black hat EUROPE 2020
DECEMBER 9–10
BRIEFINGS

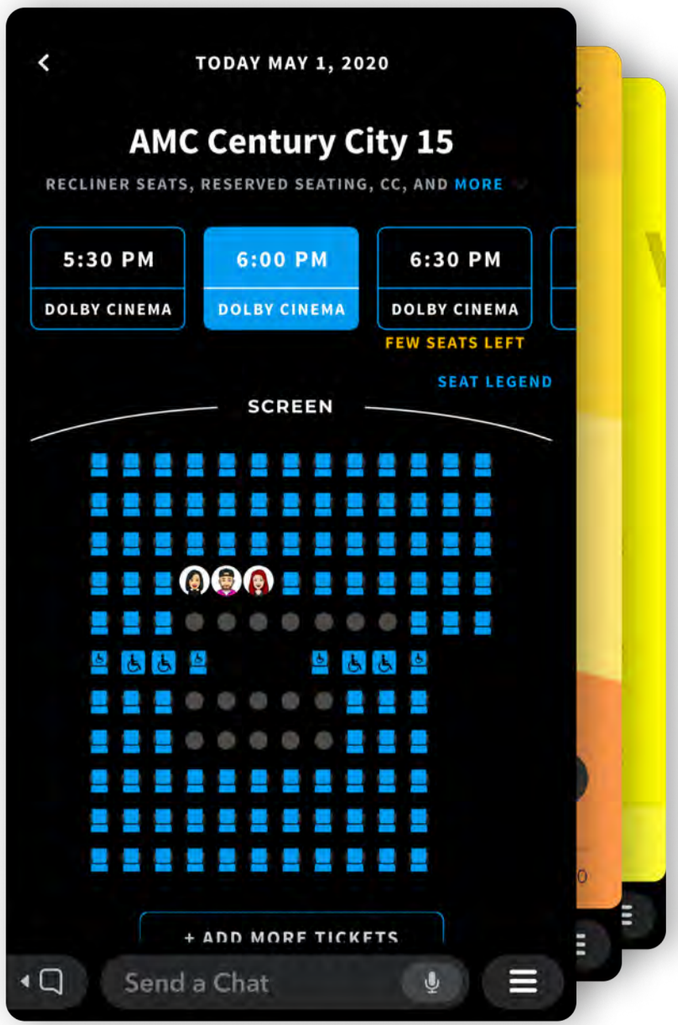
Design Pitfalls in Commercial Mini-Programs on Android and iOS

Haoran Lu, Luyi Xing, Xiaojing Liao

Indiana University Bloomington

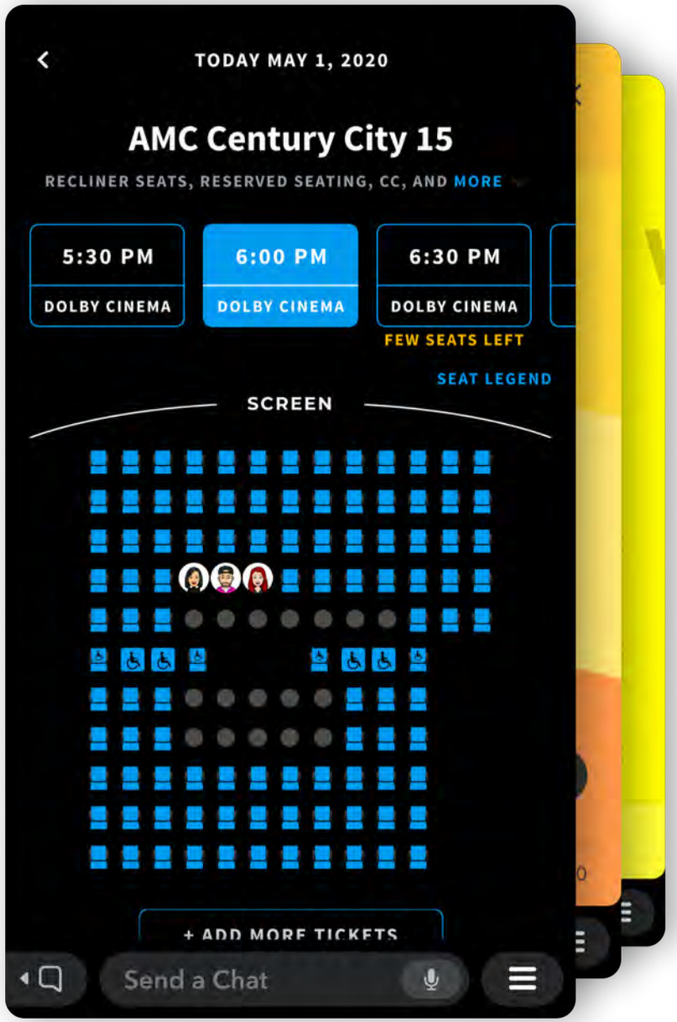
The App-in-app Paradigm

- A.k.a., mini-program
- Host app
  - A mobile app (on Android/iOS) that can run many mini-programs (or called sub-apps) in it
  - e.g., Snapchat, Wechat, Facebook
- Sub-app (Mini-program)
  - Run within the host app
  - Native-app like user experience
  - Enrich host app functionalities
  - Increase user stickiness
  - e.g., Amazon, Tesla, McDonald's, Walmart



