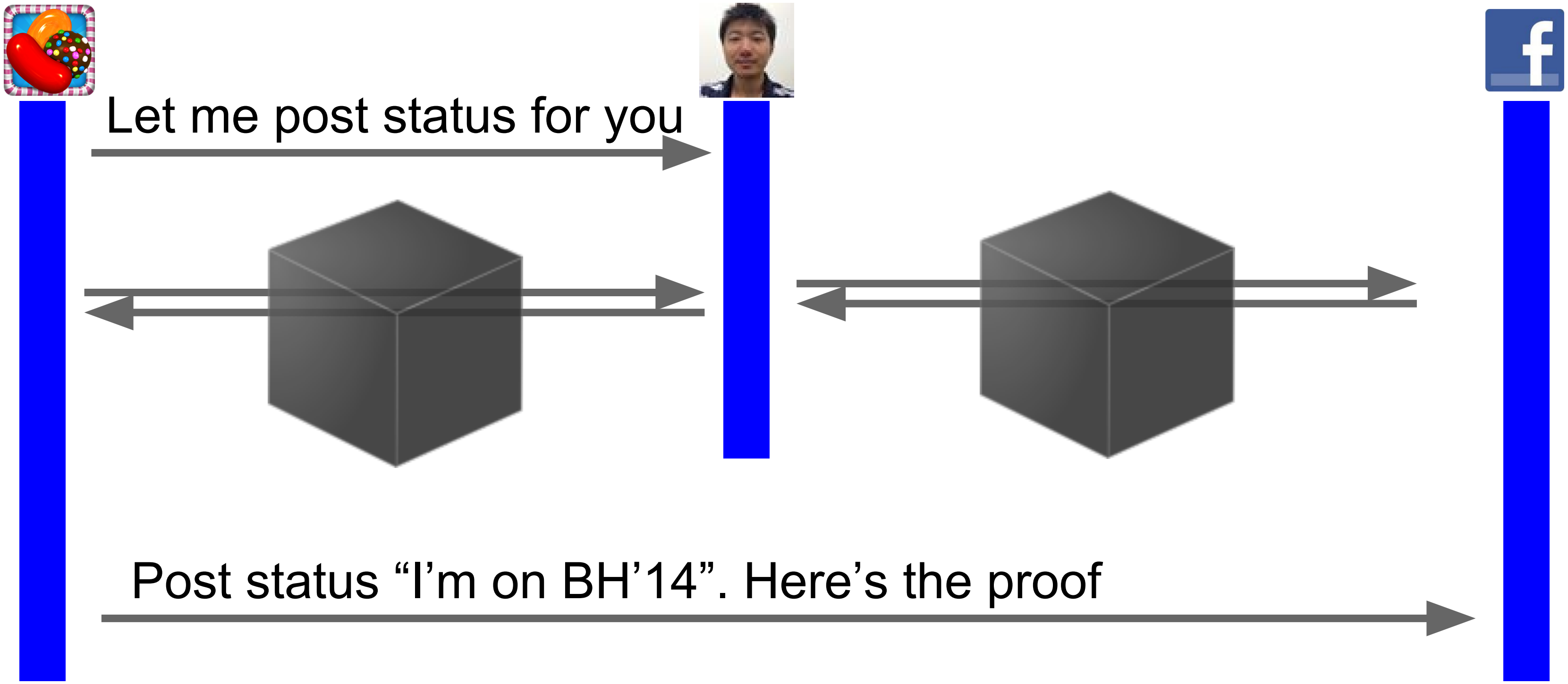
Here’s the proof

Post status “I’m on BH’14”. Here’s the proof

The proof is called “AccessToken” in OAuth

[Short version]

Basic Interaction among App, User and Provider


Let me post status for you
Post status “I’m on BH’14”. Here’s the proof

- The process can be more complex
- Ideally, App needs to prove to provider that it has AppSecret

[Short version]

App Impersonation Attack: Key Idea

Key idea:

- Get/ Use AccessToken without AppSecret
- AccessToken gives the privilege of "App+User" or "App"

How is this possible?

[Short version]

App Impersonation Attack: Made Possible by OAuth 2.0

OAuth 2.0 allows User to:

- Get AccessToken without AppSecret:
  ⇒ “Implicit grant flow”
- Use AccessToken without AppSecret:
  ⇒ “Bearer token”

How bad is it??

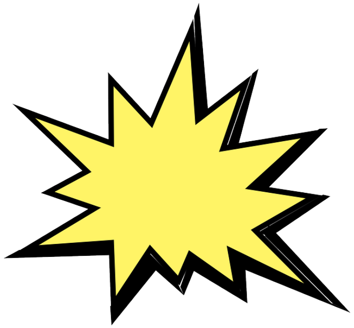
[Short version]

Consequences of App Impersonation Attack

Cause damage when not all Apps are equal:

- e.g. different access quota
- e.g. different access permissions

If User can impersonate a privileged App



[Short version]

Outline

- Short version
- Long version
  - OAuth Background
  - Previous Attacks Based on Misuse
  - App Impersonation Attack
    - Forged-implicit-grant-flow Attack
    - Forged-bearer-token Attack
    - Executive Summary
  - Case Study
    - Massive leakage of user data
    - Other sample exploits
  - Immediate Fixes & Reflections

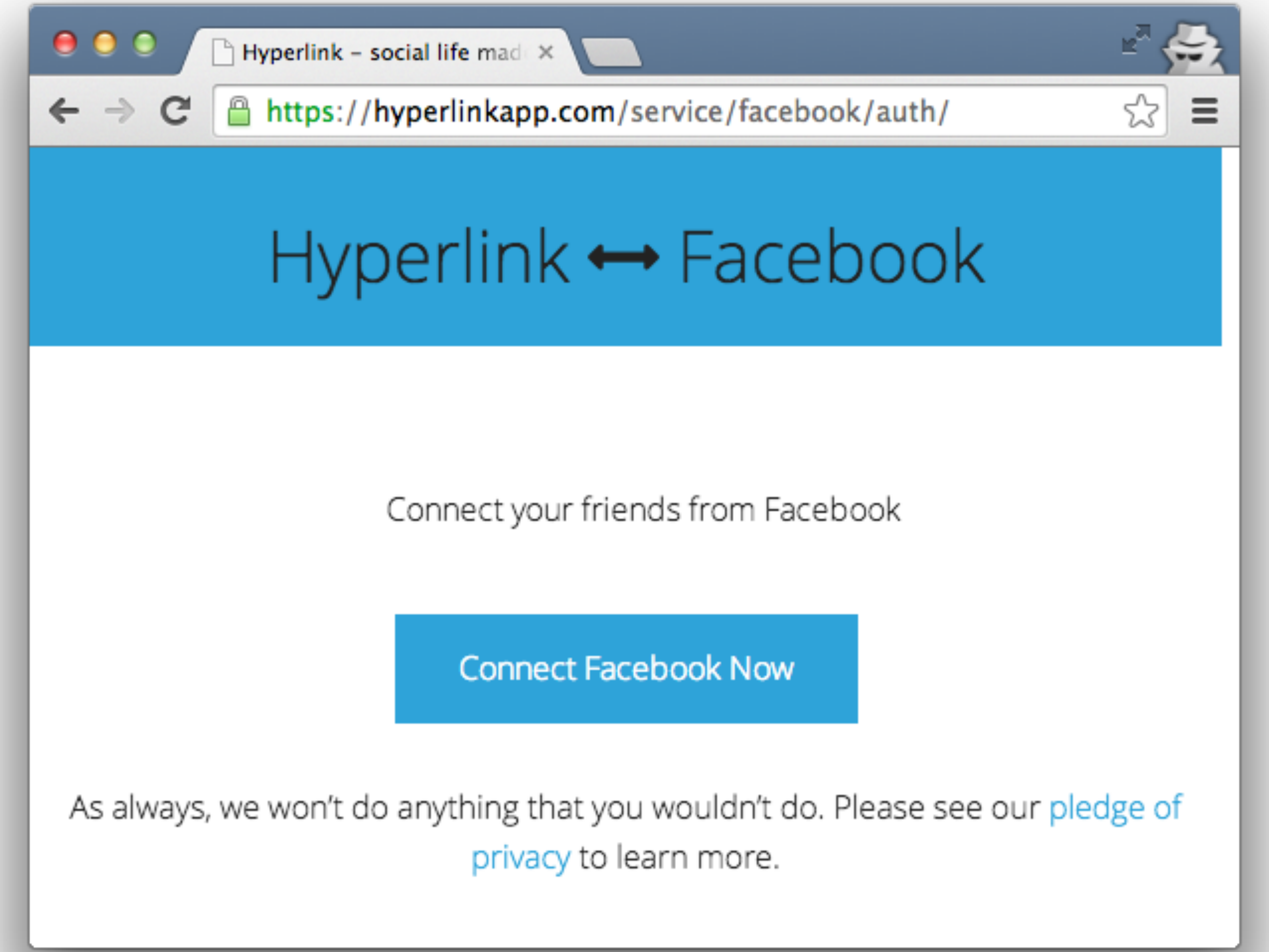
OAuth
Standardization & Landscape

- OAuth 1.0:
  - RFC5849, April 2010
  - Obsoleted by OAuth 2.0.
  - Only a few Provider, e.g. Twitter

