A Deep Dive into Privacy Dashboard of Top Android Vendors

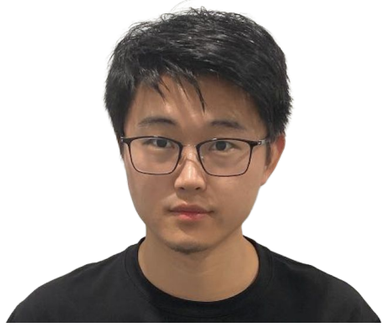
Bin Ma, Zhenyu Zhu, Wei Wen, Xiangxing Qian

About us

Bin Ma

Zhenyu Zhu

Wei Wen



Xiangxing Qian

- Security researcher at IES Red Team of ByteDance
- Privacy protection research involve Apps and Vendors
- Applied security including Mobile, Web and Cloud

Outline

- How Privacy Dashboard Works
- Android AppOps Introduction
- Details and Flaws in Vendors' Privacy Dashboards
- Measurement and Test Methods
- Conclusion

1. How Privacy Dashboard Works

Why do we need privacy dashboard ?

- **What can Apps do to invade user privacy ?**

Data access behavior	Data tamper behavior	Sensitive Action	Startup Action
- Steal Contacts - Scan Media Files - Read personal data - … ...	- Write to Calendar - Rewrite Clipboard - Delete Media File - … …	- Hidden camera - Unauthorized recording	- AutoStart - Chain Start

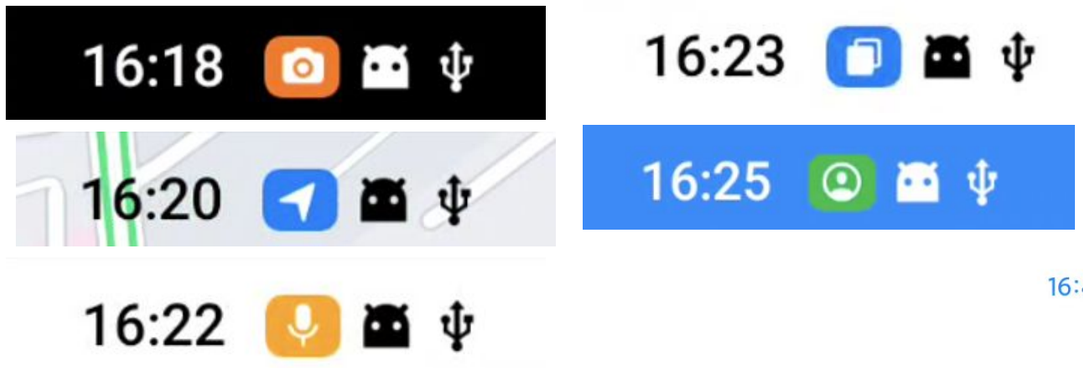
- **What can Operating System do to protect users privacy ?**

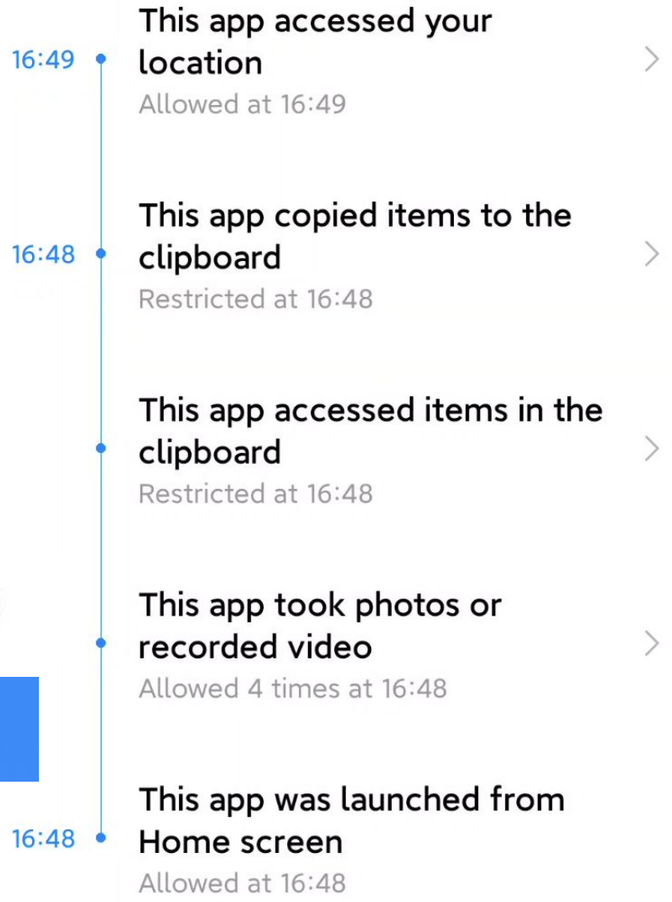
- Restrict the system API.
- Privacy dashboard.

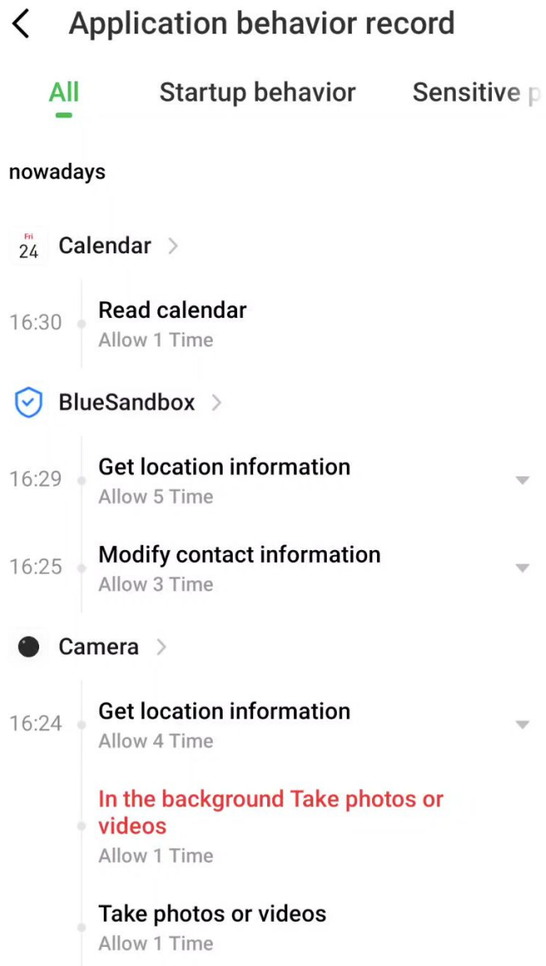
What is privacy dashboard ?