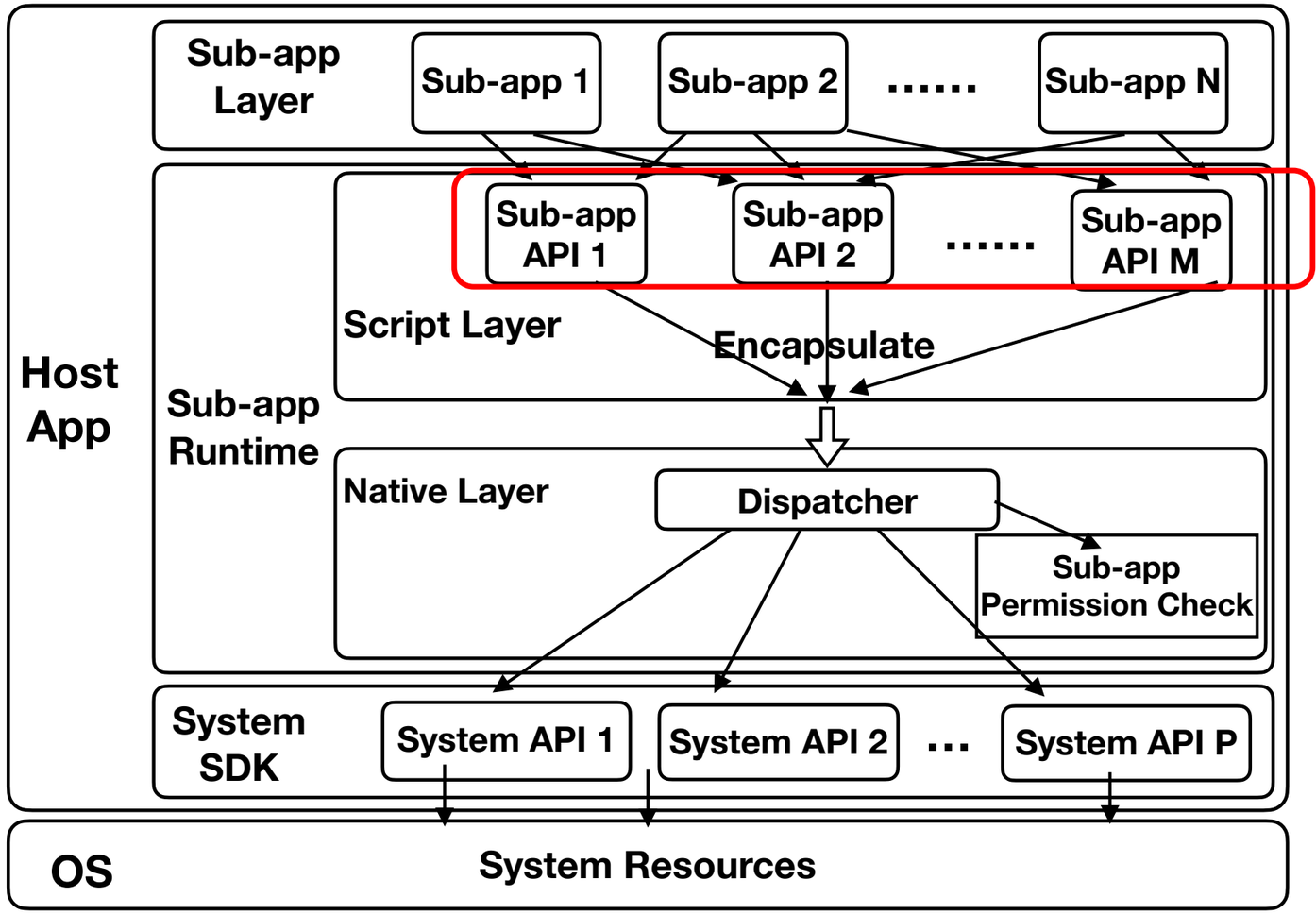
App-in-app popularity

- We studied 11 hosts, with more being released
  - WeChat, TikTok, Facebook, Snapchat, iMessage, Kodi, Alipay, etc.
  - 2.6B+ downloads
- 1,000,000+ sub-apps (mini-programs)
  - HSBC
  - Amazon
  - Microsoft Office 365
  - Airbnb, Expedia
  - Starbucks, McDonald’s
  - Health
  - …



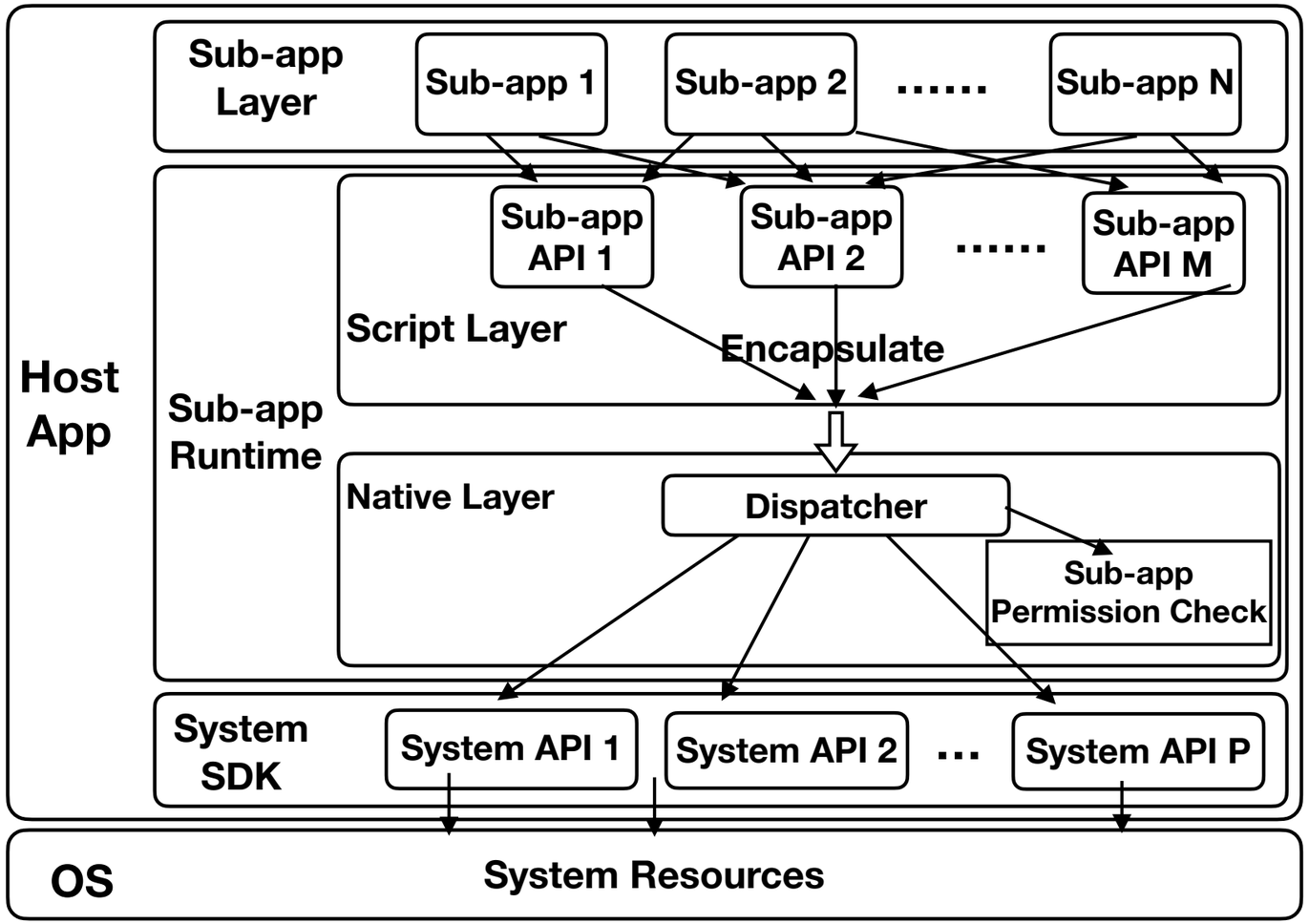
App-in-app Architecture

- Host app
  - Acts like an OS
  - Delegate system resources
- Sub-app
  - Call sub-app APIs
  - Run inside the host app



Security Model

- Sub-app permission
  - protect sensitive resources
  - scope.record – wx.startRecord
- Isolation
  - unique ID
  - unique storage
- Sub-app vetting



New Security Challenges

- It is fundamentally hard for a third-party app—the host app—to properly manage sub-apps.
  - Cannot reuse mobile OSes mechanisms/policies
    - x Isolation
    - x Permission policies
    - x UI model
    - x Lifecycle management

- Lack of standardization
  - Different policies for same resources

Security Weaknesses and Attacks

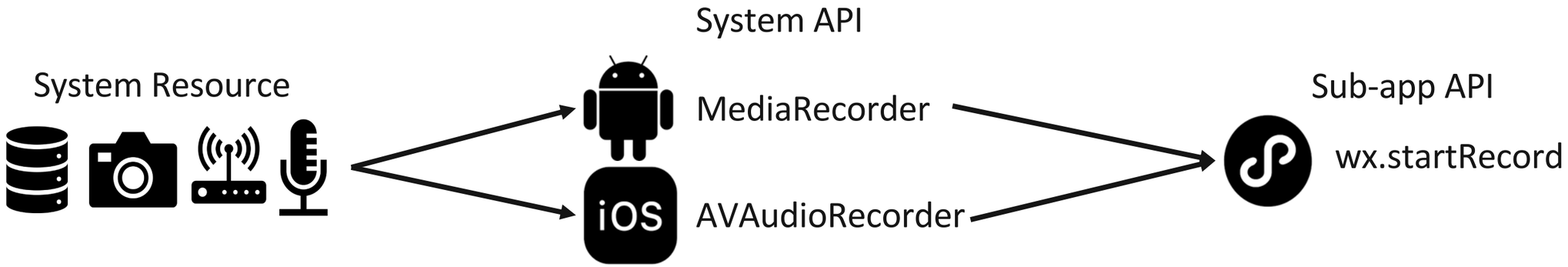
- System Resource Exposure
  - Weakness in System Resource Management
- New Overlay Hazard
  - Weakness in Access Control Management
- Sub-window Deception
  - Weakness in UI Management
- Sub-app Lifecycle Hijacking
  - Weakness in lifecycle management