OAuth
Standardization & Landscape

- OAuth 2.0:
  - Framework specification: RFC6749, Oct 2012
  - Security analysis: RFC6819, Jan 2013
  - Token types:
    - Bearer token: RFC6750
    - MAC token: E. Hammer-Lahav, draft-5 (Jan 2014)
  - Widely supported by Providers with different implementations

Authorization Code Flow Illustration



https://hyperlinkapp.com/service/facebook/auth/

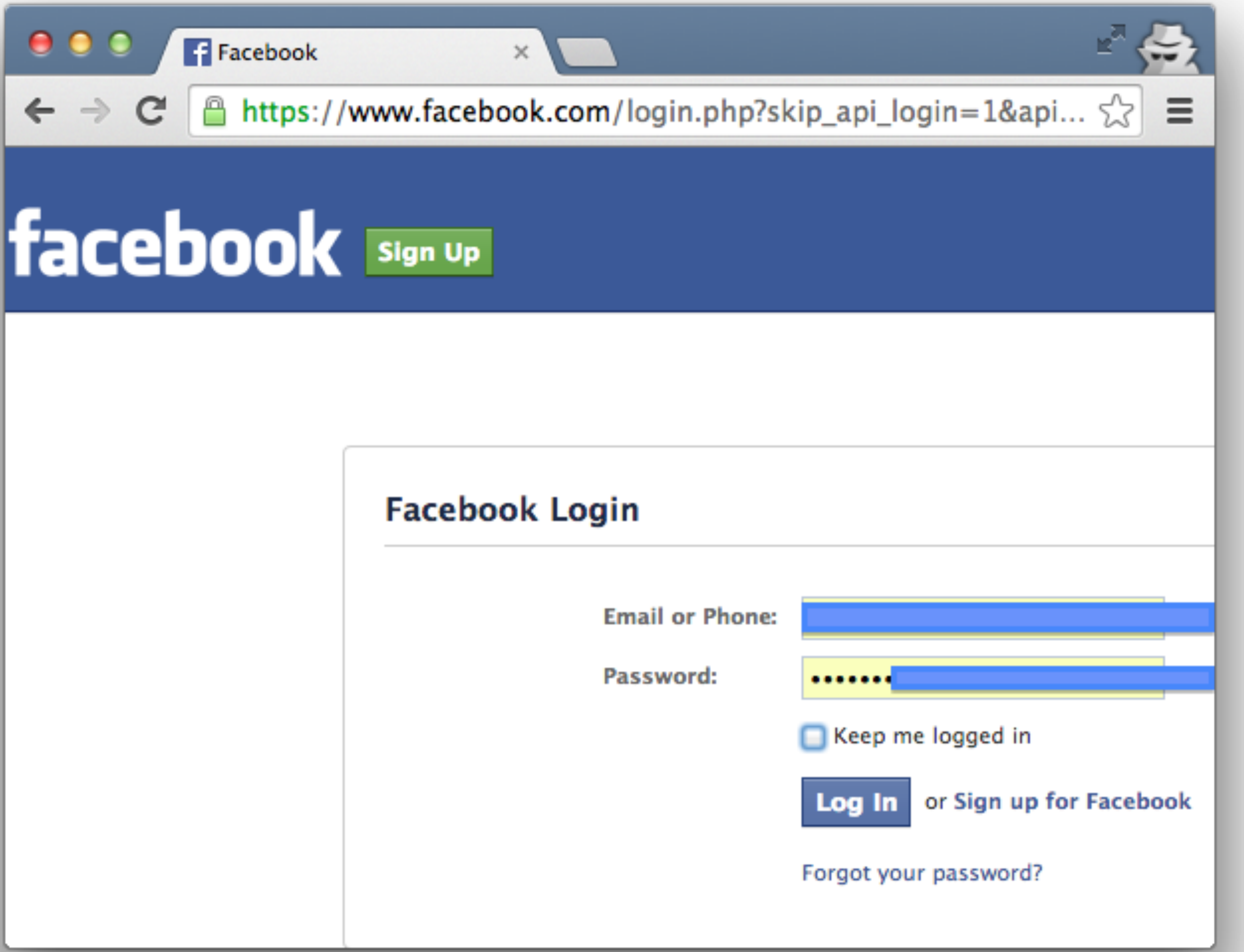
Hyperlink ⟷ Facebook

Connect your friends from Facebook

As always, we won't do anything that you wouldn't do. Please see our pledge of privacy to learn more.

1) Enter the App

Authorization Code Flow Illustration



Facebook

https://www.facebook.com/login.php?skip_api_login=1&api...

facebook Sign Up

Facebook Login

Email or Phone: [redacted]

Password: ••••••• [redacted]

□ Keep me logged in

Log In or Sign up for Facebook

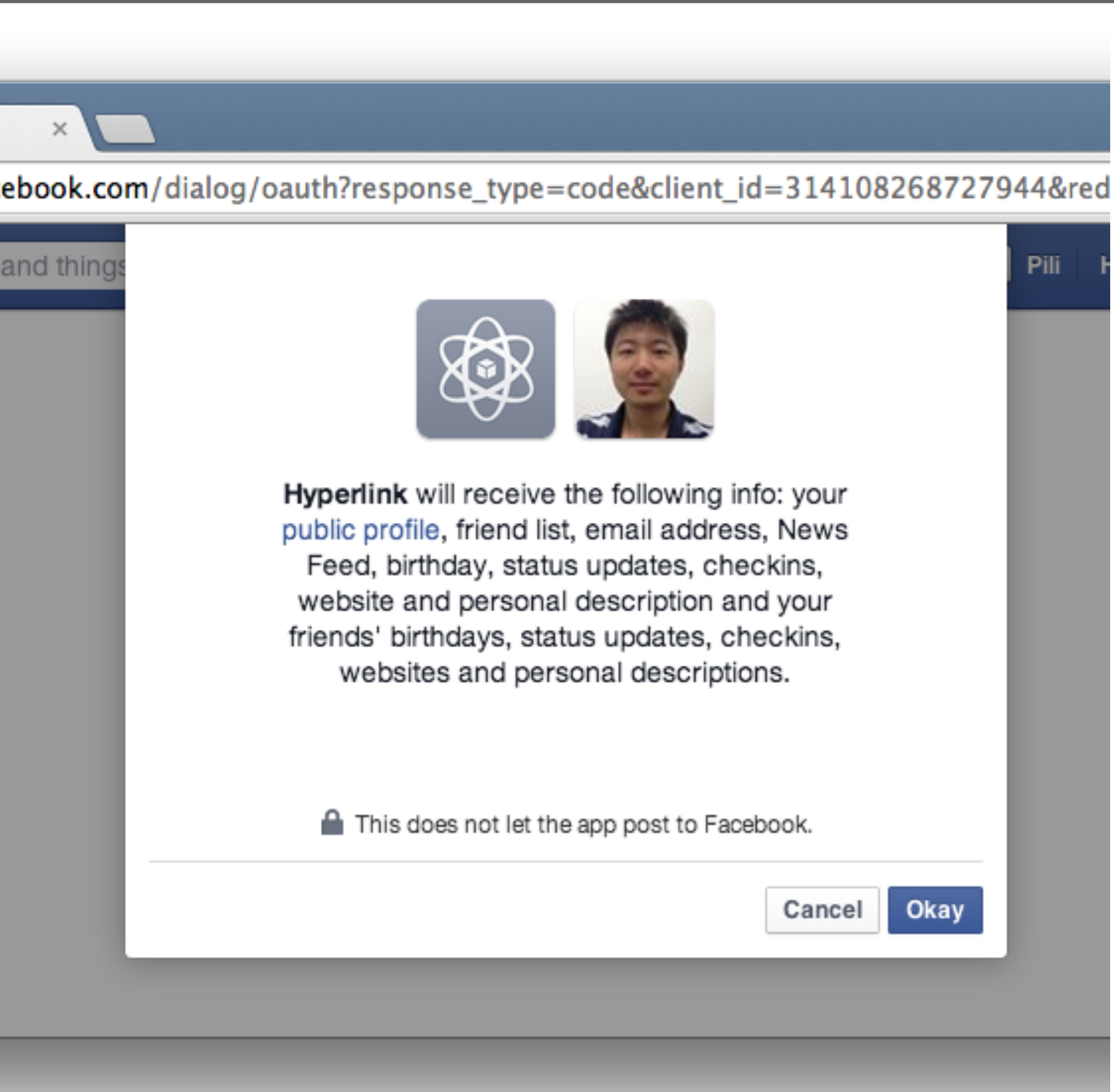
Forgot your password?

