Revisiting the Privacy Compliance Analysis For App SDKs

Abstract—**The privacy issues of mobile apps and the integrated SDKs have drawn much concern in recent years. However, existing privacy compliance analysis systems have many limitations when analyzing apps' SDKs. In this paper, we identify the limitations and design a new system (named TBD). The experimental result shows that TBD.**

I. Introduction

The privacy issues of mobile apps [1], [2] and the integrated SDKs [3] have drawn much concern in recent years.

Nowadays, smartphones and mobile apps are playing essential roles in our daily lives. More and more developers tend to use many off-the-shelf third-party libraries (TPLs) to facilitate the development process, to avoid reinventing the wheel. However, developers may omit the privacy threat in TPLs. For example, the advertisement libraries may leak users' personal information (e.g., IMEI). Some TPLs (e.g., advertising and analytic libraries) rely on personal information to provide better service. However, TPLs may collect user data that is not necessary for providing related services. Such behaviors may lead to privacy leakage or even security concerns. Besides, the permission mechanism of Android also increases the possibility of leaking user privacy via TPLs. Since the host apps share the same process space and permissions with TPLs, the mechanism violates the principle of least privilege, which leads to TPLs being over-privileged. Recent studies find that permission abuse allows TPLs to access users' personal information and causes potential privacy leakage. In addition, the incorrect use of TPLs may cause serious personal information leakage. For example, a recent report discloses that popular Android apps with over 142.5 million installations leak user data to unauthorized third parties. Another study discloses 120 of 555 apps that share users' personal information to analytic service TPLs without encryption. Besides, linking and mining large amounts of unencrypted personal information probably pose threats to serious personal privacy leakage. Thus, it is urgent to design a system to check the data access behavior and privacy policies of TPLs so that we can determine if TPLs are compliant with privacy regulations.

Various privacy-related regulations (e.g., California Consumer Privacy Act (CCPA), General Data Protection Regulation (GDPR)) have been promulgated to protect people's personal information from being abused. However, app developers may unconsciously violate regulations by using certain TPLs because they may just follow the user guidelines to invoke these TPLs without knowing the internals of these TPLs. Moreover, due to a lack of security consciousness, developers may never check whether the TPLs used in their apps have a privacy policy and whether these TPLs may have security risks with respect to privacy leakage. The latest Google Play Developer Program policies (GPDP) ask developers to ensure the TPLs used in their apps are compliant with GPDP. Developers of TPLs are required to provide privacy policies and disclose data usage. Otherwise, apps may violate regulation or market requirements and be removed from the app market. To help TPLs comply with regulations and help developers correctly use TPLs, it is necessary to analyze the consistency between TPLs' data usage and privacy policies.

Previous work has proposed methods to analyze privacy issues of apps, such as analyzing the consistency between apps' privacy policy descriptions and behaviors. However, existing methods cannot identify the app's privacy issues related to the usage of TPLs inside apps because existing methods fail to identify TPLs functionalities or analyze TPL privacy policies. PoliCheck performs dynamic analysis to get traffic flow for analyzing the apps' data usage. Then, it applies data and entity dependency tree analysis to extract data usage statements from the privacy policy, and designs rules to identify the conflicts. PPChecker checks the trustworthiness of apps' privacy policies and consistency between an app's behavior and its privacy policy without considering the latest regulations.

II. Motivating Example

To check if PoliCheck [1] can be used to analyze the apps' SDKs or not, we first construct one dataset and then use PoliCheck to process them. Finally, we manually verify the correctness of the detected privacy violations.

Constructing dataset.

We randomly selected 195 apps from Google Play as the experimental dataset. Dynamic analysis is used to obtain the data flow when the app is running. Specifically, we use monkey [4] to simulate user click events and use mitmproxy [5] to obtain the traffic generated when the app is running. Then use Nguyen [6]'s method to match personal information from the traffic. This results in a set of package, pii, domain, policyurl data streams that match the input format of policheck. We ended up with 931 such flows. We again downloaded the privacy policies of these apps from Google Play and then used htmlmaster to convert these privacy policies to plain text. Finally we got the data stream of the app and its privacy policy.

Processing with PoliCheck.

We took the data stream and the privacy policy in plain text format as input and processed it using Policheck and got 393 results.

Verifying the correctness of detected violations.

Policheck outputs some log files as well as the results of the consistency processing. First, we analyzed the consistency file and found that 395 domains that clearly belonged to a third party were skipped, such as 'auction-load.unityads.unity3d.com', because Policheck was unable to recognize these domains, which could have led to some violations not being be detected.

We assessed the accuracy of Policheck's compliance judgments and discovered notable inconsistencies. Policheck incorrectly categorized 146 data flows from 50 apps as 'omitted,' meaning these flows were not matched with statements in the privacy policy. However, the sharing or collection of this information was, in fact, described within the privacy policies of these apps. These misclassifications constituted 71.1% (58 out of 202) of the total omitted categories identified by Policheck. Importantly, all this misclassified traffic was directed to third-party domains.

Root causes of the false positives and false negatives.

Through manually checking the failures one by one, we found that most of them are caused by the following issues.

(1) Apps often use forms to declare the privacy information they share with, or collect from, third parties. However, when these forms are converted into plain text privacy policies and used as input for Policheck, the NLP employed by Policheck fails to understand the relationship between these third parties and the personal information. As a result, Policheck cannot accurately extract statements regarding third-party sharing or collection from the privacy policy. This leads to a situation where the privacy policy's form correctly describes the flow of personal information, but Policheck incorrectly categorizes these data streams—which are actually compliant with the privacy policy—as non-compliant.