Security Weaknesses and Attacks

- **System Resource Exposure**
  - **Weakness in System Resource Management**
- New Overlay Hazard
  - Weakness in Access Control Management
- Sub-window Deception
  - Weakness in UI Management
- Sub-app Lifecycle Hijacking
  - Weakness in lifecycle management

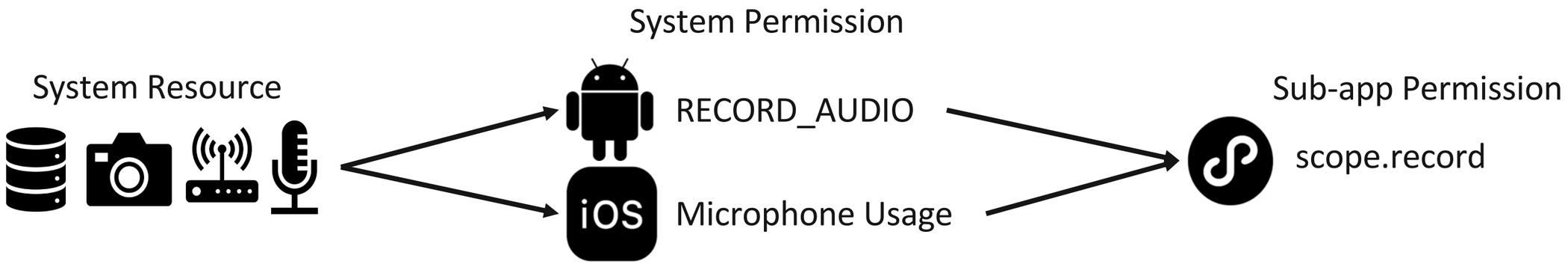
System Resource Exposure

- Expected permission requirement between sub-app API and system API



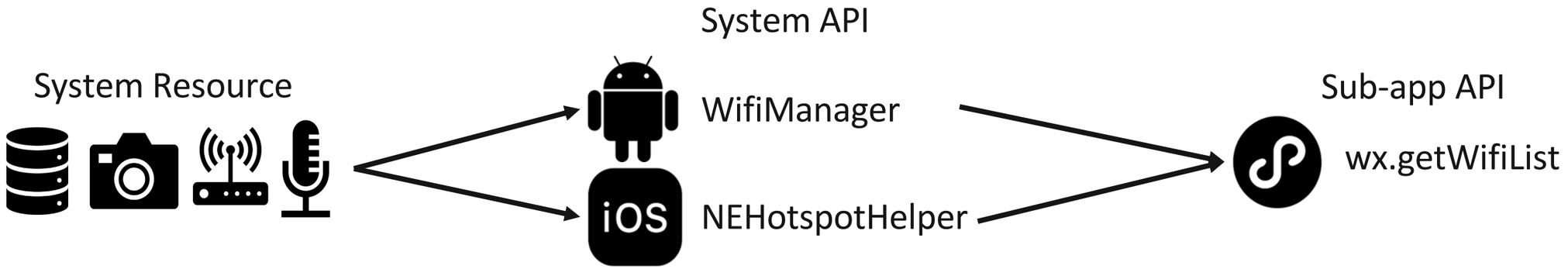
System Resource Exposure

- Expected permission requirement between sub-app API and system API



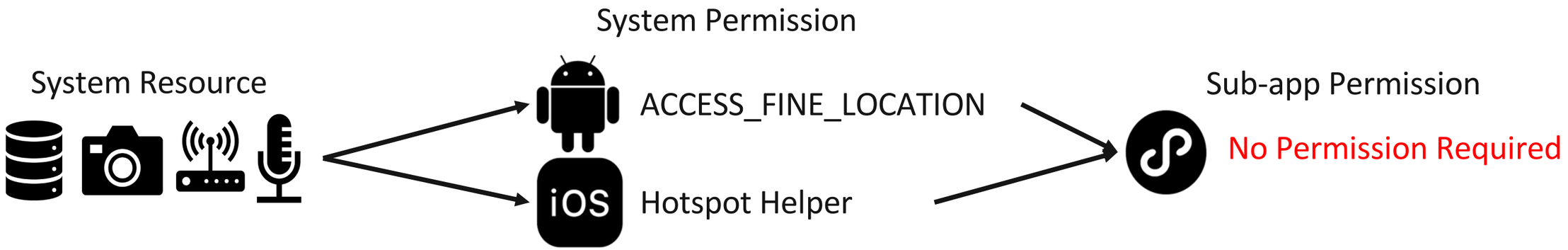
System Resource Exposure

- Inconsistent permission requirement between sub-app API and system API



System Resource Exposure

- Inconsistent permission requirement between sub-app API and system API
  - ***Escaped Sub-app API***



Possible Root Causes

- Unclear OS-level security policies
- Cross-platform discrepancy

	System Resource	Require Permission ?
Android	Bluetooth Scan	✓
iOS	Bluetooth Scan	✗

Possible Root Causes

- Unclear OS-level security policies
  - Opaque
  - scattered
  - Unsystematized

System Permission

iOS Hotspot Helper → What is protected? Location leak?

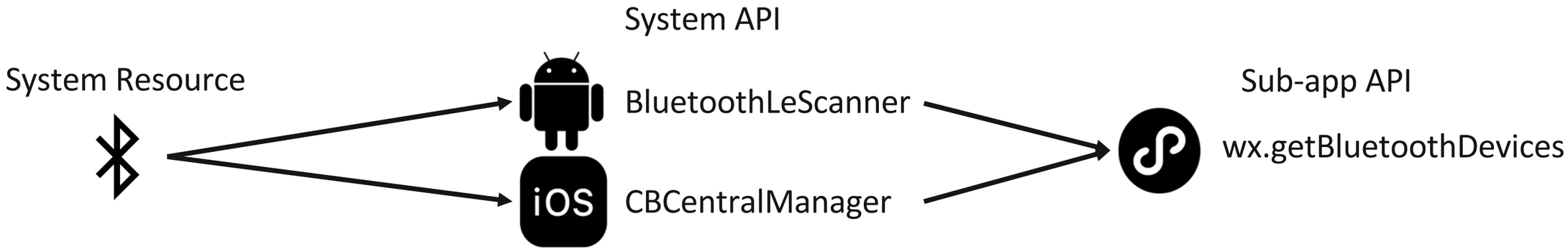
Possible Root Causes

- Unclear OS-level security policies
  - Opaque
  - scattered
  - Unsystematized

System API	System Permission
WifiManager	?