2) Redirect to provider

3.1) User authentication

(username + password)

Authorization Code Flow Illustration



ebook.com/dialog/oauth?response_type=code&client_id=314108268727944&red

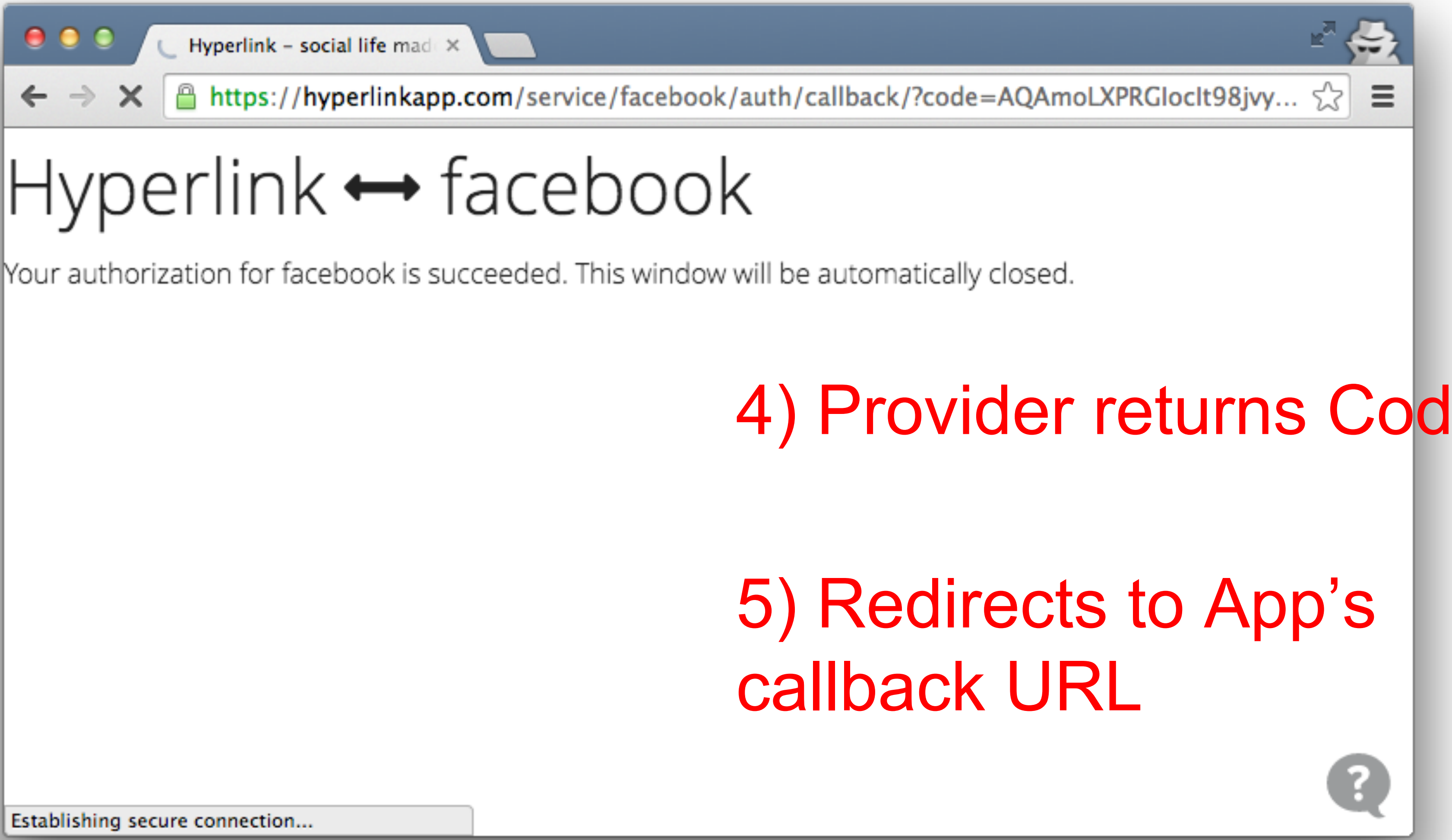
Hyperlink will receive the following info: your public profile, friend list, email address, News Feed, birthday, status updates, checkins, website and personal description and your friends' birthdays, status updates, checkins, websites and personal descriptions.

This does not let the app post to Facebook.

3.2) User authorization

(review scope and confirm)

Authorization Code Flow Illustration



Hyperlink – social life mad

https://hyperlinkapp.com/service/facebook/auth/callback/?code=AQAmoLXPRGlocIt98jvy...

Hyperlink ⟷ facebook

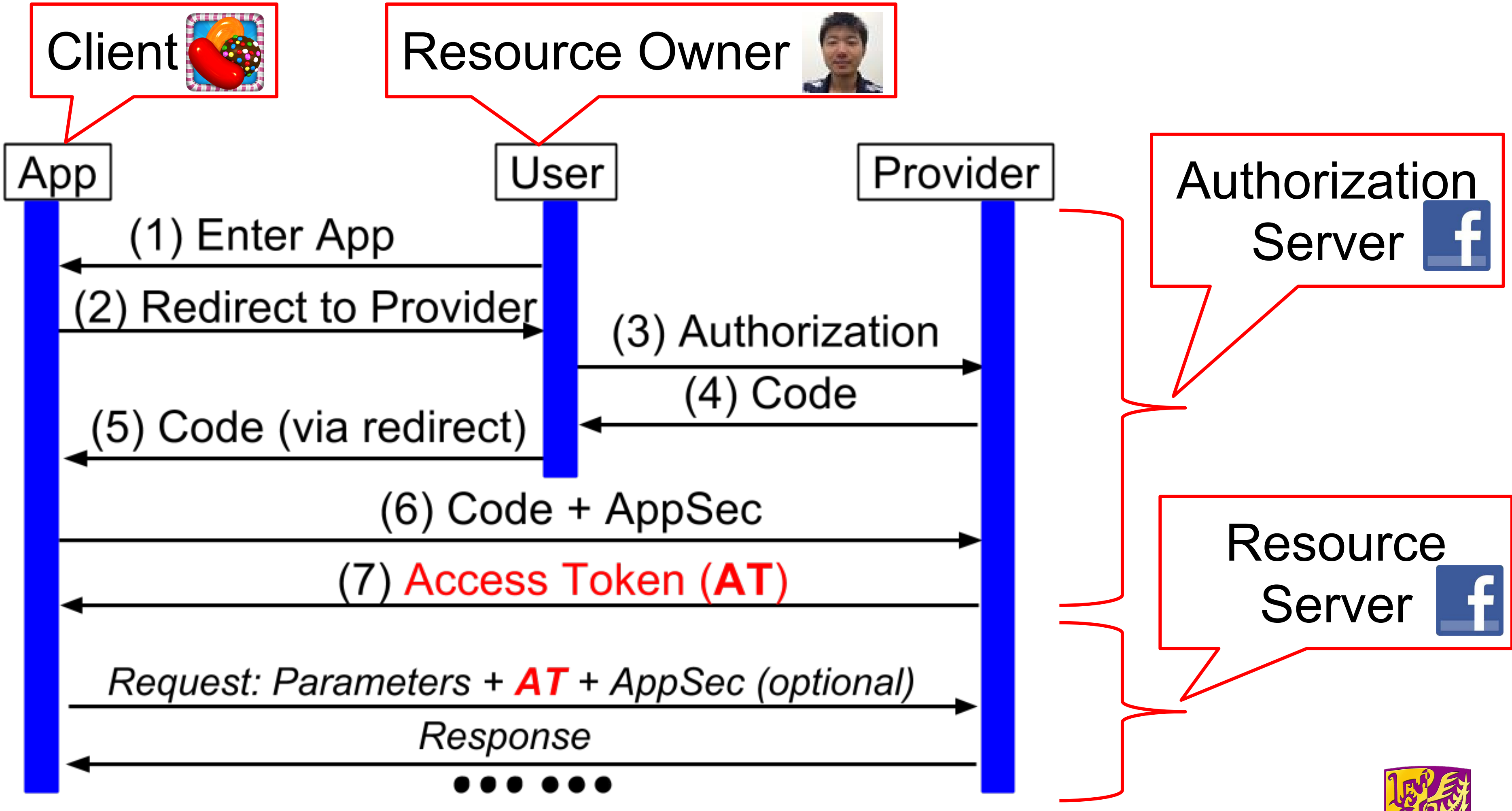
Your authorization for facebook is succeeded. This window will be automatically closed.

