Demystifying Privacy Policy of Third-Party Libraries in Mobile Apps

Kaifa Zhao†, Xian Zhan‡, Le Yu†*, Shiyao Zhou†, Hao Zhou†, Xiapu Luo†, Haoyu Wang§, Yepang Liu‡
† The Hong Kong Polytechnic University, ‡ Southern University of Science and Technology,
§ Huazhong University of Science and Technology
kaifa.zhao@connect.polyu.hk, chichoxian@gmail.com, yulele08@gmail.com, shiyao.zhou@connect.polyu.hk, cshaoz@comp.polyu.edu.hk, csxluo@comp.polyu.edu.hk, haoyuwang@hust.edu.cn, liuyp1@sustech.edu.cn

***Abstract*—The privacy of personal information has received significant attention in mobile software. Although researchers have designed methods to identify the conflict between app behavior and privacy policies, little is known about the privacy compliance issues relevant to third-party libraries (TPLs). The regulators enacted articles to regulate the usage of personal information for TPLs (e.g., the CCPA requires businesses clearly notify consumers if they share consumers' data with third parties or not). However, it remains challenging to investigate the privacy compliance issues of TPLs due to three reasons: 1) Difficulties in collecting TPLs' privacy policies. In contrast to Android apps, which are distributed through markets like Google Play and must provide privacy policies, there is no unique platform for collecting privacy policies of TPLs. 2) Difficulties in analyzing TPL's user privacy access behaviors. TPLs are mainly provided in binary files, such as *jar* or *aar*, and their whole functionalities usually cannot be executed independently without host apps. 3) Difficulties in identifying consistency between TPL's functionalities and privacy policies, and host app's privacy policy and data sharing with TPLs. This requires analyzing not only the privacy policies of TPLs and host apps but also their functionalities. In this paper, we propose an automated system named ATPChecker to analyze whether Android TPLs comply with the privacy-related regulations. We construct a data set that contains a list of 458 TPLs, 247 TPL's privacy policies, 187 TPL's binary files and 641 host apps and their privacy policies. Then, we analyze the bytecode of TPLs and host apps, design natural language processing systems to analyze privacy policies, and implement an expert system to identify TPL usage-related regulation compliance. The experimental results show that 31% TPLs violate regulation requirements for providing privacy policies. Over 39.5% TPLs miss disclosing data usage in their privacy policies. Over 47% host apps share user data with TPLs and over 20% of host apps miss disclosing usage of TPLs and interactions with TPLs. Our findings remind developers to be mindful of TPL usage when developing apps or writing privacy policies to avoid violating regulations.**

***Index Terms*—Privacy policy, third-party library, Android**

I. Introduction

Nowadays, smartphones and mobile apps are playing essential roles in our daily lives [1]. 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 [2]. However, developers may omit the privacy threat in TPLs. For example, the advertisement libraries may leak users' personal information (e.g., IMEI) [3]. Some TPLs (e.g., advertising and analytic libraries) rely on personal information to provide better service [4]. 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 [5]–[7]. Besides, the permission mechanism of Android also increases the possibility of leaking user privacy via TPLs [8]–[18]. 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 [19]–[26]. In addition, the incorrect use of TPLs [26]–[28] may cause serious personal information leakage. For example, a recent report [26] discloses that popular Android apps with over 142.5 million installations leak user data to unauthorized third parties. Zhang et al. [27] disclose 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 [28], [29]. 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)) [30]–[34]) 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)* [35] asks developers to ensure the TPLs used in their apps are compliant with GPDP [36]. 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 [37], [38]. 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.

* The corresponding authors.

Previous work [27], [39]–[44] 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 [39] 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 [40] checks the trustworthiness of apps' privacy policies and consistency between an app's behavior and its privacy policy without considering the latest regulations.

Thus, there is a research gap with respect to discrepancy analysis between the privacy-related behaviors of TPLs and regulations. To this end, we propose a tool, named ATPChecker (**A**utomated **T**hird-party library **P**rivacy compliance **Checker**) to automatically identify whether the usages of in-app TPLs complies with privacy-related regulations (e.g., GDPR Article I.(47), CFR 332.6). As regulation analysis requires expert knowledge of both legal and software engineering, we invited one expert and one senior researcher to read through related regulations [30]–[33], [45]–[47] and summarize requirements for TPLs. They have conducted research on software engineering and privacy policy analysis for over seven years and two years, respectively. ATPChecker identifies the inconsistency between TPLs and regulation by analyzing TPLs' bytecode (§III-A) and privacy policies (§III-B). For example, ATPChecker discovers whether TPLs correctly provide privacy policies. ATPChecker discloses whether host app's TPL usage comply with regulations(§III-C, §III-D).