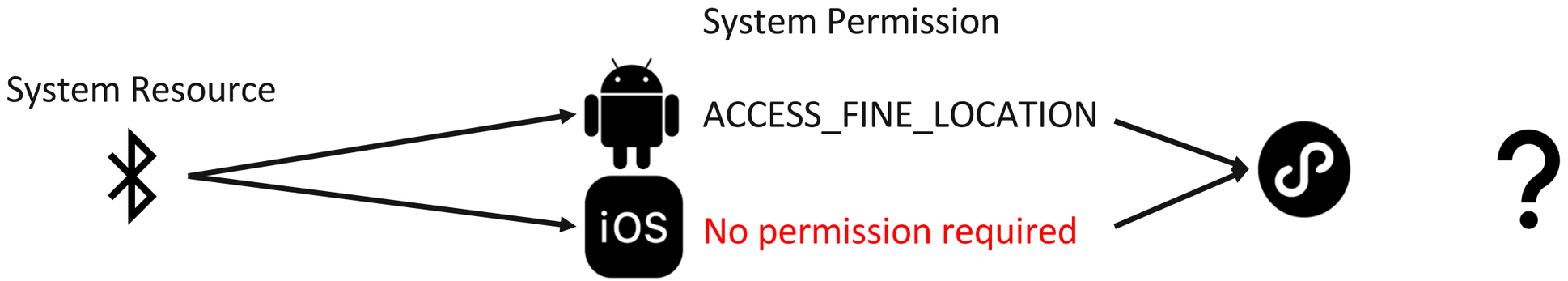
Possible Root Causes

- Cross-platform discrepancy
  - app-in-app are cross-platform
  - resource protection inconsistency between Android and iOS



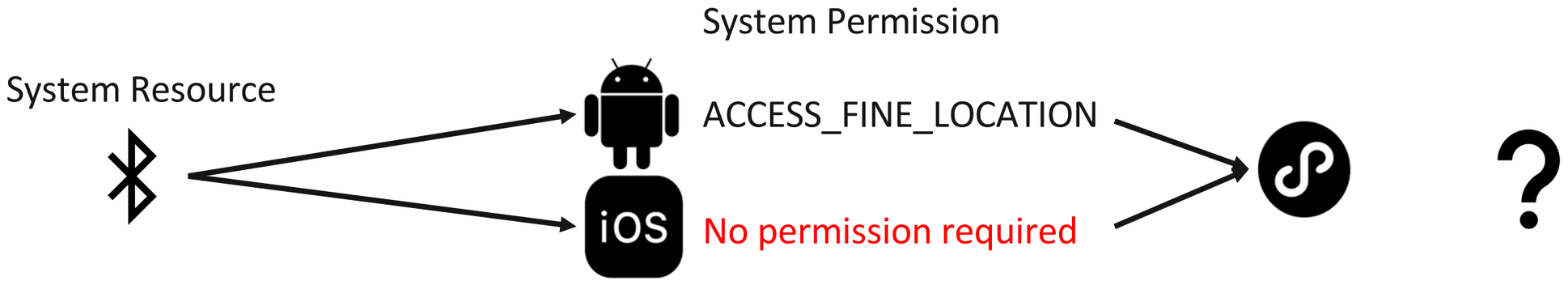
Possible Root Causes

- Cross-platform discrepancy
  - app-in-app are cross-platform
  - resource protection inconsistency between Android and iOS



Possible Root Causes

- Cross-platform discrepancy
  - app-in-app are cross-platform
  - resource protection inconsistency between Android and iOS



Results

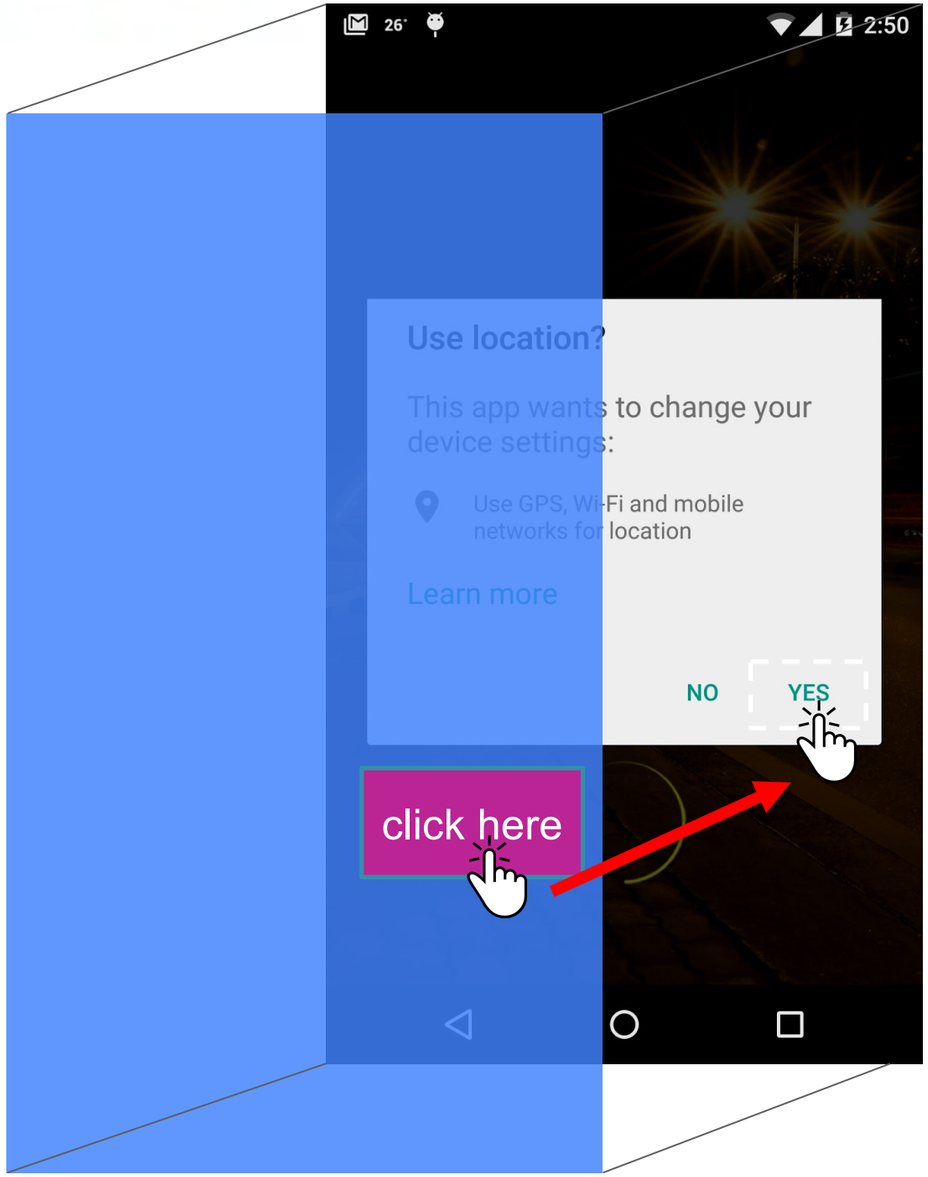
- 11 Host Apps
- 39 Escaped Sub-app APIs
- 5 exposed System Resources
- 6 affected host apps

Security Weaknesses and Attacks

- System Resource Exposure
  - Weakness in System Resource Management
- **New Overlay Hazard**
  - **Weakness in Access Control Management**
- Sub-window Deception
  - Weakness in UI Management
- Sub-app Lifecycle Hijacking
  - Weakness in lifecycle management

New Overlay Hazard

- Previous UI redressing / clickjacking attack on Android
- Defense: *Hide Overlays*
  - HIDE_NON_SYSTEM_OVERLAY_WINDOWS