A permission record center that allows users to see how Apps accessing data.

- Monitor sensitive behaviors of a specific app
- Shows the timeline of a certain permission used by different apps
- Prompt some sensitive access in real time

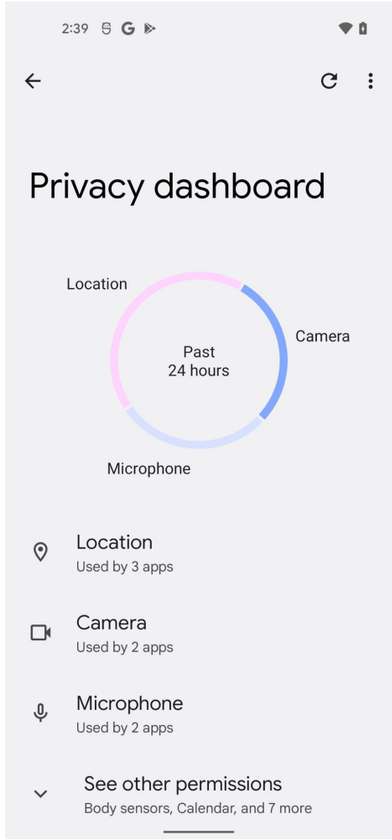


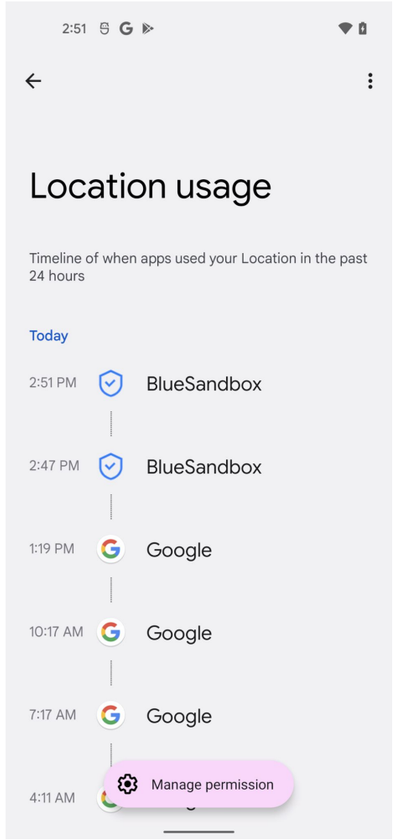




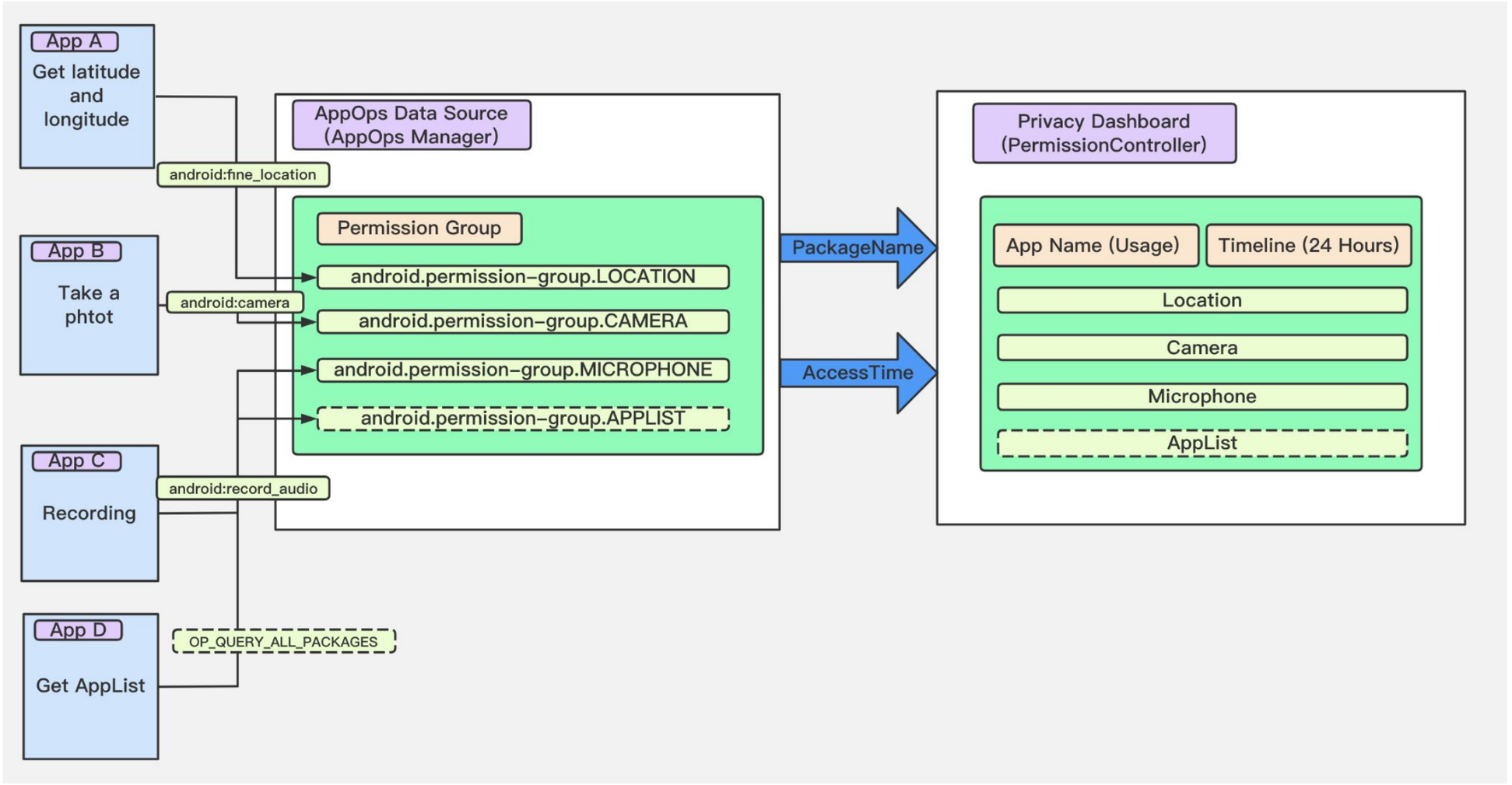
Privacy Dashboard & Android 12

- Monitor permissions
  - Location
  - Microphone
  - Camera
- Shows the Apps access to these permission
- Shows the timeline and frequency of these access.