It is non-trivial to develop ATPChecker. First, it is not straightforward to get TPLs and corresponding privacy policies. In contrast to Android apps, which are distributed through markets like Google Play with their privacy policies, there is no central platform for collecting privacy policies of TPLs. To counter this issue, we first construct a data set that contains a TPL repository and a host app repository. The TPL repository contains a) a TPL list, which contains 458 TPLs crawled from AppBrain [48], b) 187 TPLs' binary files, which are in the format of *jar* or *aar*, from Maven Repository [49], and c) 247 TPLs' privacy policies which are manually collected. The details of dataset components and relations are given in §IV. The host app repository contains a) 641 distinct host apps which are collected from Google Play based on the list from AppBrain and b) the host app's privacy policies. Second, it is not easy to analyze TPLs' user privacy access behavior because TPLs are mainly provided in the format of binary files, such as *jar* and *aar*, and TPLs' whole functions may not be executed independently without being triggered by host apps. ATPChecker performs static analysis [50], [51] on TPLs and uses data flow analysis to trace TPLs' personal information usage without the need of TPLs' source code (§III-A). We exclude dynamic analysis since it cannot execute all possible paths in apps or TPLs [52]. Third, analyzing privacy policy is difficult because developers use various natural languages to describe the usage of personal information and TPLs. To analyze data and TPL usage-related statements in privacy policies, ATPChecker implements an expert system to extract abstract data usage patterns (§III-D) and investigates TPL usage compliance.

Overall, our contributions are summarized as follows:

- We propose a novel system named ATPChecker to analyze the compliance of Android TPLs. ATPChecker uses static analysis to identify TPL's user data access behaviors and host apps' data interaction with TPLs, and uses natural language processing techniques to analyze TPLs' and host apps' privacy policies. Combining the results of bytecode analysis and privacy policy analysis, ATPChecker determines whether TPLs and usage of TPLs in host apps comply with the regulation.
- To evaluate the performance of ATPChecker and facilitate further research in this area, we construct a privacy dataset that includes 187 TPLs' binary files and their privacy policies, and 641 host apps and their privacy policies.
- ATPChecker discovers over that 31% of TPLs miss providing privacy policies and 39% of TPLs' privacy policies conceal data usage. ATPChecker finds that over 20% of host apps violate the regulation requirements for clearly disclosing data interactions with TPLs.

II. Background

A. Android Third-party Libraries

Android TPLs provide abundant functions (e.g., user data analysis and advertising recommendations), which can be reused by developers in their apps to facilitate development progress. However, TPLs may introduce data leakage issues. Due to Android's permission mechanism, TPLs share the same privileges with host apps [19]–[21]. TPLs may abuse permissions to access users' personally identifiable information without users' consent [22], [23], which results in personal information leakage [8]–[18], [53].

Android third-party libraries are generally available as binary files, such as **.jar* or **.aar*. Developers mainly import binary files into their projects to use the TPLs without inspecting the TPLs' data usage, which results in the violation of regulation requirements. Furthermore, it is very time-consuming for developers to understand TPLs' data usage behavior. Thus, developers may not be able to clearly describe the TPLs' data usage in their privacy policies or may just provide links to TPLs' privacy policies [40], thus making their apps and privacy policies violate regulations.

B. Privacy Policy

Privacy policies describe how the data controllers use, disclose, store, manage and share users' personal information [30]–[33]. Regulations require privacy policies to describe TPLs' data access behaviors clearly. Besides, the coverage of personal information [32] is not limited to identical information, such as ID, but also includes any information that can be used to identify or infer a specific person [32]. To comply with

regulation requirements, privacy policies should clearly claim the following information [54]: 1) personal information, 2) the software's contact information, 3) the purpose of collecting personal information, 4) types of data shared with third parties and 5) the rights that users have.