New Overlay Hazard

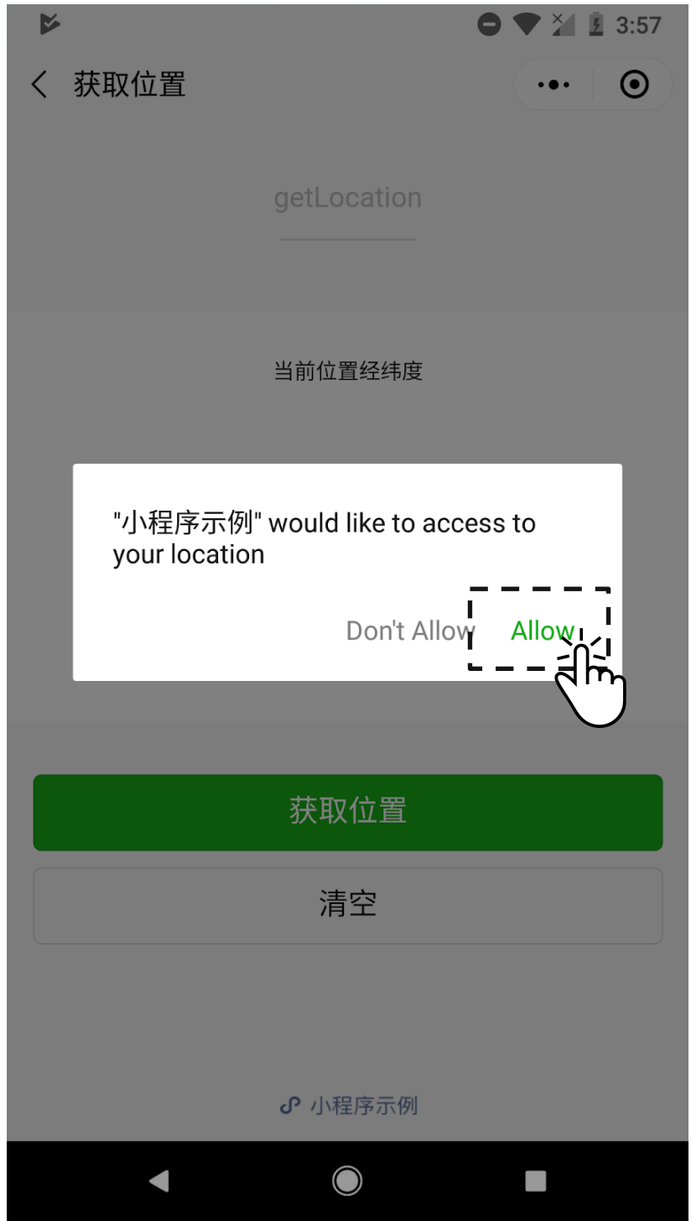
- HIDE_NON_SYSTEM_OVERLAY_WINDOWS
  - Not available to the host apps

```
<permission android:name="android.permission.HIDE_NON_SYSTEM_OVERLAY_WINDOWS"
            android:protectionLevel="signature|installer" />
```

https://android.googlesource.com/platform/frameworks/base/+/88bc1ce35fc6b2fa58d7a1b321ce209d2f5ef83c/core/res/AndroidManifest.xml

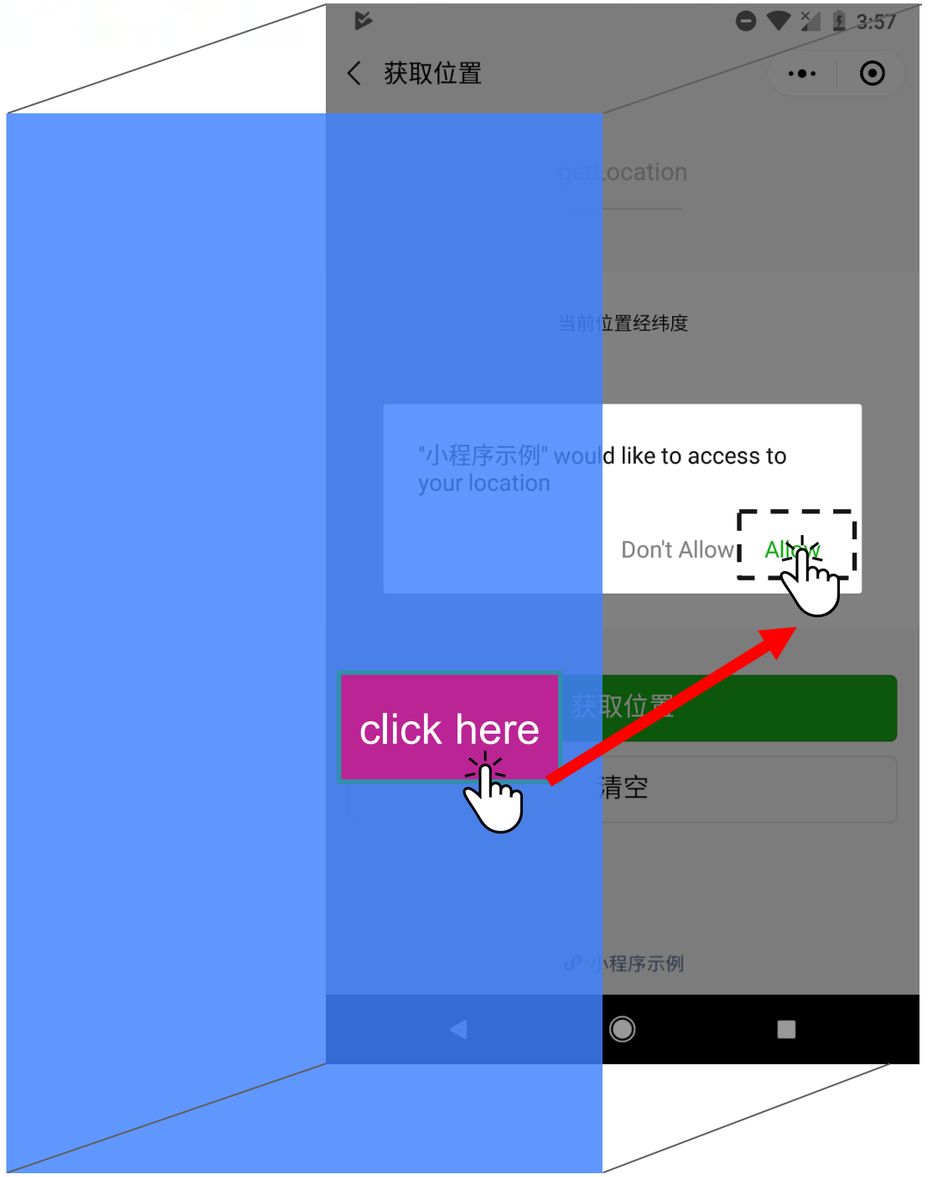
New Overlay Hazard

- Host Apps are powerful
  - Hold lots of permissions
  - Delegation through Sub-app APIs
- *Sub-app permission pop-ups are not protected*



New Overlay Hazard

- Malicious Android app
  - cover the screen, launch sub-apps
  - draw overlays over host-apps' permission granting window
- Colluding Sub-app
  - invoke sensitive sub-app APIs
- channels for coordination
  - clipboard
  - WebSocket
  - …



Results

- 10 Host Apps available on Android
- All 10 are vulnerable
- Exposed System Resources
  - Camera, Microphone
  - External storage
  - Location
  - Contacts
  - …