How do these privacy dashboards work ?



2. Android AppOps Introduction

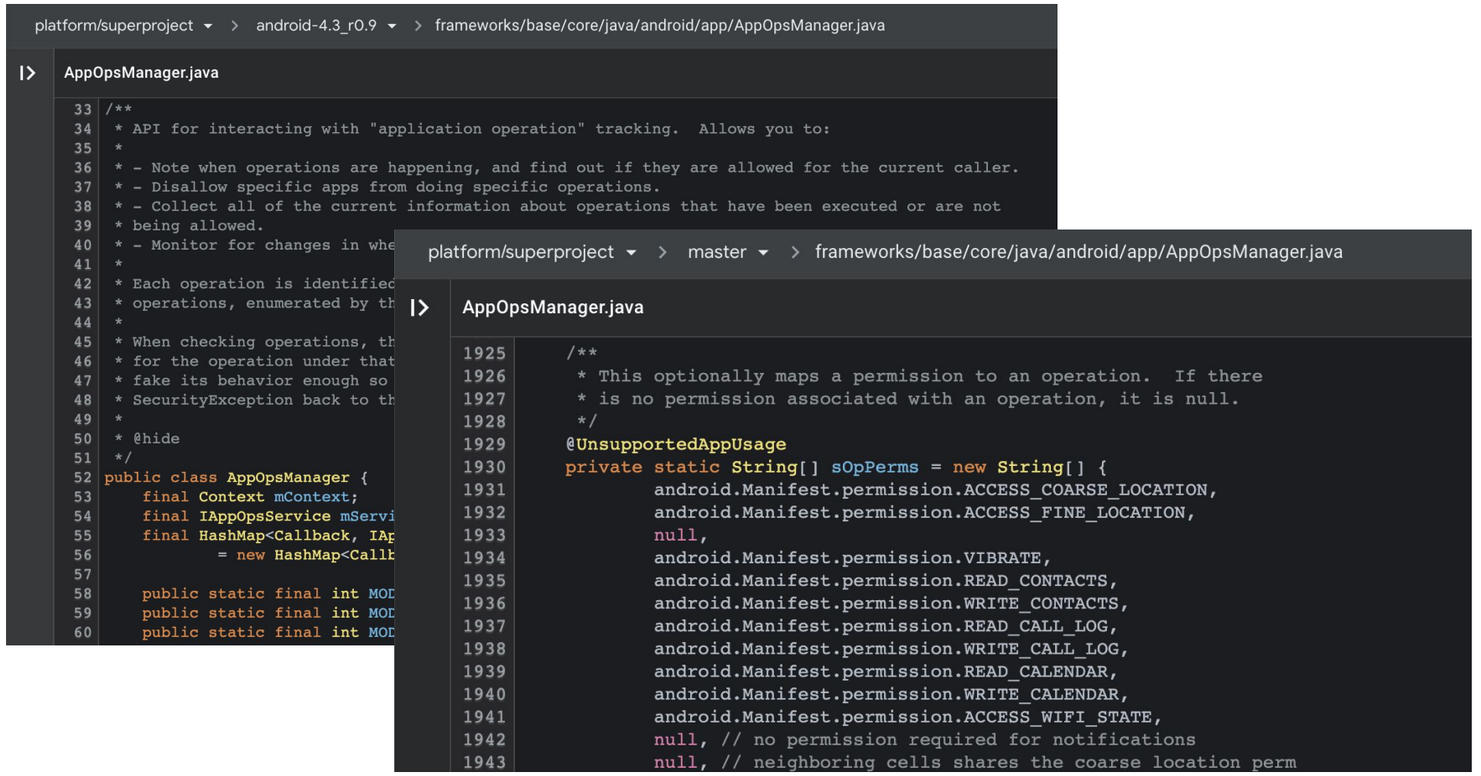
What is AppOps?

Introduction

- Formally added in Android 4.3
- Unit - uid/package
- Access control and tracking
  - Permission
  - Operation

Operations vs Permissions

- 53 permissions but 105 ops
- Clipboard and so on



Work mechanism and flow of AppOps

Check Permission

- Befor check op
- CheckPermission()

Check and Note op

- note[Proxy]Op[NoThrow]
- startOp[NoThrow]