C. Regulation Requirements for TPLs and Usage of TPLs

Regulations have enacted Articles to normalize personal information usage by software. CALIFORNIA CONSUMER PRIVACY ACT (CCPA) [33] ARTICLE 1798.120 claims that "*a business should notify consumers if they sell consumers' personal information to third parties*". CYBERSECURITY PRACTICES GUIDELINES–SECURITY GUIDELINES FOR USING SOFTWARE DEVELOPMENT KIT (SDK) FOR MOBILE INTERNET APPLICATIONS (APP) (SGSDK) [31], has been enacted to standardize the management of third-party libraries. For example, SGSDK ARTICLE 5.1 D) claims "*The SDK discloses the scope, purpose, and rules of the SDK's processing of personal information to the App in a clear, understandable and reasonable manner. The actual behavior of the SDK's collection and use of personal information should be consistent with the statement in the public document.*" Similar requirements are also mentioned in GDPR [32] ARTICLE 14.2 and CCPA [33] ARTICLE 4. INFORMATION SECURITY TECHNOLOGY-PERSONAL INFORMATION (PI) SECURITY SPECIFICATION (PISS) [30] Article 9.7 specifies the requirements for third parties. Regulations [55] also enact Articles to regulate usage of TPLs. For example, regulations [30]–[32], [34], [55] require apps to disclose the purpose, method and scope of usage of TPLs. In PISS, GDPR and CCPA, TPLs are regarded as data controllers and are required to expose their data usage.

III. METHODOLOGY

This section introduces how ATPChecker identifies TPLs' data usage (§III-A) and how to identify the consistency of TPLs' data usage with the statements in privacy policies (§III-B). We will also describe how to determine whether host apps describe TPLs usage correctly in their privacy policies (§III-D) and whether host apps use TPLs correctly (§III-C). Fig. 1 shows the framework of ATPChecker.

TABLE I: Tracked data types in static analysis.

Data Type
Ad ID, username, password, name, location, contact, phone number, email address, IMEI, Wi-Fi, MAC address, GSF ID, Android ID, serial number, SIM serial number

A. Identify personal information usage in third-party libraries

ATPChecker analyzes TPL's personal information (PI) usages by performing static analysis based on *soot* [50] and does not require TPLs' source code. ATPChecker conducts data flow analysis, specifically variable use-define analysis [56], to locate data of interest (DOI) (Tab. I), which are mainly related to user identification information summarized from regulations. ATPChecker uses function call graphs (FCG) to trace PI flow among methods. However, our analysis discovers that the soot cannot effectively discover the TPL's main functions and entry points to construct a valid FCG [51]. Thus, we propose the following methods to optimize the FCG construction and improve the data flow analysis precision.

TPL FCG construction: ATPChecker iterates over each class and method. Since apks mainly use TPLs by invoking their *public* methods, ATPChecker extends TPLs *public* methods to entry points set to optimize FCG construction.

TPL DOI identification: We summarize PI from regulations [30]–[33] as listed in Tab. I. Then, we manually crawl PI-related APIs [57]. Based on those APIs, ATPChecker iterates over all statements and locates DOI with target variables. ATPChecker performs inter and intra-procedural data flow analysis to identify all PI-related statements. For interested data types without official API, such as email and password,

Algorithm 1: Data usage analysis

Input: method m_t, statement s_t, variable v_t
Output: data usage FD of variable v_t

```
BackwardAnalysis(m_t, s_t, v_t)
IntraMethodVarAnalysis(m_t, s_t, v_t)
Def IntraMethodVarAnalysis(m, stmt, var):
    uses = find statements that use var in method m
    for u in uses do
        store(var, u) in DF
        ForwardAnalysis(m, u, var)

Def BackwardAnalysis(m_t, s_t, v_t):
    defs = getDefsOfAt(v_t, s_t, cfg)
    for d in defs do
        src_var = get the signature of v_t in d
        caller_list = find methods that invoke m_t
        for m in caller_list do
            src_stmt = statements that invoke m_t in m
            src_v = variables in src_stmt that correspond
              to v_t
            if src_v is Variable then
                BackwardAnalysis(src_v, src_stmt, src_m)
            else if src_v is Constant then
                store (v_t, src_stmt) in DF

Def ForwardAnalysis(m, stmt, var):
    tar_stmt = find all statements that contain stmt
    for s in tar_stmt do
        store (var, s) in DF
        if s contain invoke statements then
            tar_m = method that invokes stmt in s
            tar_v = variable in tar_m that correspond to
              var
            ForwardAnalysis(tar_method, s, tar_v)
```