Security Weaknesses and Attacks

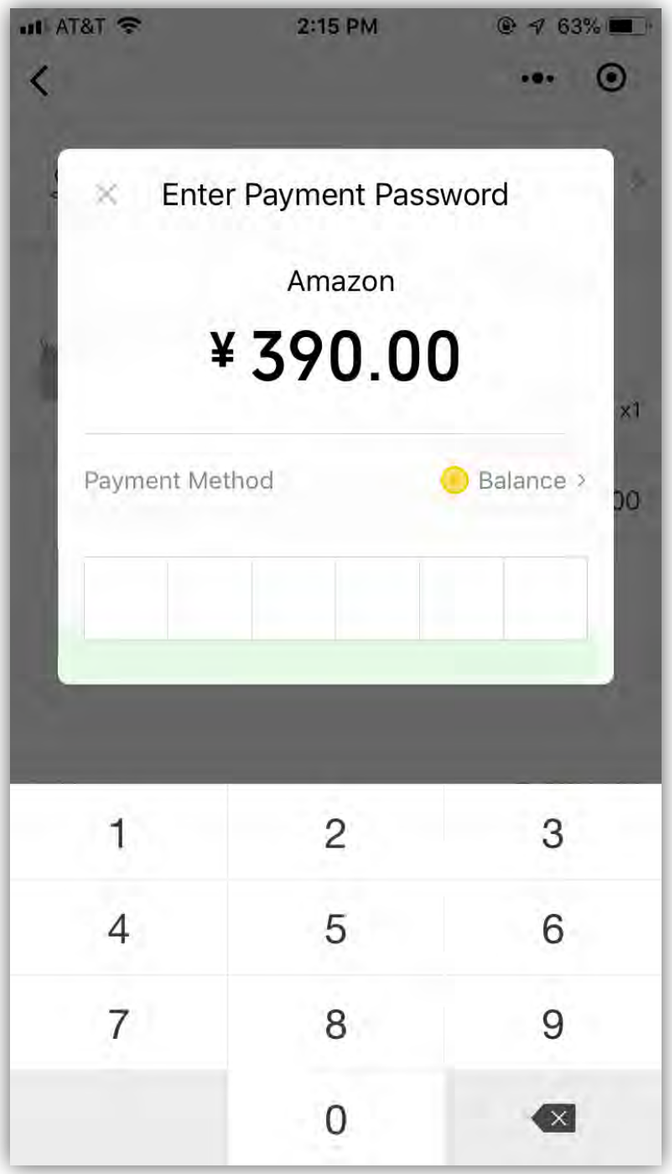
- System Resource Exposure
  - Weakness in System Resource Management
- New Overlay Hazard
  - Weakness in Access Control Management
- **Sub-window Deception**
  - **Weakness in UI Management**
- Sub-app Lifecycle Hijacking
  - Weakness in lifecycle management

Sub-window Deception

- app-in-app UI model
  - sub-app takes over the screen

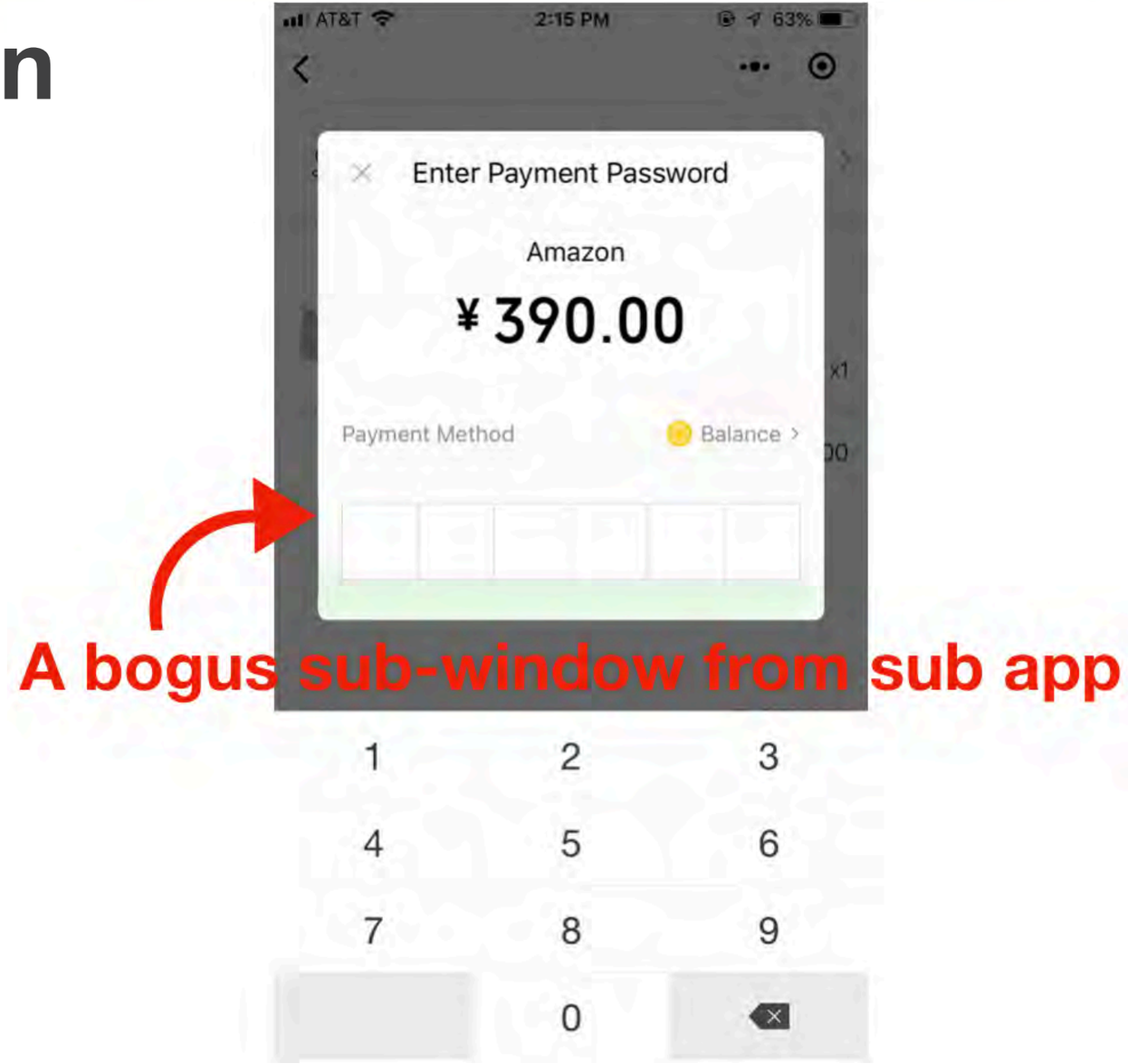
Sub-window Deception

- app-in-app UI model
  - sub-app takes over the screen
- Sensitive UI of the host app
  - payment password