(2) The data streams appear as domain names of third parties, which are typically referred to in privacy policies by entity names (e.g., cdp.cloud.unity3d.com might be referred to as "Unity" within the privacy policy) [1]. Policheck must therefore perform domain-to-entity mapping. This is achieved by manually constructing a list of synonyms for each entity (e.g., "Unity" includes cdp.cloud.unity3d.com in its list of synonyms).

Policheck first determines whether the domain name to which the data is flowing is a first-party or a third-party domain. If it is a third-party, Policheck checks whether the domain name exists in the synonym list. If it does, the domain name is mapped to the corresponding entity. If it doesn't, the data flow is skipped. The synonym list is constructed manually, which inevitably leads to omissions. As a result, some domains are unrecognized and ignored by Policheck.

III. System Design

A. Data Flow Extraction

We use dynamic analysis to capture the data flow of apps, which primarily includes two parts: traffic capture and traffic analysis.

Traffic Capture: We use Monkey to generate random events that simulate user actions, allowing each app to run long enough to trigger all possible violations. At the same time, we use mitmproxy to monitor the network traffic generated during the Monkey run. We also install the certificate provided by mitmproxy on the test device to decrypt HTTPS traffic.

However, we found that during Monkey testing, one-third of the apps would display permission requests or privacy policy consent dialogs upon installation and launch. This could cause Monkey, which generates random click events, to either fail to click the 'allow' option or click the 'deny' option, potentially preventing the app from triggering some violations or causing the app to exit, interrupting the test. To address this, we added a module to consent to permission requests when the app is opened, before the Monkey test. When the app is launched, it first waits for thirty seconds to give the app enough time to load the permission request dialog. Then, using uiautomator2, it captures the clickable elements on the app interface and extracts their text. Since the text indicating consent varies widely (e.g., 'allow', 'I permit', etc.), simple string matching is not feasible. Therefore, we use an NLP model for sentiment analysis to determine if the text conveys positive sentiment similar to 'allow'. If it does, the element is clicked to bypass the permission request dialog.

In this way, we capture all HTTP/HTTPS traffic generated during the app's runtime.

Traffic Analysis: After capturing the traffic, we need to identify the privacy-sensitive information sent by the app to servers. Table 2 shows the types of privacy information we match. We use specific string matching to identify privacy information in the traffic, including methods proposed by Nguyen [6]. to match privacy information in various cases (upper/lower), hashing (e.g., MD5, SHA-1), or encoding (e.g., base64). This results in a tuple representing the data flow (app, destination domain, data type).

Additionally, we use apktool [7] to decompile the APK and examine the resulting smali code to check if the SDK's classes contain the domain and the APIs used to obtain the data type. This indicates that the SDK is responsible for collecting the privacy information and sending it to its server, resulting in another tuple (sdk, app, destination domain, data type).

IV. Experiments and Evaluation

To verify the performance of the Request Permission Popup Skip module, we conducted an experiment.

Experimental Setup. We randomly selected 30 apps that trigger permission request pop-up windows. Initially, we used Monkey to test these apps without our module and employed Frida to hook certain APIs to gather personal information, such as getLastKnownLocation. We recorded the frequency of these API calls. Next, we uninstalled and reinstalled the apps, this time incorporating our permission skip module, and again counted the number of API calls.

Experimental Results. The results of the experiment are shown in Fig. 1. It can be seen that the number of API calls for location information and ad IDs increases after adding the module. This increase occurs because the app calls the relevant APIs to access sensitive information after automatically agreeing to certain permission requests, such as those for location information. These results demonstrate that our module can effectively increase the chances of detecting hidden violations.

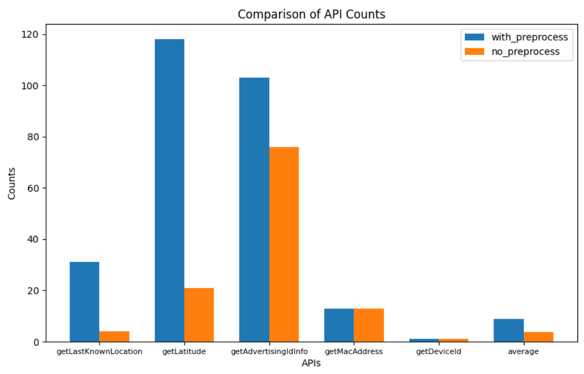

Fig. 1. Add a comparison of the number of sensitive api calls before and after the permission request popup is skipped.

V. Related Work

VI. Conclusion

References

[1] B. Andow, S. Y. Mahmud, J. Whitaker, W. Enck, B. Reaves, K. Singh, and S. Egelman, "Actions speak louder than words:{Entity-Sensitive} privacy policy and data flow analysis with {PoliCheck}," in *Proc. USENIX Security*, 2020.

[2] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie, "{PolicyLint}: Investigating internal privacy policy contradictions on google play," in *Proc. USENIX security*, 2019.

[3] K. Zhao, X. Zhan, L. Yu, S. Zhou, H. Zhou, X. Luo, H. Wang, and Y. Liu, "Demystifying privacy policy of third-party libraries in mobile apps," in *Proc. ICSE*, 2023.

[4] Google, Inc. Ui/application exerciser monkey. [Online]. Available: https://developer.android.com/tools/help/monkey.html

[5] A. Cortesi, M. Hils, T. Kriechbaumer, and contributors. (2010–) mitmproxy: A free and open source interactive HTTPS proxy. [Online]. Available: https://mitmproxy.org/

[6] N. M. B. S. Trung Tin Nguyen, Michael Backes, "Share first, ask later (or never?) studying violations of gdpr's explicit consent in android apps," in *Proc. USENIX*, 2021.

[7] A. Connolly. APKTool. [Online]. Available: https://github.com/iBotPeaches/Apktool