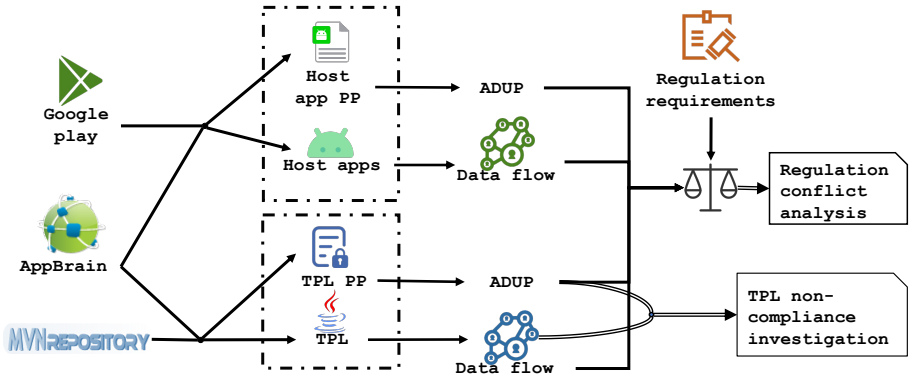

Fig. 1: Framework of ATPChecker.

we use keywords matching method to identify potential data leakage. Specifically, we user verb, i.e., "get" and "request" plus keywords, i.e., "email" and "contact", combination to mach those interested data.

Algorithm 1 sketches the TPL data usage analysis method. ATPChecker iterates over all statements in TPL to locate the target variables (var) that represent personal information listed in Tab. I. Then, ATPChecker conducts both inter-procedure and intra-procedure analysis to get the data flow. For inter-procedures analysis, ATPChecker backward analyzes statement s_t in method m_t with the DOI v_t (line 1). The backward analysis locates the definition statements ($defs$) in m_t that relates to DOI v_t (line 11). For statements in $defs$, ATPChecker iterates $defs$ and locates the variable signature of v_t in d (line 12-13). ATPChecker iterates FCG of target TPL and finds the methods ($caller_{list}$) that invoke m (line 14). For each caller m, ATPChecker locates the statements (src_{stmt}) in m that invoke m_t. ATPChecker finds the variable (src_v) in src_{stmt} that corresponds to DOI v_t (line 17). If src_v is a variable, ATPChecker continues backward analysis, starts with src_{stmt} in src_m and focuses analysis on variable src_v (line 18-19); if src_v is a constant, ATPChecker finds one piece of code that describes the data usage of var (line 21).

To locate the v_t's data usage in m_t, ATPChecker performs *intra-method analysis*. Specifically, ATPChecker locates all statements ($uses$) that uses var in method m (line 5). For each statement u in $uses$, ATPChecker stores the usage of variable var (line 7) because the target variable has been used in the statement u. Then, ATPChecker conducts forward analysis (line 8) to identify whether var is used further. For each statement s in tar_{stmt}, ATPChecker stores the data flow describing the data usage. If s contains invoke statements, ATPChecker locates the method tar_m that calls $stmt$ (line 28), builds the variable tar_v corresponding to target variable v_t and continues the forward analysis (line 30).

B. Identify data usage statements in TPL's privacy policy

TPL privacy policy analysis aims to identify the data usage statements. The pipeline of our privacy policy analysis consists of three steps: 1) preprocessing of privacy policy, 2) named entity extraction, and 3) data usage action construction. Finally, each privacy policy is abstracted into a set of **a**bstract **d**ata **u**sage **p**atterns (ADUP) that describes *who* will (*not*) *access what kinds of data* with *whom*. Each ADUP is in form of {data_entity, action, {data_type}, {data_recipient}, {neg}}. For example, *"We share your personal information with our service providers," (in Adad's privacy policy)* is abstracted as {*app, share,* {*personal information*}, {*service provider*}, *false*}.

1) Preprocessing privacy policy: To download the apps' privacy policies from Google Play [58], we use the *Selenium Webdriver* [59] to open websites of privacy policies. Then, we use *html2text* [60] to transform downloaded privacy policies into plain text. However, the transformed plain text contains many special characters, e.g. $*$ and |, which will lead to unnatural segmentation of sentences. ATPChecker first uses patterns [61] to eliminate special characters. Besides, privacy policies use enumeration items [41] to detail related data items, which may lead to a whole sentence being segmented by item index and further cause losing information. ATPChecker merges the items by searching for sentences ending in a colon, and the next sentence starts with list items, such as bullets, roman numerals, and formatted numbers.