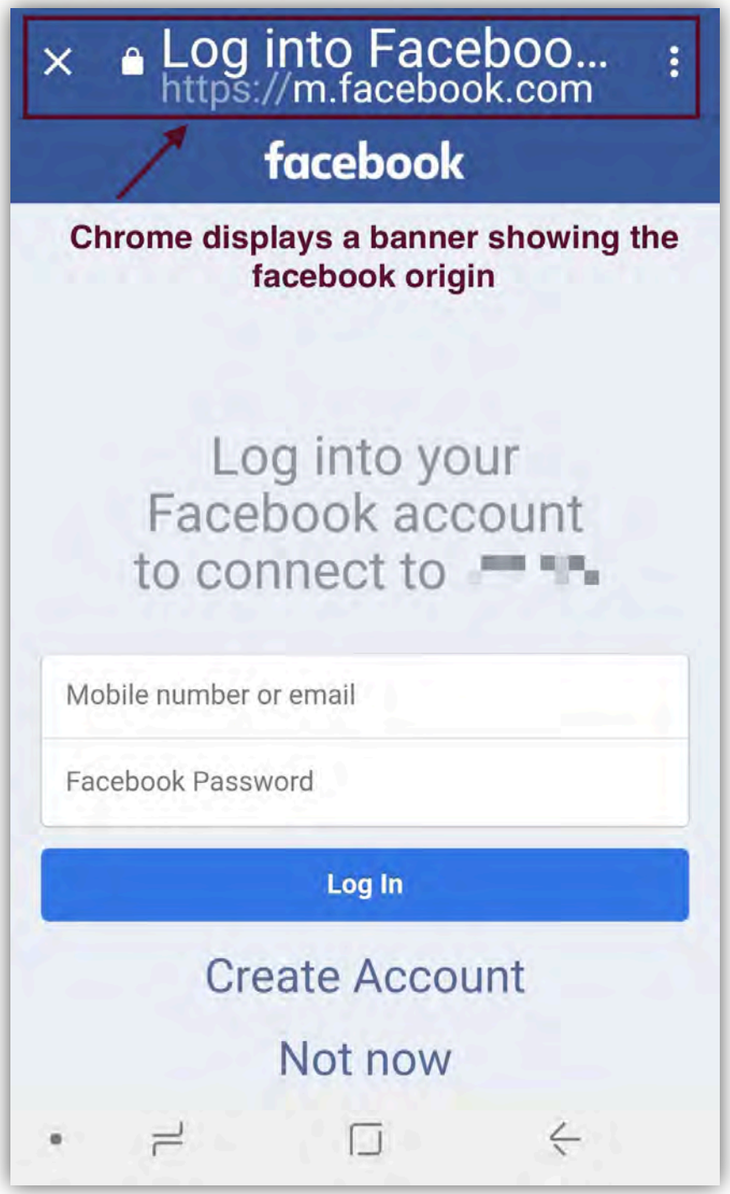
Sub-window Deception

- app-in-app UI model
  - sub-app takes over the screen
- Sensitive UI of the host app
  - payment password



Sub-window Deception

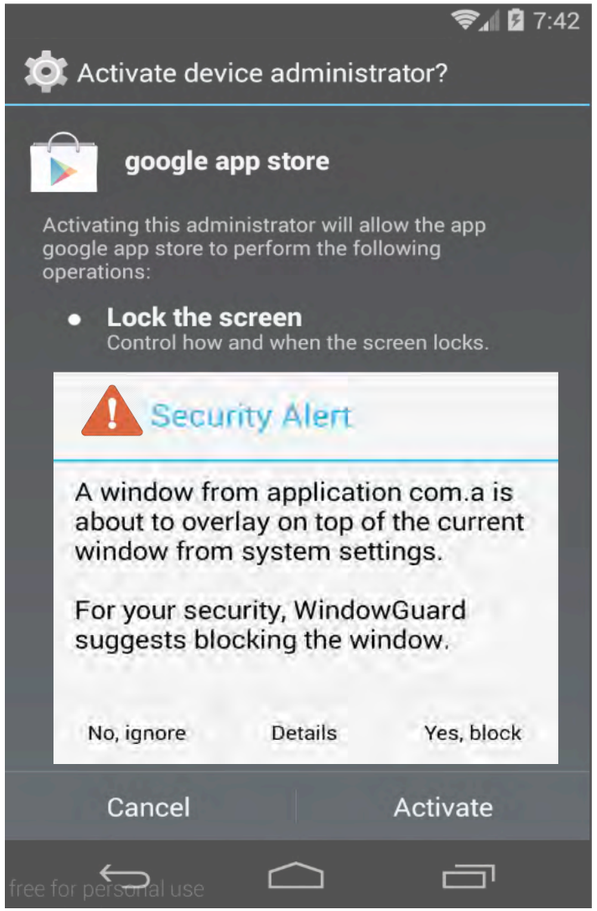
- app-in-app UI model
  - sub-app takes over the screen
- Sensitive UI of the host app
  - Web App navigation address bar



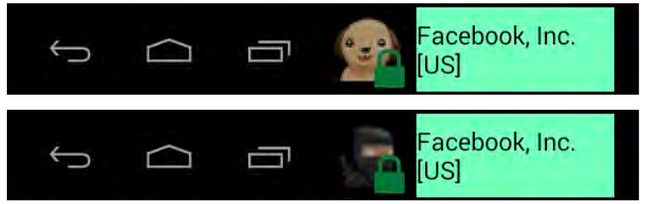
Defense

- Current anti-phishing techniques
  - app level ✓
  - sub-app level ✗

WindowGuard, NDSS'17



What the app is that? S&P'15

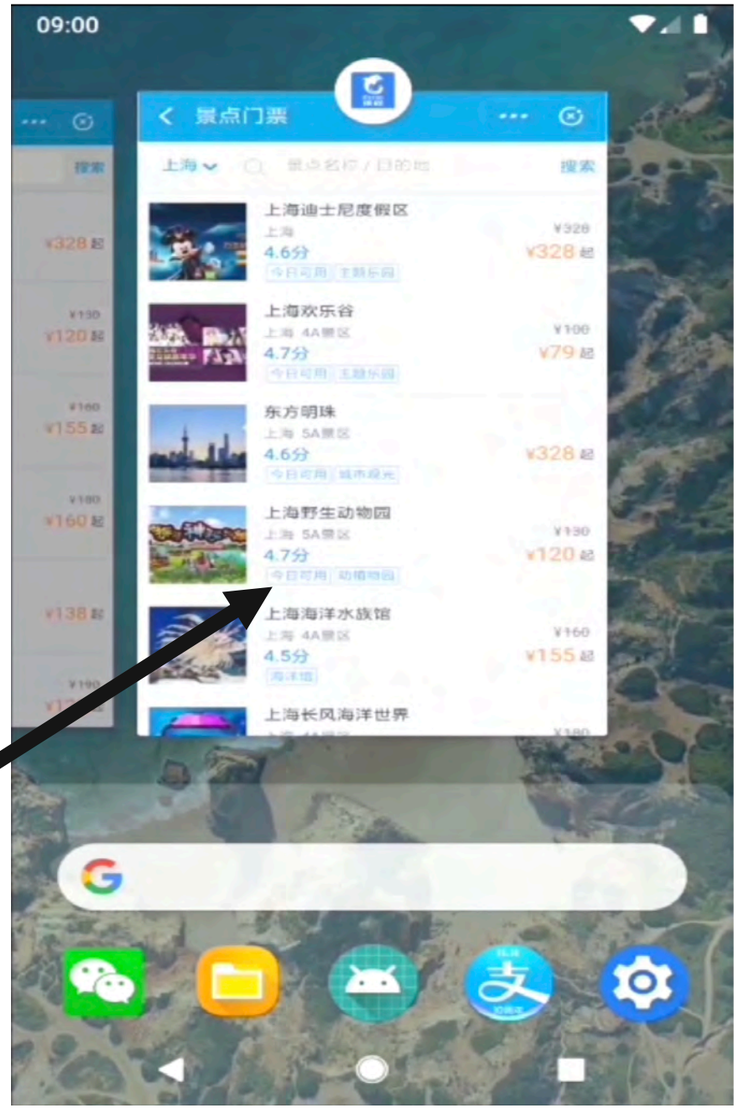


Security Weaknesses and Attacks

- System Resource Exposure
  - Weakness in System Resource Management
- New Overlay Hazard
  - Weakness in Access Control Management
- Sub-window Deception
  - Weakness in UI Management
- **Sub-app Lifecycle Hijacking**
  - **Weakness in lifecycle management**