General

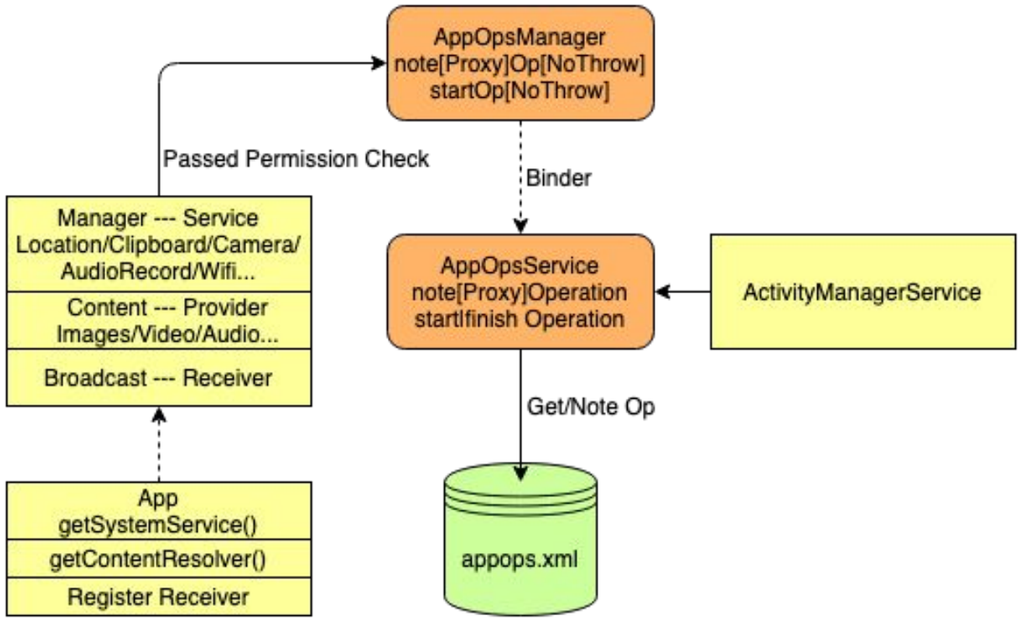
- Manager/Service

  AppOpsManager API

Exception

- AMS

  AppOpsService API



AppOps data dump

- dumpsys appops or appops dump
  - -- package
- record detail
  - Op String
  - Allow/default/ignore...
  - Access or reject time
  - Background or foreground
  - Proxy or not
  - Duration time

```
~> adb shell appops dump | more
Current AppOps Service state:
  Settings:
    top_state_settle_time=+30s0ms
    fg_service_state_settle_time=+10s0ms
    bg_state_settle_time=+1s0ms

  Op mode watchers:
    Op COARSE_LOCATION:
      #0: ModeCallback{11
      #1: ModeCallback{5a
      #2: ModeCallback{96
    Op READ_CALL_LOG:
      #0: ModeCallback{8a
    Op WRITE_CALL_LOG:
      #0: ModeCallback{8a
    Op READ_SMS:
      #0: ModeCallback{8a
```

```
Package com.android.settings:
  VIBRATE (allow):
      Access: [pers-s] 2021-09-23 18:06:42.068 (-2d22h25m45s342ms) duration=+21ms proxy[uid=0, pkg=null]
  WRITE_SETTINGS (default):
      Reject: [pers-s]2021-09-26 16:03:03.710 (-29m23s700ms) proxy[uid=0, pkg=null]
  CAMERA (allow):
      Access: [pers-s] 2021-06-20 20:04:00.140 (-97d20h28m27s270ms) duration=+10s647ms proxy[uid=0, pkg=null]
  READ_CLIPBOARD (allow):
      Access: [pers-s] 2020-11-10 21:46:15.079 (-319d18h46m12s331ms) proxy[uid=0, pkg=null]
  WRITE_CLIPBOARD (allow):
      Access: [pers-s] 2021-06-01 15:13:05.613 (-117d1h19m21s797ms) proxy[uid=0, pkg=null]
  GET_USAGE_STATS (default):
      Reject: [pers-s]2021-09-26 12:14:16.780 (-4h18m10s630ms) proxy[uid=0, pkg=null]
  TOAST_WINDOW (allow):
      Access: [pers-s] 2021-08-18 12:11:42.273 (-39d4h20m45s137ms) duration=+2s543ms proxy[uid=0, pkg=null]
  WRITE_EXTERNAL_STORAGE (allow):
      Access: [pers-s] 2021-09-17 11:16:22.184 (-9d5h16m5s226ms) proxy[uid=0, pkg=null]
  REQUEST_DELETE_PACKAGES (allow):
      Access: [pers-s] 2021-06-08 12:10:37.176 (-110d4h21m50s234ms) proxy[uid=0, pkg=null]
Package com.google.android.hiddenmenu:
  WRITE_SETTINGS (default):
      Reject: [pers-s]2021-01-22 14:38:34.348 (-247d1h53m53s62ms) proxy[uid=0, pkg=null]
  REQUEST_DELETE_PACKAGES (allow):
      Access: [pers-s] 2021-08-13 16:18:48.926 (-44d0h13m38s484ms) proxy[uid=0, pkg=null]
Package com.android.wallpaperbackup:
  REQUEST_DELETE_PACKAGES (allow):
      Access: [pers-s] 2021-08-13 17:18:03.455 (-43d23h14m23s955ms) proxy[uid=0, pkg=null]
```

Interface for AppOps data extraction

Apps Developers

- **How**
  - AppOpsManager.OnOpNotedCallback
    - logPrivateDataAccess
    - onNoted/onSelfNoted/onAsyncNoted
  - AppOpsManager.setOnOpNotedCallback
- **When**
  - Single Activity onCreate()
  - Application attachBaseContext()

Security Tester

- Static inject
  - App -- modify bytecode and repack
- Dynamic inject
  - App -- inject *OnOpNotedCallback
  - system_server -- hook noteOperation, etc.
- Custom rom
  - How Vendors do?

3. Privacy Dashboards of Vendors

Privacy dashboard technique of custom vendors

- Extract AppOps records purely
  - accept/reject
  - access time
  - background/foreground
  - duration
- AppOps modification
  - self-designed opcode
  - noteOperation/noteProxyOperation
- Custom implementation
  - empty or temporary privacy data
  - album delete protection

Privacy Dashboard Overview of Android vendors

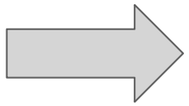
Privacy Dashboard		A	B	C	D	E
Privacy Data Collection	Location	**TBI**	**TBI**	**TBI**	**TBI**	✅
	Camera	✅	✅	✅	✅	✅
	Microphone	✅	✅	✅	✅	✅
	Contacts	✅	✅	✅	✅	✅
	Message	✅	✅	✅	✅	✅
	Clipboard	✅	**TBI**	✅	❌	❌
	Device Info	**TBI**	✅	✅	✅	❌
	External Storage	✅	✅	✅	✅	✅
	App List	❌	❌	✅	✅	❌
Background Behavior		✅	✅	❌	❌	❌
Frequency		✅	**TBI**	❌	❌	**TBI**
Album Protection		**TBI**	**TBI**	✅	**TBI**	**TBI**