Establishing secure connection...

4) Provider returns Code

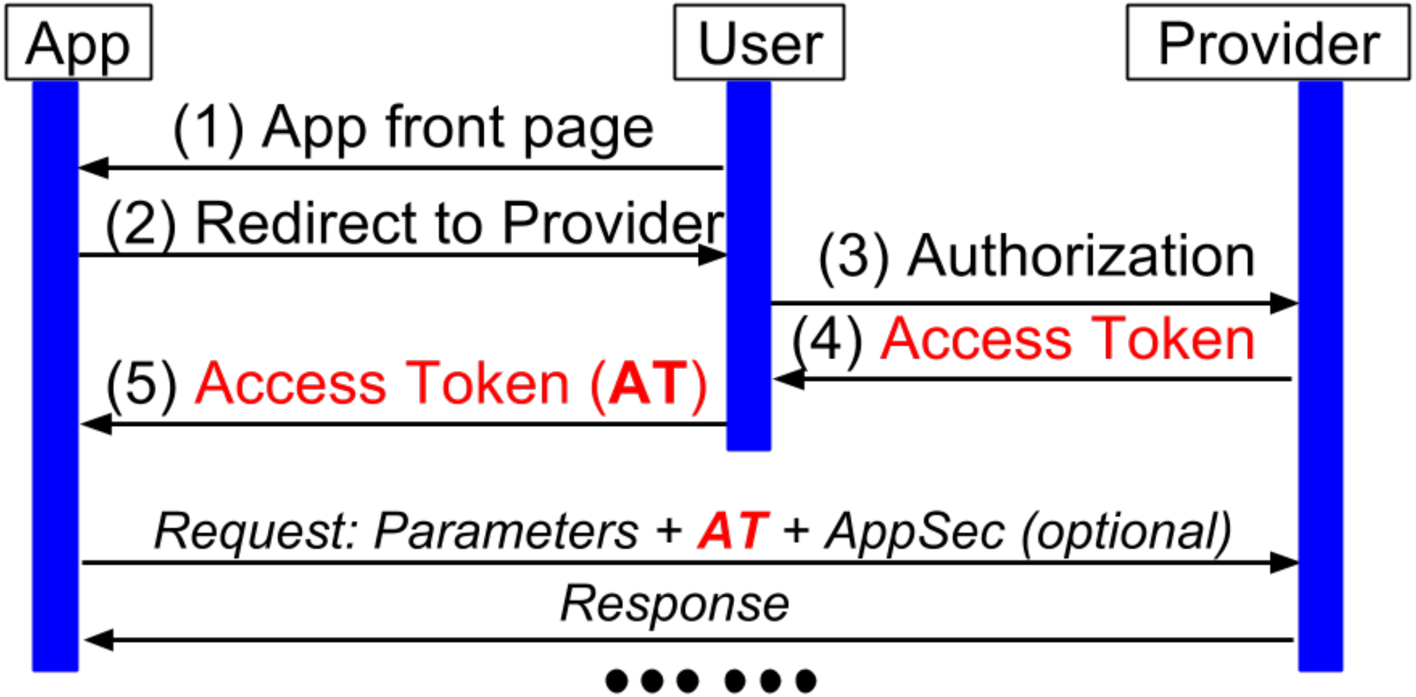
5) Redirects to App’s callback URL

OAuth Background
Authorization Code Grant



OAuth Background
Implicit Grant

Implicit Grant Flow

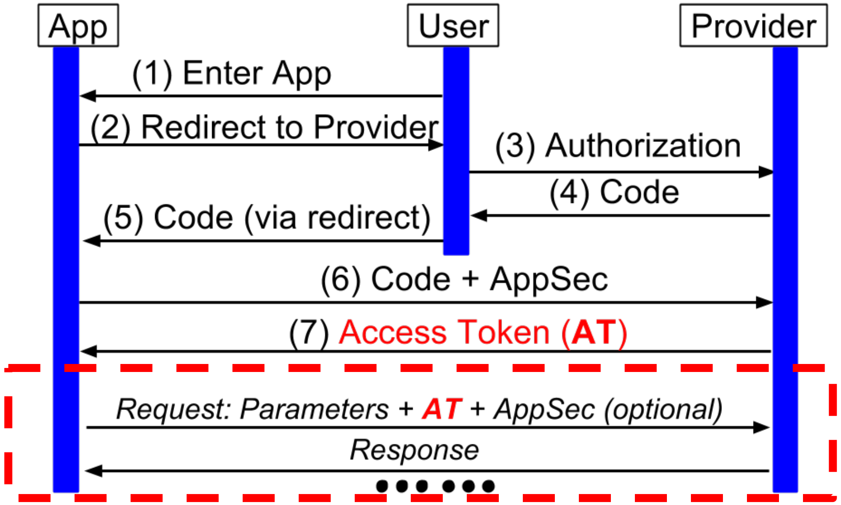


OAuth Background
Implicit Grant

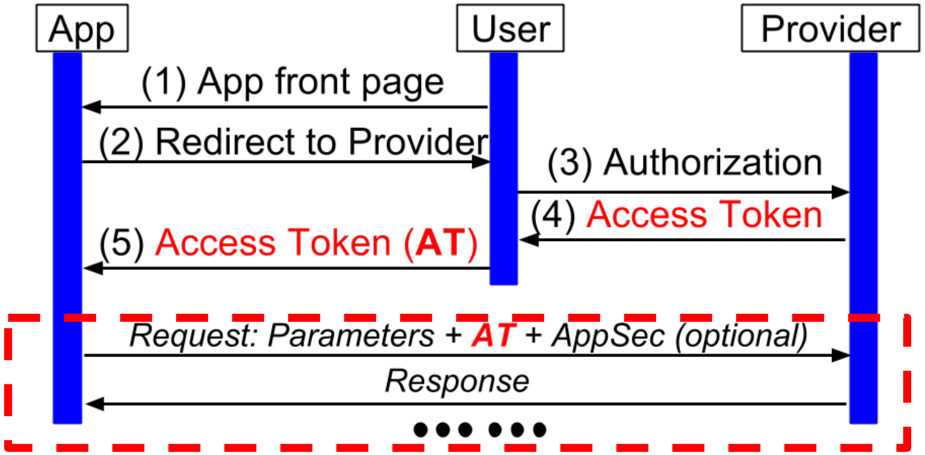
Properties of implicit grant flow:

- Access token is returned directly via User
- No AppSecret is used
- Originally introduced to ease developers
- Official usage:
  - Where resource is limited
  - Where App can not keep AppSecret anyway
  - Be avoided whenever authorization code grant is available

OAuth Background
How to use the Token?



Authorization code grant flow



Implicit grant flow

OAuth Background
Token Types

“Request: Parameters + AccessToken” means:

- Bearer token: Put the AccessToken in the request directly
- MAC token: Put the AccessToken and Parameters together and sign using AppSecret

OAuth Background
General Advice to Developers

General advice, now common knowledge for App developers:

- Use Authorization-code-grant flow if possible
- Use MAC token if possible

Previous Attacks on OAuth

Mainly based on misuse and other weak parts in Provider/App, e.g.:

- Session fixation: state is not used/checked
- Covert redirect: open redirector

General wisdom: Secure if all the guidelines are followed by Provider and App

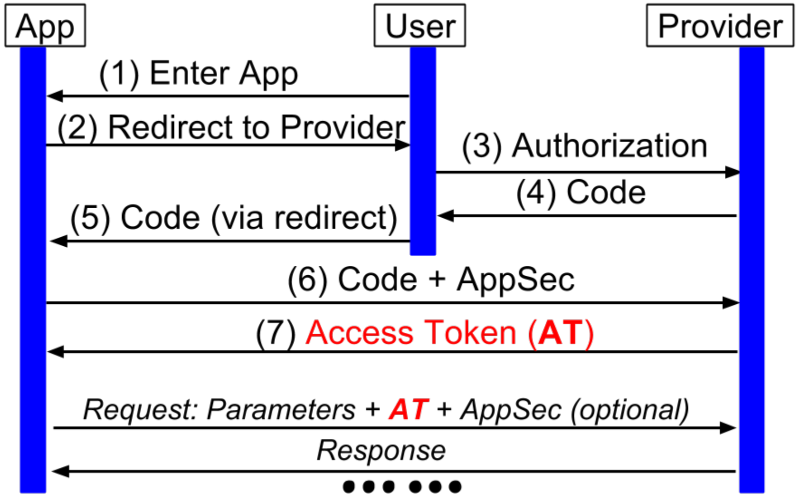
App Impersonation Attack

- Forged implicit grant flow attack
  - ⇒ Obtain AccessToken without AppSecret
- Forged bearer token attack
  - ⇒ Use AccessToken without AppSecret

