THE UNIVERSITY OF QUEENSLAND AUSTRALIA

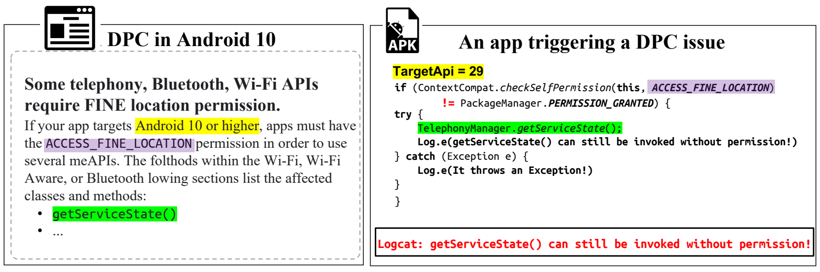
Investigating Documented Privacy Changes in Android OS

Chuan Yan, Mark Huasong Meng, Fuman Xie, Guangdong Bai
The University of Queensland, Australia

FSE 24

Introduction

In recent years, Android has taken proactive measures to adapt its access control policies for such data, in response to the increasingly strict privacy protection regulations around the world. When each new Android version is released, its privacy changes induced by the version evolution are transparently disclosed, and we refer to them as ***documented privacy changes*** (DPCs). However, whether the actual access control enforcement in the OS implementations aligns with the disclosed DPCs becomes a critical concern. In this work, we conduct the first systematic study on the consistency between the operational behaviors of the OS at runtime and the officially disclosed DPCs. We propose **DopCheck**, an automatic DPC-driven testing framework equipped with a large language model (LLM) pipeline. It features a serial of analysis to extract the ontology from the privacy change documents written in natural language, and then harnesses the few-shot capability of LLMs to construct test cases for the detection of ***DPC-compliance issues*** in OS implementations.



DPC in Android 10

Some telephony, Bluetooth, Wi-Fi APIs require FINE location permission.

If your app targets Android 10 or higher, apps must have the `ACCESS_FINE_LOCATION` permission in order to use several meAPIs. The folthods within the Wi-Fi, Wi-Fi Aware, or Bluetooth lowing sections list the affected classes and methods:

- `getServiceState()`
- ...

An app triggering a DPC issue

```
TargetApi = 29
if (ContextCompat.checkSelfPermission(this, ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
try {
    TelephonyManager.getServiceState();
    Log.e(getServiceState() can still be invoked without permission!)
} catch (Exception e) {
    Log.e(It throws an Exception!)
}
}

Logcat: getServiceState() can still be invoked without permission!
```

Fig 1. An example of the DPC issue. Android 10's documentation states that the API getServiceState() requires a particular permission (left), whereas an app that does not request the permission can still invoke it without throwing any exception (right)

DPC Ontology Construction

Table 1. Nine types of DPC entities

Entity Category	Entity	Pattern/Archors	Regex	Semantics
Aggregate	version	[Tag:*p*] [Start: "Android", End: 9-13]	-	Lower/Higher
	application	[Tag:*p*] ["app" + verb]	-	Hyperlink
Subject	API	[Tag:*code*] [End: ()]	\b[a-zA-Z_0-9]+\(\)	Hyperlink, Class, S/I,[1] expected return value
Property	permission	[Tag:*code*] [All capital letters joined with _]	([A-Z]+_)+[A-Z]+	Hyperlink, P/N[2]
	attribute	[Tag:*code*] [<>]	-	Hyperlink, P/N
Result	exception	[Tag:*code*] ["throw", "occur", End: "Exception"]	-	Condition[3]
	return	[Tag:*p*] ["return", End: "Noun/Num Entity"[4]]	-	API
	figure	[Tag:*figure*]	-	-
	effect	[Tag:*p*][Infinitive clause,[5] sentence contains API]	-	API

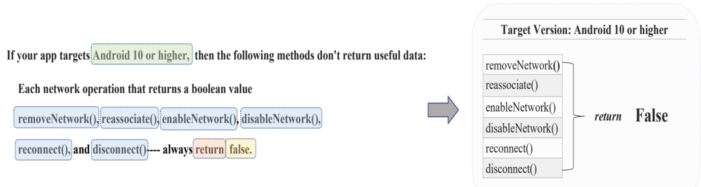

If your app targets Android 10 or higher, then the following methods don't return useful data:
Each network operation that returns a boolean value
removeNetwork(), reassociate(), enableNetwork(), disableNetwork(),
reconnect(), and disconnect()---- always return false.
Target Version: Android 10 or higher
removeNetwork()
reassociate()
enableNetwork()
disableNetwork()
reconnect()
disconnect()
return
False

Fig 2. An example of the API-return subsumptive relationship

Considering that DopCheck's test cases mostly center around APIs, we formulate the name of an API as the subject entity, and define another 8 DPC entities to construct the context of an API invocation.