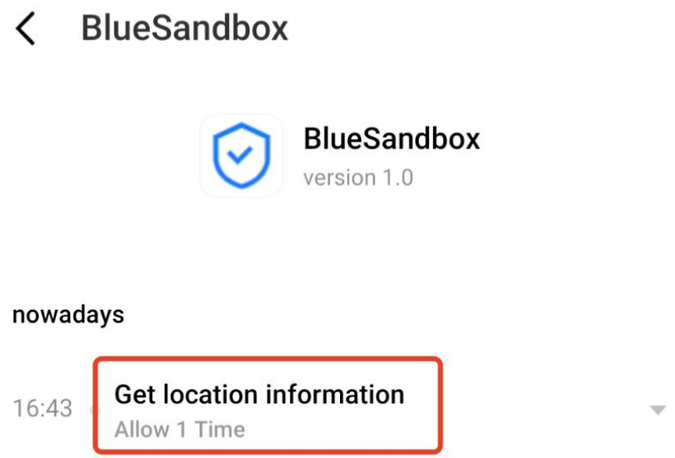
✅ Support ❌ Not Support **TBI** To be Improved, Partially Support or Inaccurate

Case 1: Get network signal but regard as location collection

```
//AndroidMenifest
<uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>



//Code Snippet
WifiManager wifiManager = (WifiManager)
context.getSystemService(context.WIFI_SERVICE);

WifiInfo wifiInfo = wifiManager.getConnectionInfo();

int rssi = wifiInfo.getRssi();
```

App call getConnectionInfo to get RSSI signal





Privacy Dashboard result of Vendor C

Case 1: Get network signal but regard as location collection

- Call Stack

```
WifiManager.getConnectionInfo()
    WifiServiceImpl.getConnectionInfo()
        WifiPermissionsUtil.enforceCanAccessScanResults()
            WifiPermissionsUtil.checkCallersLocationPermission()
                WifiPermissionsUtil.noteAppOpAllowed()
                    AppOpsManager.noteOp(OPSTR_FINE_LOCATION)
```

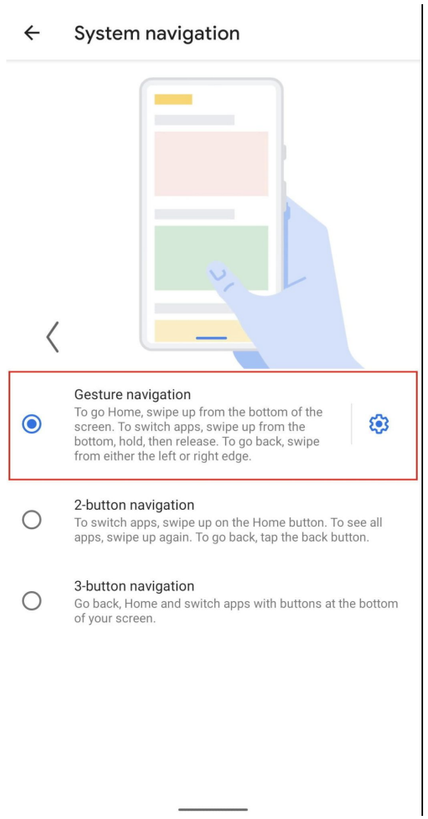
- Over-designed monitoring strategy
  - location-related API(getSSID、getBSSID) but not getRssi
  - Same implementation in AOSP

Case 2: Background camera obtain caused by gesture navigation

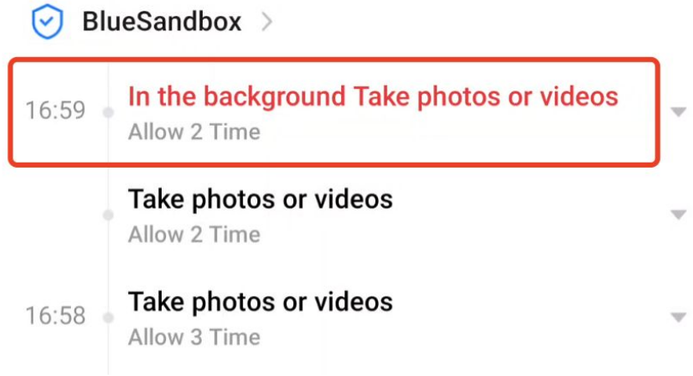
```
public void onPause() {
    super.onPause();
    //Some recycle code
    mCamera.release();
}
```

Camera release normally

+



Gesture Navigation



Privacy Dashboard result of Vendor A

Case 2: Background camera obtain caused by gesture navigation

```
→  /tmp adb shell dumpsys appops --package [redacted] --op CAMERA
Current AppOps Service state:
  Settings:
    top_state_settle_time=+5s0ms
    fg_service_state_settle_time=+5s0ms
    bg_state_settle_time=+1s0ms


  Uid u0a293:
    state=cch
    capability=---
    appWidgetVisible=false
      CAMERA: mode=ignore
    Package [redacted]:
      CAMERA (allow):
        null=[
          Access: [top-s] 2021-09-24 16:59:44.783 (-29m36s214ms) duration=+2s902ms
          Access: [bg-s] 2021-09-24 16:59:47.686 (-29m33s311ms) duration=+60ms
        ]
```

bg-s: background self

duration: obtain time

Case 2: Background camera obtain caused by gesture navigation

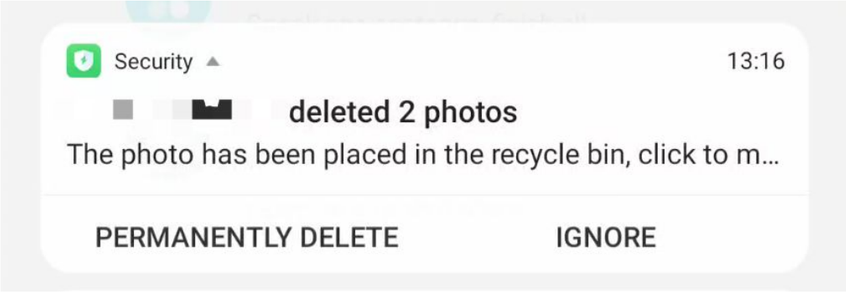
- Critical point between fg and bg - uid stage change
- Time delta from onPause to onUidStateChange