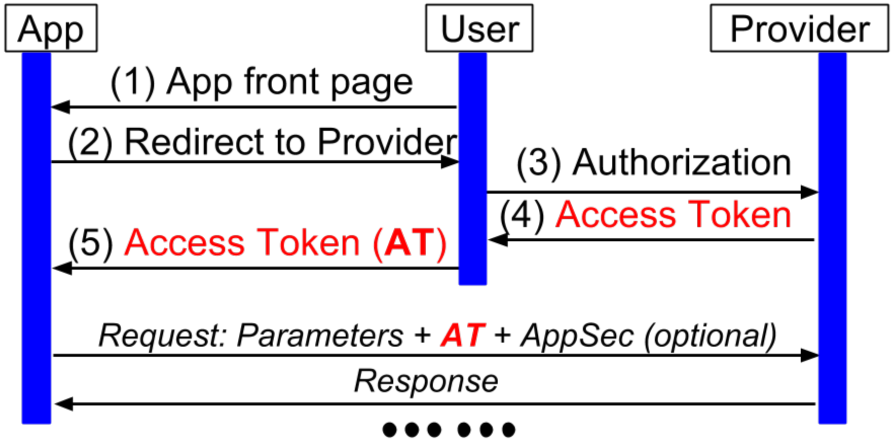
Without AppSecret ⇒ App Impersonation

Forged Implicit-Grant-Flow Attack

- Harvest `client_id` and `redirect_uri` from step (1)-(3) in authorization code grant
- Use the same parameters in implicit grant flow



Authorization code grant flow



Implicit grant flow

Forged Bearer Token Attack

- Put access token directly in:
  - HTTP request headers
  - URL parameters
  - POST fields

(RFC6750)

Bearer Token

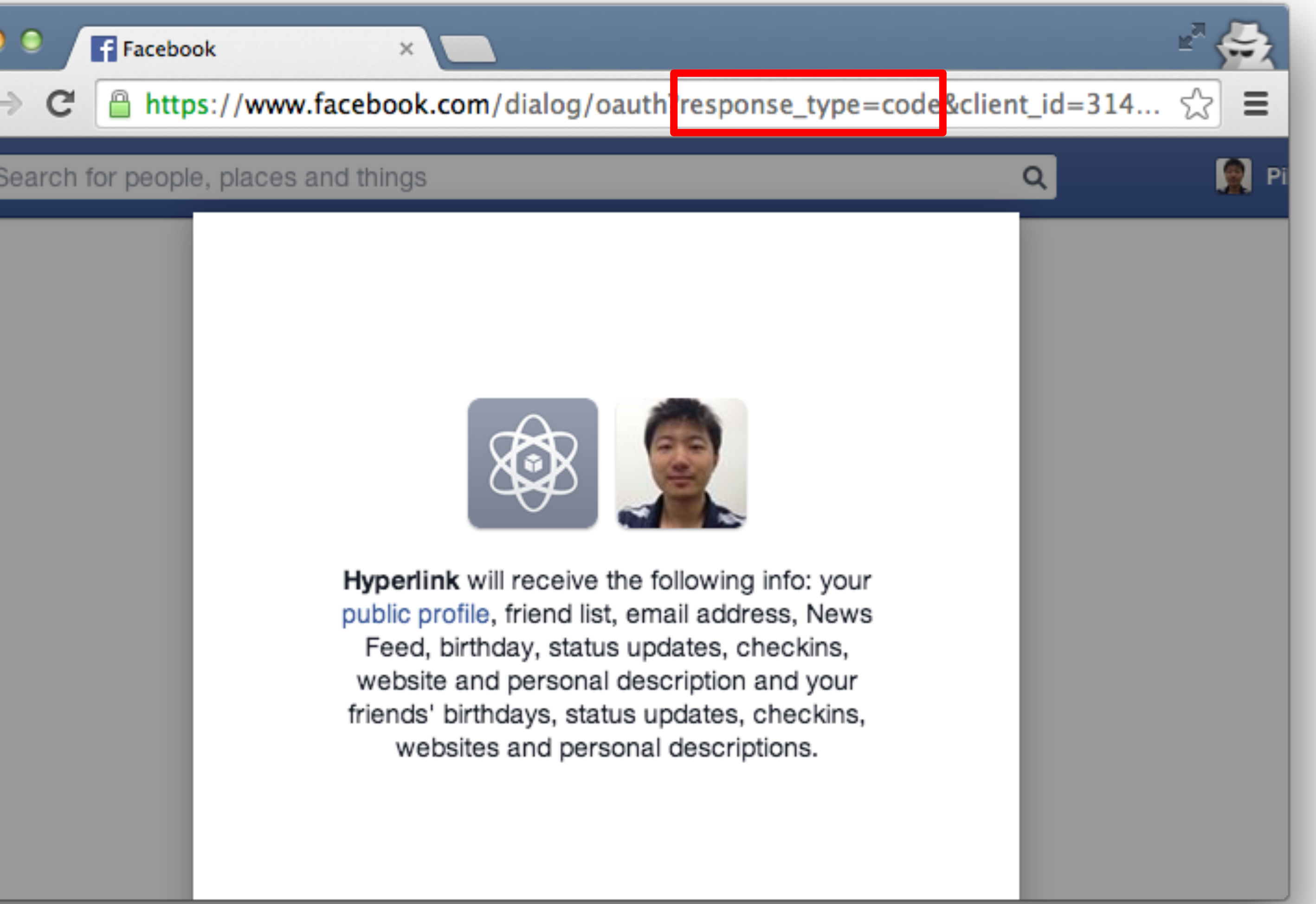
> A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material (proof-of-possession).

Forged Bearer Token Attack

- Token Type:
  - Most providers do not implement token_type
  - Most providers do not implement MAC token
  - Those who implement do not enforce a type
  - Those who implemented do not provide opt-outs

> (RFC6750)
>
> token_type
>
> REQUIRED. The type of the token issued as described in
> Section 7.1. Value is case insensitive.