After DopCheck has recognized entities, its next objective is to discover subsumptive relationships that establish connections among entities, and thereby facilitate the test case generation process. we have defined five subsumptive relationships that are based on the connection between entities.

Test Case Generation

Table 2. Four categories of DPCs to facilitate assertion construction

DPC Category	Entity Criteria		Testing Mode
	Required Entities[1]	Optional Entities[2]	
API changes	*API*	*permission, attribute, exception, return, effect*	Return value validation
Permission changes	*permission*	*API, attribute, exception, return*	GUI/Prompt validation
GUI changes	*figure*	*API, exception, return*	GUI validation
Attribute changes	*attribute*	*API permission, , exception, return*	Property testing

Message

System: You are a senior Android development engineer.

User: Give me two examples of call to `CallStateChanged()`, one has `READ_CALL_LOG`, the other does not have.

Assistant: Manually write the `CallStateChanged()` test cases.

Fig 3. An example of in-context learning

DopCheck harnesses the concept of in-context learning, which allows us to guide the model in producing results that align with specific requirements in a few-shot.

During the execution of the test cases, DopCheck checks whether the invocation of the APIs leads to expected behaviors or not. for example, to check whether the return value matches the descriptions in the document, or to check whether an expected security exception is thrown. To define assertions for this purpose, we take a category-wise strategy.

Conclusion

DopCheck managed to identify a total of 19 bugs, with 13 of them discovered in Android 13 and 6 in Android 10 for the first time. Our work reveals the inconsistency between what does documentation claim and how Android OS actually behave. Our findings emphasize the importance of further research and action to address discrepancies in DPCs, aiming to better align documented capabilities with their actual behavior.

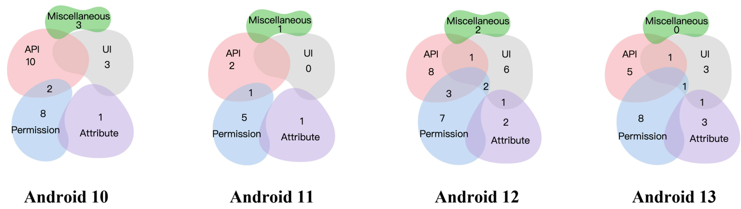
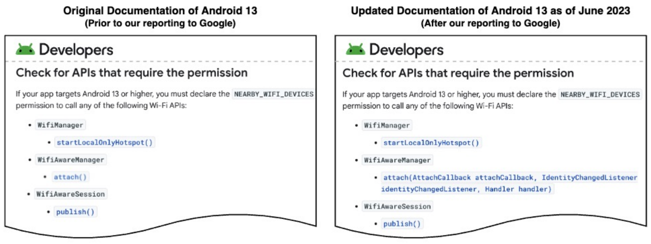


Fig 4. Distribution of DPC categories for each Android version



Original Documentation of Android 13
(Prior to our reporting to Google)

Developers

Check for APIs that require the permission

If your app targets Android 13 or higher, you must declare the `NEARBY_WIFI_DEVICES` permission to call any of the following Wi-Fi APIs:

- `WifiManager`
  - `startLocalOnlyHotspot()`
- `WifiAwareManager`
  - `attach()`
- `WifiAwareSession`
  - `publish()`

Updated Documentation of Android 13 as of June 2023
(After our reporting to Google)

Developers

Check for APIs that require the permission

If your app targets Android 13 or higher, you must declare the `NEARBY_WIFI_DEVICES` permission to call any of the following Wi-Fi APIs:

- `WifiManager`
  - `startLocalOnlyHotspot()`
- `WifiAwareManager`
  - `attach(AttachCallback attachCallback, IdentityChangedListener identityChangedListener, Handler handler)`
- `WifiAwareSession`
  - `publish()`

Fig 5. Change in Android documentation before and after our reporting

Acknowledgments. This work was produced by UQ TrustLab in collaboration with the National University of Singapore (NUS) and led by Assoc. Prof. Guangdong Bai. This work is supported by Australian Research Council Discovery Projects (DP230101196, DP240103068).

UQ TrustLab

Read our paper