Sub-app Lifecycle Hijacking

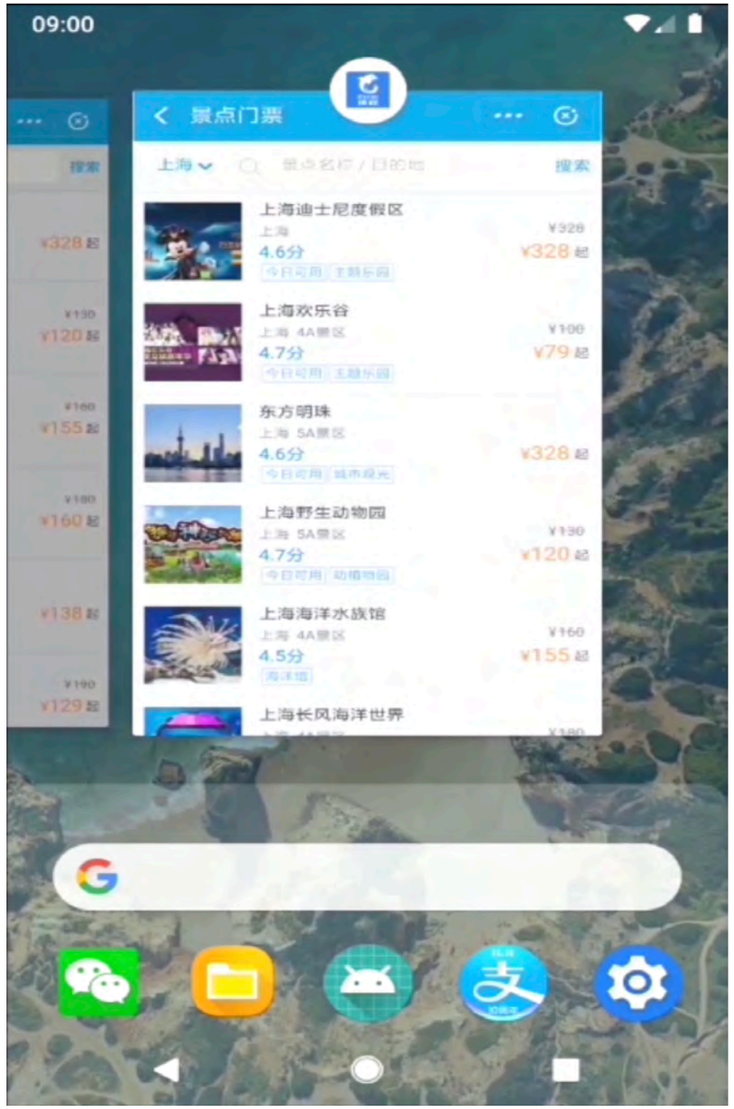
- Host app create tasks in the *Recents* screen for each sub-app



Inserted by WeChat

Sub-app Lifecycle Hijacking

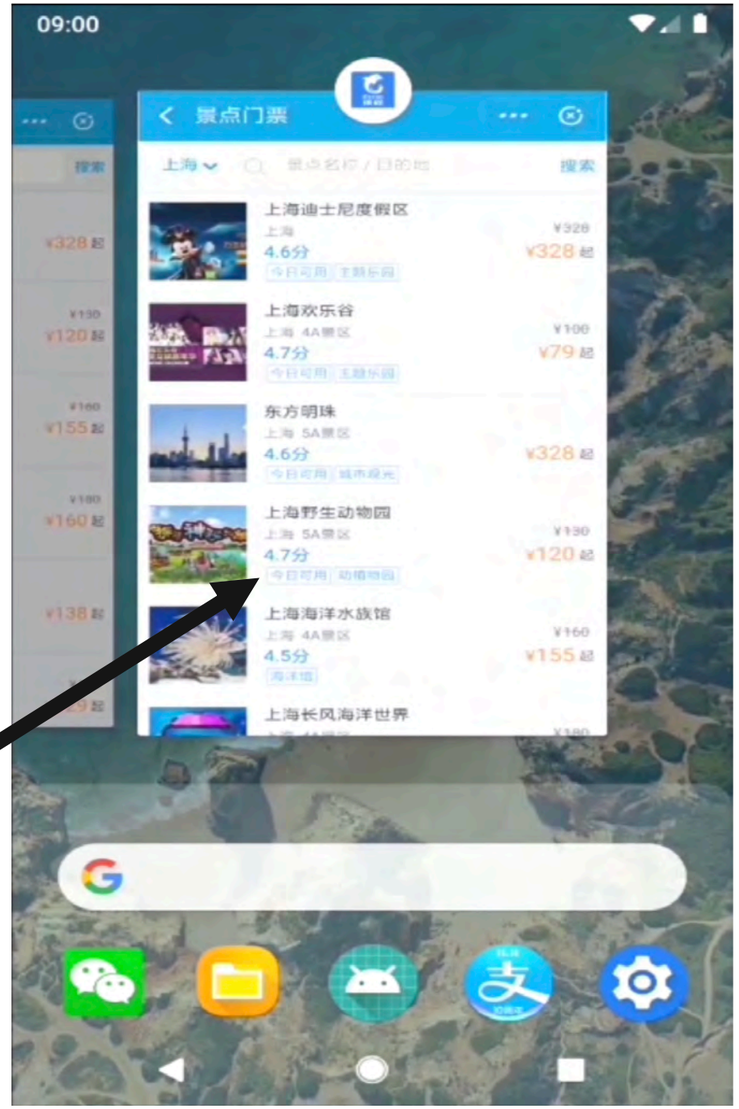
- Host app create tasks in the *Recents* screen for each sub-app
  - Limited # of simultaneous sub-app
  - Silent Mandatory recycling
  - e.g. the 1st sub-app disappears after the 6th sub-app is launched



Sub-app Lifecycle Hijacking

- A malicious app can imitate the disappeared sub-app
  - insert a new task in the Recents screen. (Task Hijacking)

- What is the disappeared sub-app?
- When to insert?

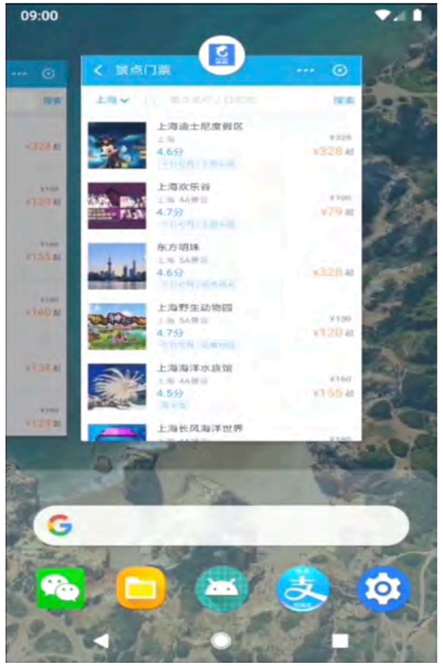


Inserted by Malicious App

Sub-app Lifecycle Hijacking

Sub-app side channel

- Observe the sub-app launch operations in real-time
- Sub-app launch -> file change in external file storage



tencent/MicroMsg/wxacache

Sub-app Lifecycle Hijacking

Consequence:

- health sub-app -> health information
- Banking sub-app -> credentials
- traveling sub-app -> travel history/plan
- …

Measurement

	iOS	Android
System Resource Exposure	A, D, J, Q, T, W	A, D, J, Q, T, W
Sub-app Permission Acquisition	-	A, B, C, D, F, O, W, Q
Sub-window Deception	A, B, D, W, S, Q	A, B, C, D, F, W, O, Q
Sub-app Lifecycle Hijacking	-	A, W, Q

A: HostApp A; B: Baidu; C: Chrome; D: DingTalk; F: Firefox; O: Opera; Q:QQ; S: Safari; W: WeChat

We reported all vulnerabilities and our PoC attacks to affected host app vendors, who all acknowledged the problems (see https://sites.google.com/view/appinapp/home).

Lessons and Conclusions

- Risks in app-in-app system:
  - Lack of security standard
  - Host app’s limited app-level capabilities
  - Lack of OS-level support

Thank you!