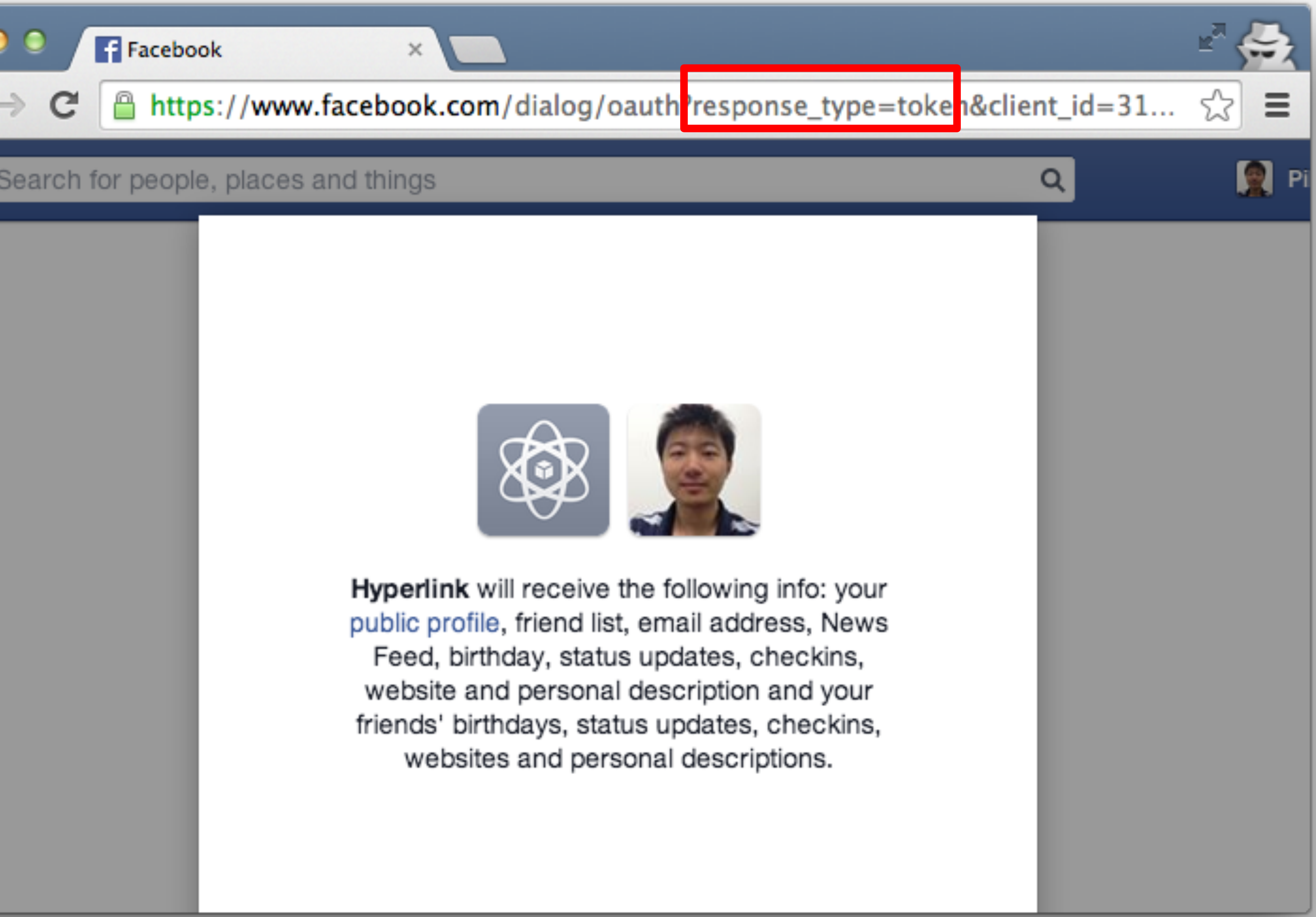
App Impersonation Attack Illustration


Facebook
https://www.facebook.com/dialog/oauth?response_type=code&client_id=314...
Search for people, places and things
Pi

Hyperlink will receive the following info: your public profile, friend list, email address, News Feed, birthday, status updates, checkins, website and personal description and your friends' birthdays, status updates, checkins, websites and personal descriptions.

Go to normal authorization page

App Impersonation Attack Illustration

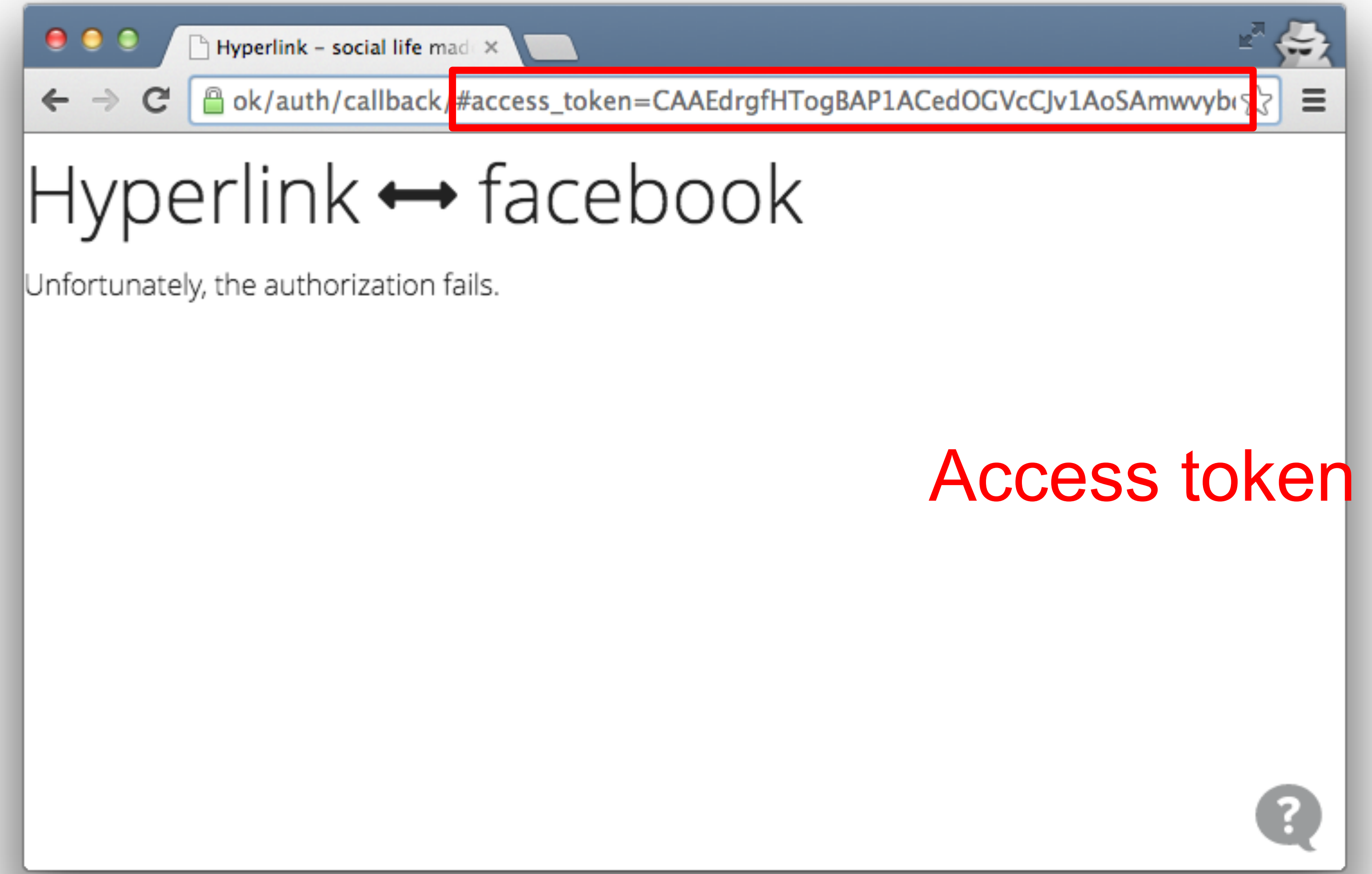


https://www.facebook.com/dialog/oauth?response_type=token&client_id=31...

Hyperlink will receive the following info: your public profile, friend list, email address, News Feed, birthday, status updates, checkins, website and personal description and your friends' birthdays, status updates, checkins, websites and personal descriptions.

Change respose_type to “token”

App Impersonation Attack Illustration



Hyperlink – social life mad ×

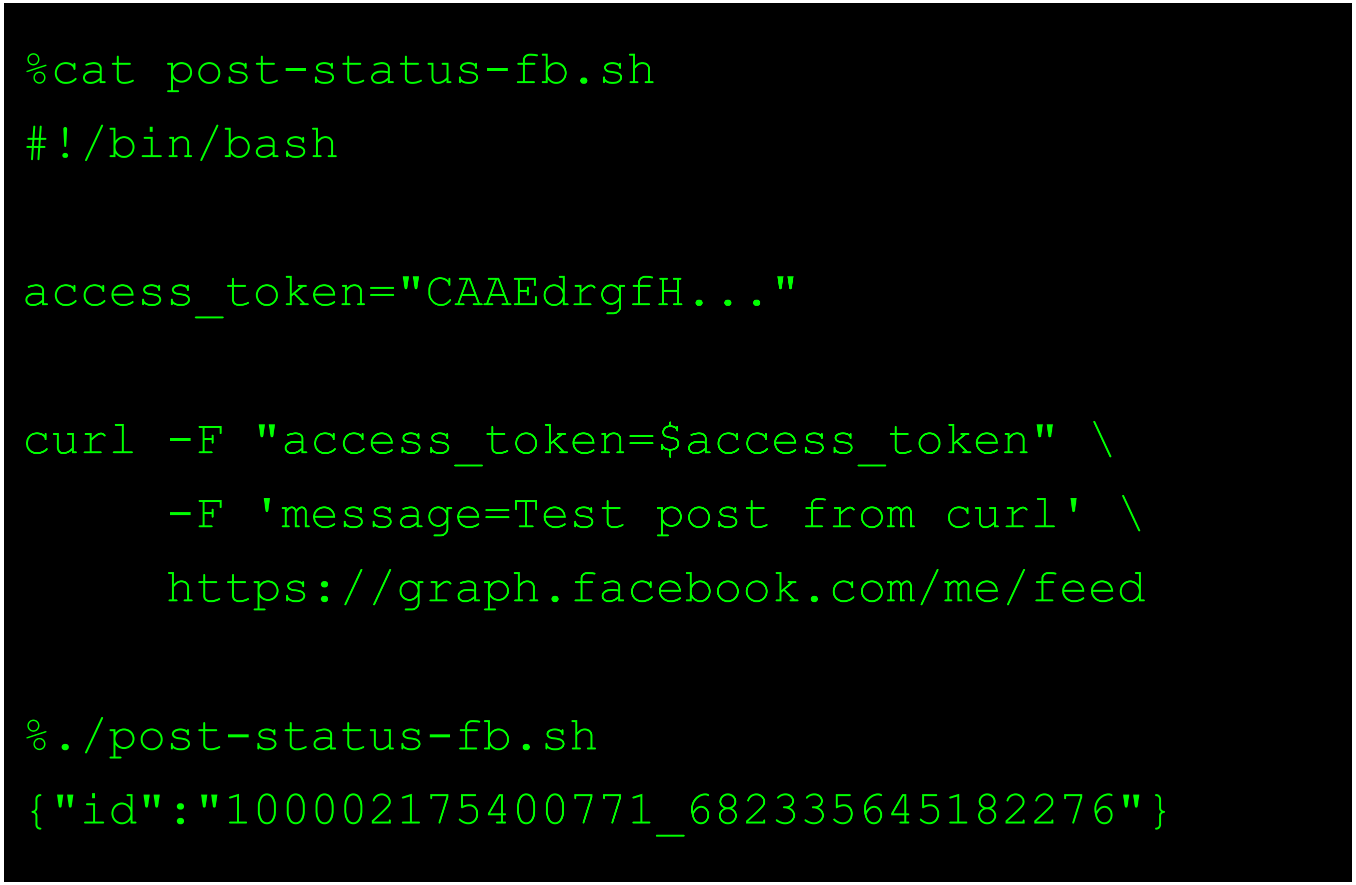
ok/auth/callback/#access_token=CAAEdrgfHTogBAP1ACedOGVcCJv1AoSAmwvyb