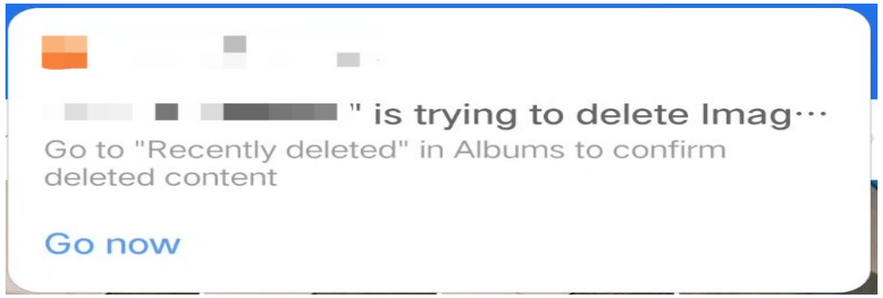
Gesture			3-button		
onPause	onUidStateChange	Time(ms)	onPause	onUidStateChange	Time(ms)
00:15:16.228	00:15:16.547	319	00:18:50.736	00:18:51.416	520
00:15:35.211	00:15:35.524	313	00:19:06.964	00:19:07.596	568
00:15:44.235	00:15:44.596	361	00:19:13.507	00:19:14.26	519
00:15:54.436	00:15:54.762	326	00:19:21.844	00:19:22.446	598
00:16:03.650	00:16:04.16	366	00:19:30.672	00:19:31.273	601

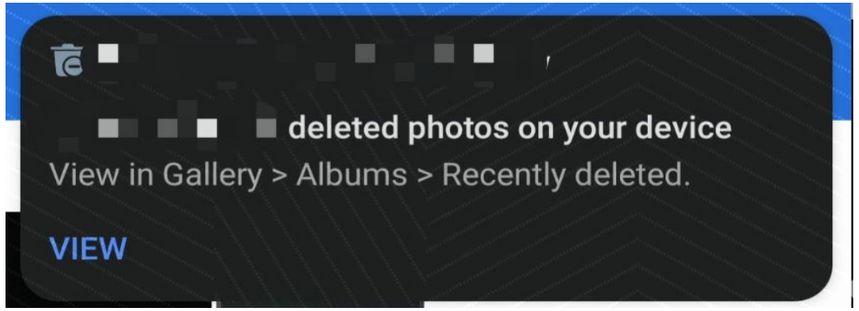
- Gesture navigation with UI change may cause false report

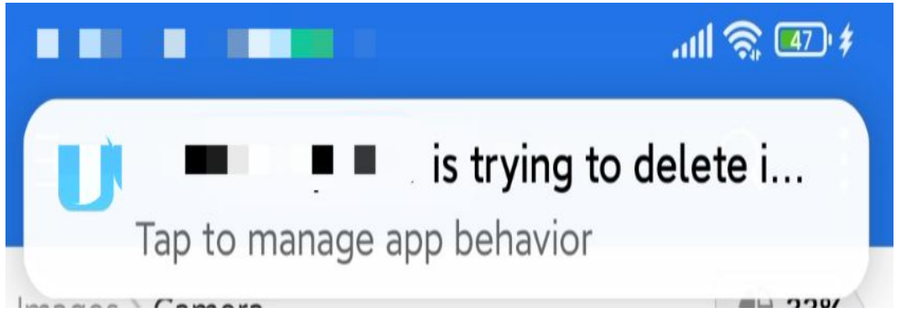
Case 3: App may bypass album protection to delete files

- Delete photo options blocked by vendor's protection









Case 3: App may bypass album protection to delete files

- File delete monitor logic and bypass ways

Vendors	Protection	Bypass
B	override “delete” of MediaProvider	java.io.File.delete()
E	“DCIM” path & override java.io.File.delete()	native delete (libc remove)
A & C	“DCIM” path &“unlink” hook	rename files to other path, then delete

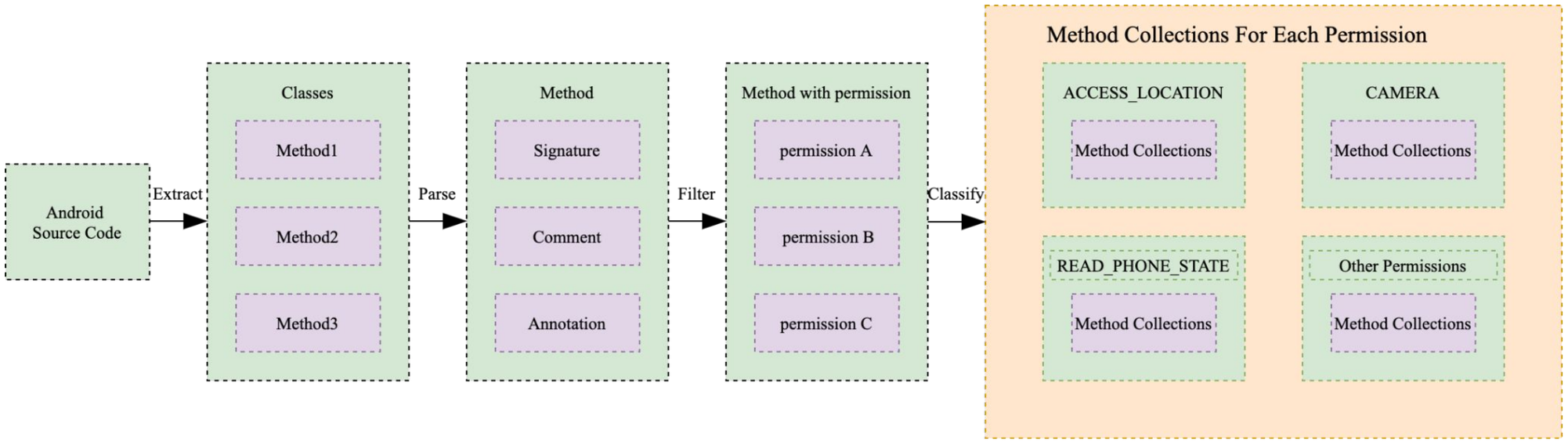
4. Measurement and Test Methods

Measurement And Test Methods

- Measurement: Two Ways of Extracting AppOps API
  - AppOps APIs From Source Code Analysis
  - Sensitive APIs From Reachability Analysis