Hyperlink ⟷ facebook

Unfortunately, the authorization fails.

Access token obtained!

App Impersonation Attack Illustration



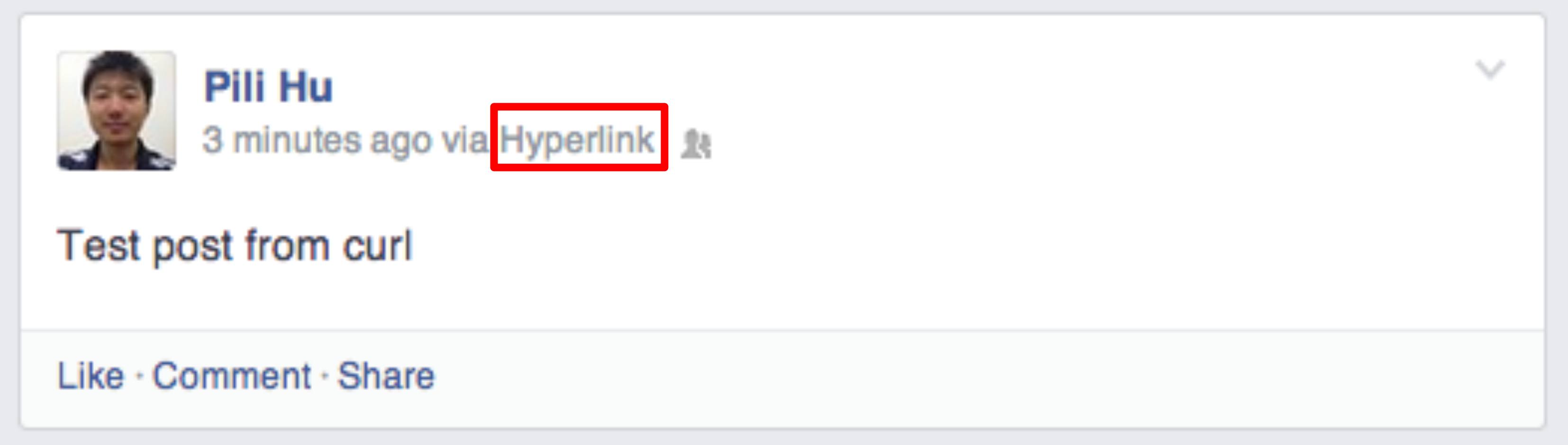
```
%cat post-status-fb.sh
#!/bin/bash

access_token="CAAEdrgfH..."

curl -F "access_token=$access_token" \
     -F 'message=Test post from curl' \
     https://graph.facebook.com/me/feed

%./post-status-fb.sh
{"id":"100002175400771_682335645182276"}
```

Resource request

Can be done fully in browser if the endpoint uses GET method.

Or with the help of some brower extensions/ developer tools.

App Impersonation Attack Illustration



Pili Hu

3 minutes ago via Hyperlink

Test post from curl

App Impersonation Attack Executive Summary

```
/authorize?response
type=code&client_id=XXXX&state=XXXX&redirect_uri=XXXX
```

```
/authorize?response
type=token&client_id=XXXX&state=XXXX&redirect_uri=XXXX
```

```
/api?access_token=XXX&other_parameters
```

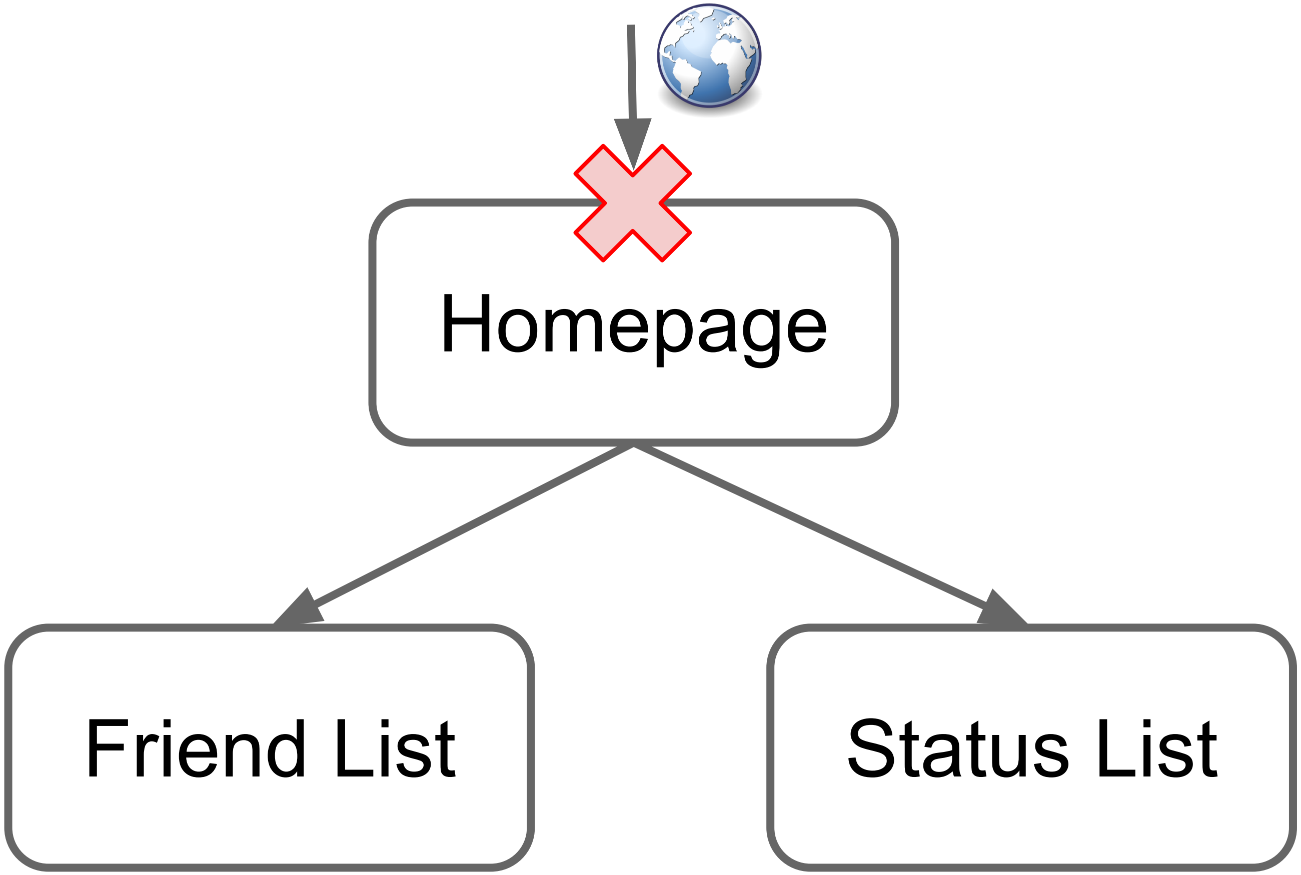
Case Study of Provider X

Big Deal?