- Test Methods
  - BlueSandbox: A Sensitive API Verify Tool

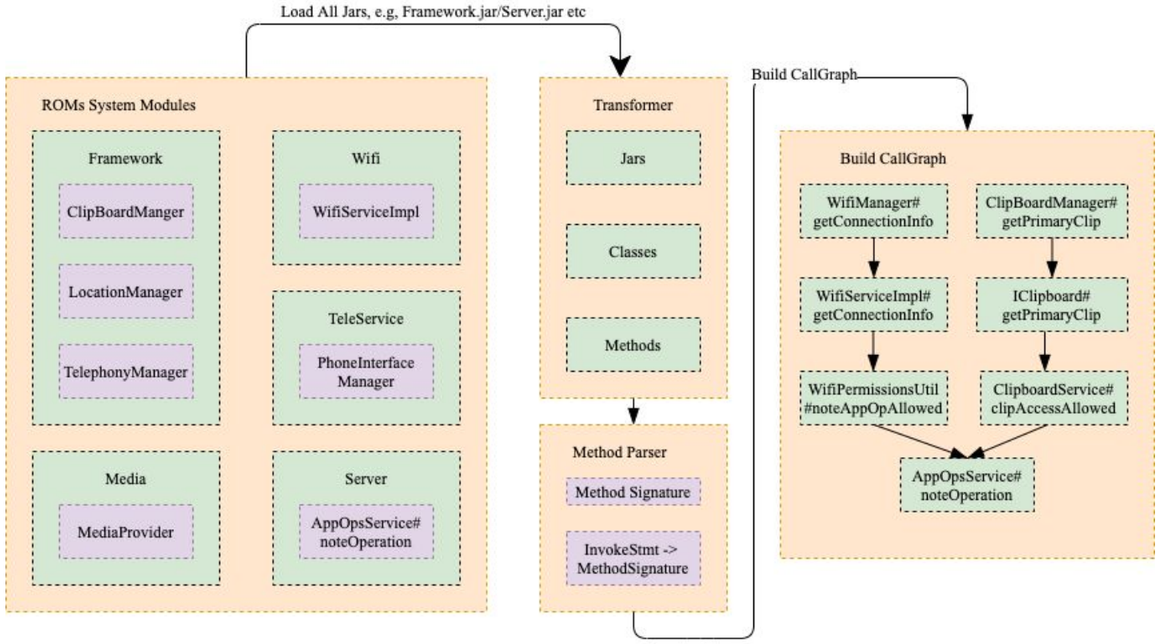
AppOps API From Source Code Static Analysis



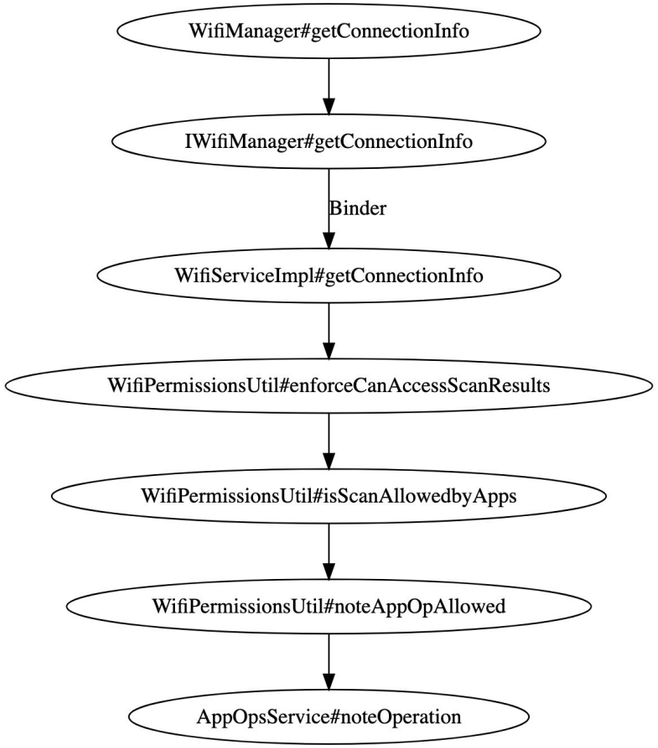
Steps:

1. Extract All Classes from Android Source Code
2. Extract All Methods from Classes, and build **<Signature, Comment, Annotation>** For Each Method
3. Filter Methods with **Comment or Annotation Marked Permissions**
4. Classify Method Collections For Each Permission

AppOps API From Reachability Analysis



Reachability Analysis

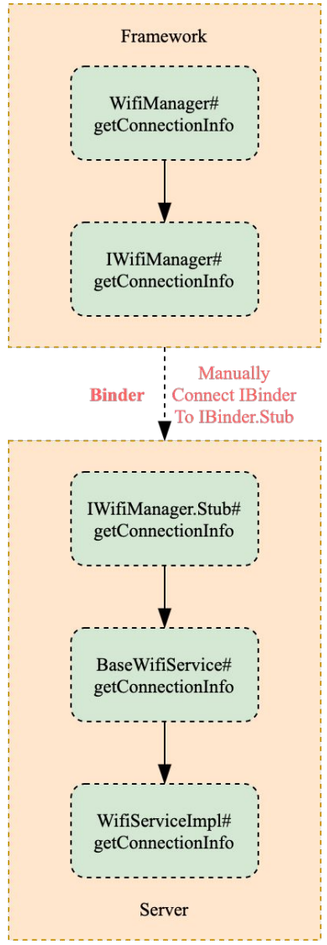


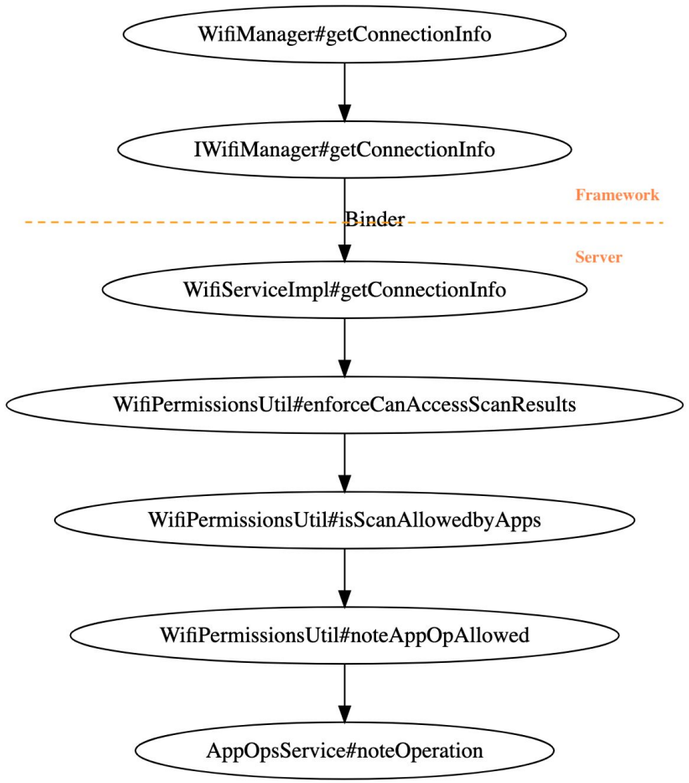
Steps:

1. Preparation: Framework/Wifi/Media/Server/TeleService Jars
2. Transformer: extract methods, construct <Method, InvokeMethod> Collections
3. CallGraph Builder: build a fully CallGraph within system modules
4. Method Collections: find all reachable paths to AppOps through Reachability Analysis

Difficulty: Build A Fully CallGraph Within System Modules

```
# Framewwork android.net.wifi.WifiManager
public WifiInfo getConnectionInfo() {
    // invoke statements
    return mService.getConnectionInfo();
}

# Server com.android.server.wifi.WifiServiceImpl
public class WifiServiceImpl extends IWifiManager.Stub {
    public WifiInfo getConnectionInfo(String, String) {
        mWifiPermissionsUtil.enforceCanAccessScanResults();
    }
}
```



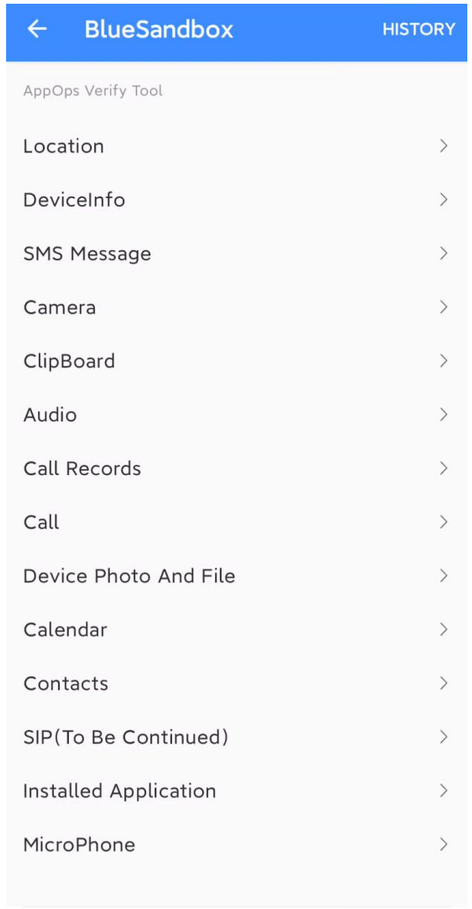


A Connected CallGraph

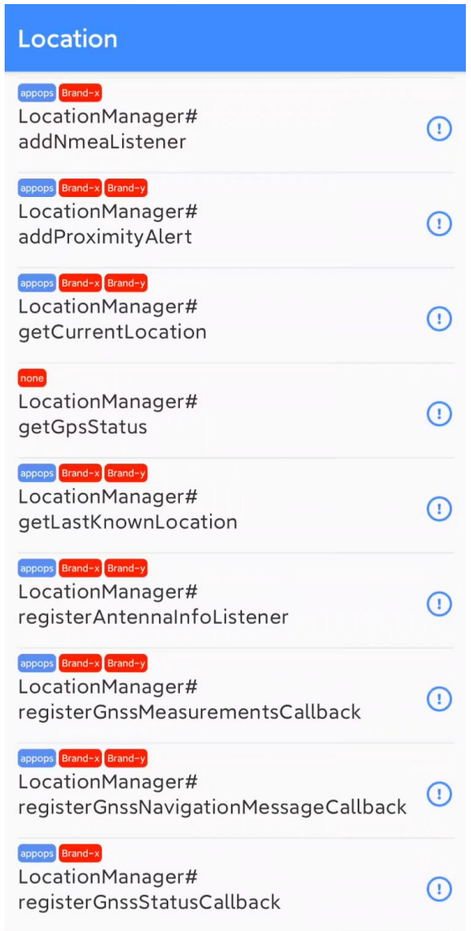
Binder: Code Snippet

Binder Connection In Static Analysis

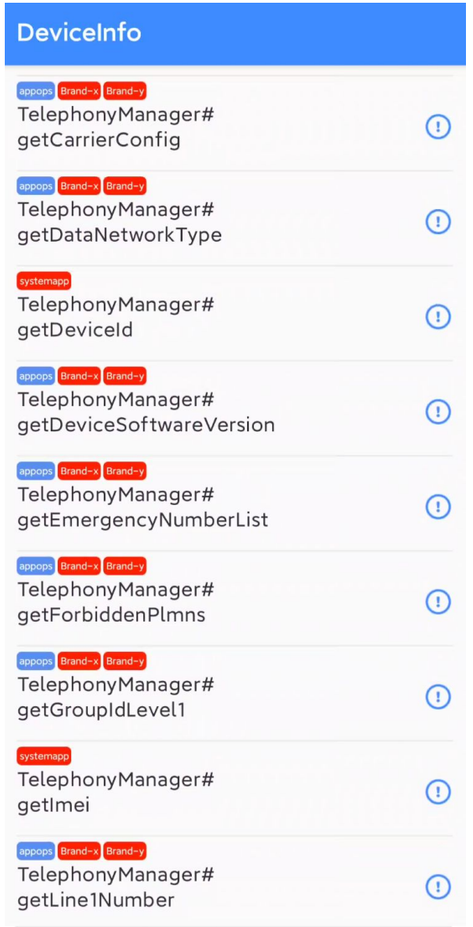
BlueSandbox: Sensitive API Verify Tool



BlueSandbox



e.g, Location APIs



e.g, DeviceInfo APIs

5. Conclusion

- Privacy Dashboard is a new feature on Android 12 aimed to keep track of apps' sensitive behaviors, but several android vendors achieved their own before

- AppOps provide interfaces to track sensitive permission access but not accurate or comprehensive for Vendors to present on privacy dashboard directly

- Vendor's custom privacy protection mechanism need to be improved and verified fully to protect billions of users' privacy