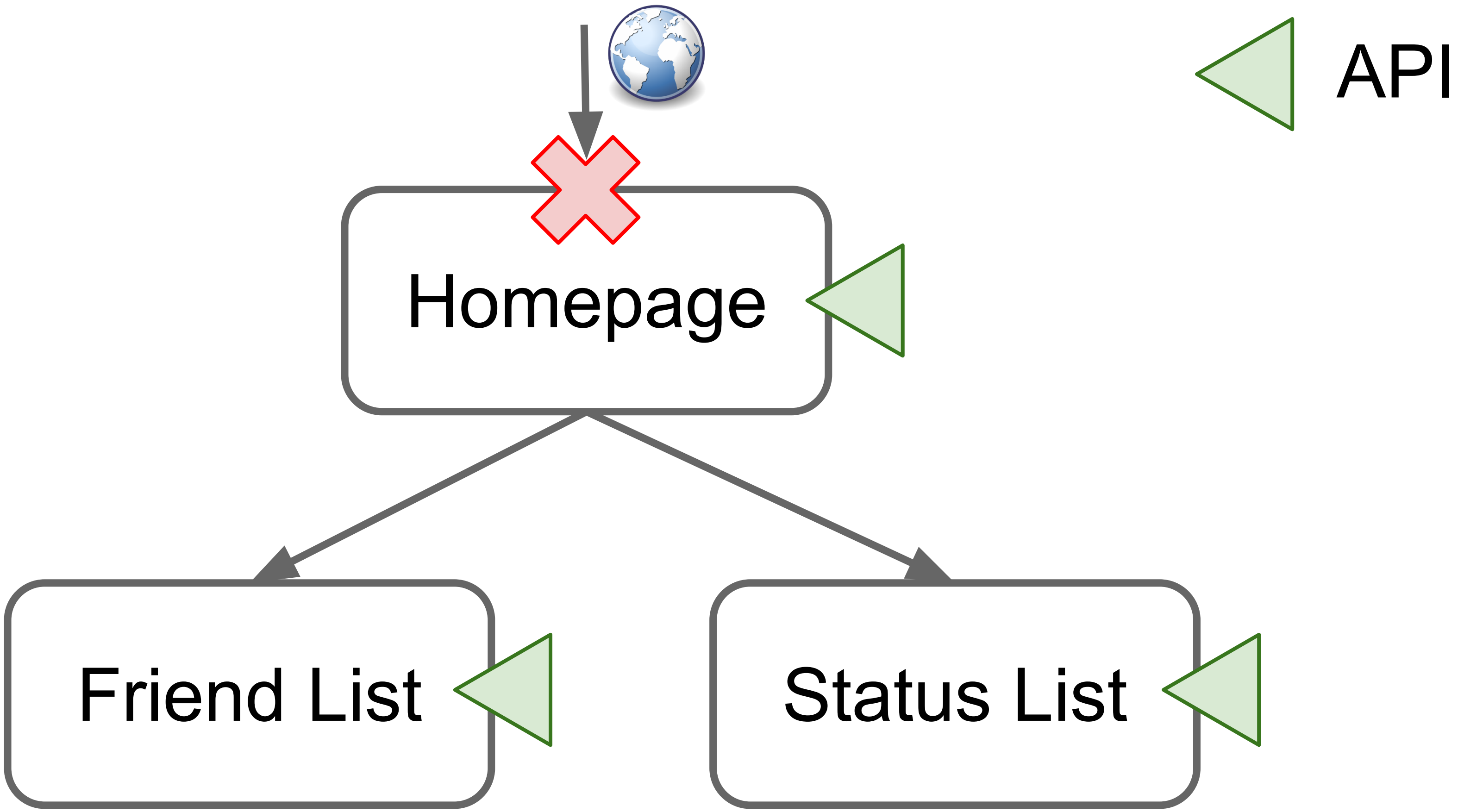
Case Study of Provider X

Provider X: A Facebook-like (not Facebook) OSN with >100 million users

Case Study of Provider X
Basic Setup and User Perception



Case Study of Provider X
API Access and Problematic Scopes



“read_status” v.s.
“read_self_status”/ “read_friend_status”/ “read_other_status”

Case Study of Provider X
API Access Permissions

Feedback of the inconsistency:

- Provider X: by design (June, 2013)
- Users: surprised to know; unaware of it
  - Interview with real users
  - Quantitative study on 4400 users

Case Study of Provider X Rate Control

Apps are differentiated on Provider X:

- Normal App: 200 Queries/hour
- Some higher level App: 900 Queries/hour

⇒ Takes years to collect the data even if it’s “public”

Case Study of Provider X
Rate Control

We find at least one Privileged App:
> 1 million queries/hour

100 million users / 1 (million/hour) = 100 hours

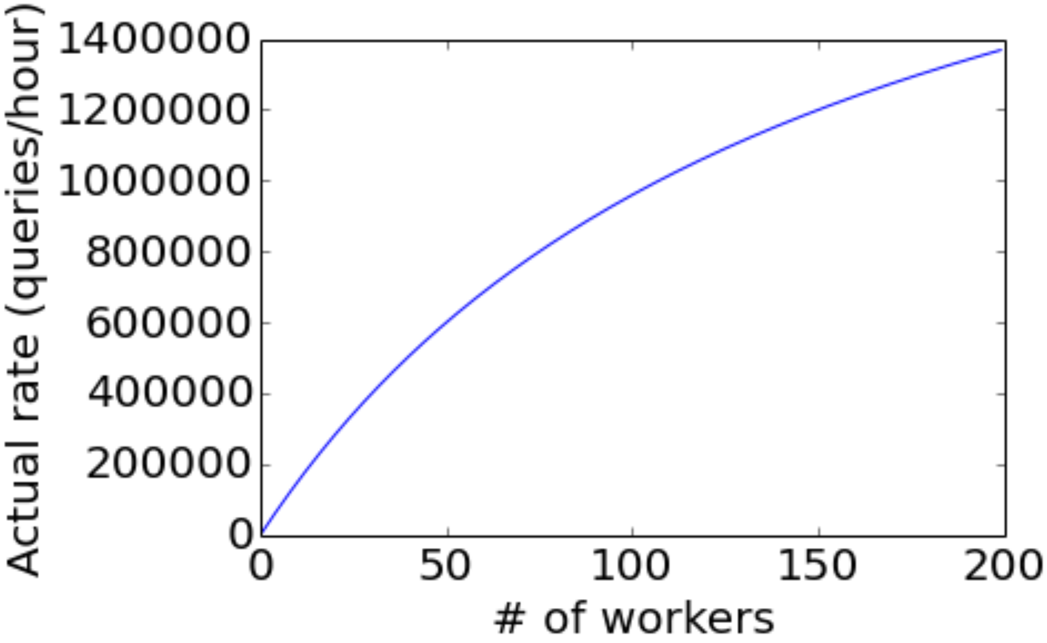
Cost: < US$ 100
(AWS EC2 m3.2xlarge for 100 hours)

Case Study of Provider X
Estimate Achievable Rate

Model: $r = c * w / (w + b)$

- r: observed rate
- c: capacity
- w: # of work processes
- b: background rate (from other Apps)

Case Study of Provider X Estimate Achievable Rate



w1=50,r1=600K (Q/hour)

w2=100,r2=960K (Q/hour)

⇒ c=2.4M, b=150

How to leak 100 million private user data in one week?

- OAuth App Impersonation
- Privileged App that possesses large quota
  - 1 million quries/hour
- Problematic design of scope
  - “read_status” == “read_everyone’s_status”
- Inconsistent access control misperceived by users
  - Provider: public data
  - User: private data

Other Sample Exploits

- Send notifications with embedded URLs to all users of the App
- Acquire access privileges that are otherwise unavailable for normal App
- App reputation Attack, e.g. “posted via XXX”
- and more ...

Refer to our upcoming paper in ACM COSN’14 for details

Immediate Fixes

- Opt-out/ opt-in for implicit grant flow
- Opt-out/ opt-in for bearer token type
- Review “scope” design
- Review rate control mechanism
- Review privileged Apps

Reflections

- OAuth 2.0 has diverse implementations that differ from specification
- New attacking surface: App Impersonation
- App Impersonation combined with other flaws can result in serious exploits
- Protecting App is a MUST when designing the next generation of the OAuth protocol

Thanks & Q/A

OAuth App Impersonation Attack

Project Page:

http://mobitec.ie.cuhk.edu.hk/oauth/

Pili Hu

hupili.net

Wing Cheong Lau

www.ie.cuhk.edu.hk/~wclau/