2) Data access extraction: With preprocessed privacy policy sentences, ATPChecker analyzes each sentence to identify data access descriptions. Since the interrogative sentences are commonly used in privacy policies but do not describe the types of accessed personal information, we omit them by checking the interrogative words (i.e., who, where, when, why, whether, what, and how). For example, the sentence "*What personal information do we collect*" is omitted. Then, we construct the part-of-speech (POS) tagging, and dependency parsing tree (DPS) for each sentence. We will omit sentences without verb words because those sentences will not claim data access behaviors. For example, "*Do-Not-Track Signals and Similar Mechanisms*" in *Facebook Open Source Privacy Policy* only claims the title of the paragraph. For the remaining sentences, we follow the following steps to extract data action, data entity, and data actor.

Identification of data action words. We first identify whether the verb words are in the data usage word list (Tab. III). We stem the word using *nltk* [62] to deal with English verb tenses. We also identify the modifier of the target verb because the privacy policies also claim the data they will not use or

TABLE II: Personal information and corresponding APIs.

PI	Class Name	Method Name
Ad ID	AdvertisingIdClient	getadvertisingidInfo
AD ID	AdvertisingIdClient	getInfo
Bluetooch address	android.bluetooth.BluetoothAdapter	getAddress
Camera	android.hardware.Camera	setPreviewDisplay
Cell Location	android.telephony.TelephonyManager	getCellLocation
Contact	android.provider.ContactsContract	PhoneLookup
GPS Location	android.location.Location	getLatitude
	android.location.Location	getLongitude
	android.location.LocationManager	getLastKnownLocation
	android.location.LocationManager	requestLocationUpdates
	android.location.LocationManager	getLongitude
Sim Number	android.telephony.TelephonyManager	getSimSerialNumber
IMEI	android.telephony.TelephonyManager	getDeviceId
IMSE	android.telephony.TelephonyManager	getSubscriberId
TimeZone	java.util.Calendar	getTimeZone
MAC address	android.net.wifi.WifiInfo	getMacAddress
Password	android.accounts.AccountManager	getPassword
Phone Number	android.telephony.TelephonyManager	getLine1Number
SMS	android.telephony.SmsManager	sendTextMessage
SSID	android.net.wifi.WifiInfo	getSSID
User Credential	android.accounts.AccountManager	getAccounts
UserName	android.os.UserManager	getUserName

collect. To achieve this, we analyze the dependency parsing tree and recognize the words whose relation with the target verb is "*advmod*" (adverbial modifier) or "*mmod*" (modal verb modifier). If the words or their tags are negative (i.e., "neg"), we regard the data usage action as negative one. To this end, ATPChecker identifies data sharing or collection operations.

Identification of data entity. We identify objects of identified collecting and sharing words (target verb) according to DPS. ATPChecker first merges the noun phrase (NP) tag into one noun tag according to POS tagging to simplify sentence structure. For example, consider the following sentence in the privacy policy of the app *video.reface.app* "*With the use of Google Analytics, we collect such data as IP-Address, your device model, screen resolution and operation system, session duration, your location*". As only one noun or noun phrase will be identified as the object of the target verb [63], ATPChecker locates all coordinating nouns for the object. Specifically, ATPChecker iterates the constituency parsing tree, which is calculated by two stages conditional random field (CRF) model [64], to find the sub-trees whose *a) root label are common noun (NN tag) or Noun Phrase (NP)* or *b) coordinating conjunction (CC tag) or punctuation exist in the sub-tree*. The condition *a)* denotes that the leaves in the tree can be merged into one noun, and *b)* indicates there may exist enumeration items or coordinating nouns. If we omit the relations, ATPChecker cannot fully extract all data entities, i.e., false-positive occurs. To this end, we obtain all candidate data objects in each sentence.

Identification of data actor. To identify the data actor, ATPChecker analyzes DPT and locates the words whose relation with sharing and collection verbs is nominal subject ("*nsubj*") or direct object ("*dobj*"). Suppose the sentence or the object is a prepositional complement or clausal complement of a preposition ("*ccomp*"). We iterate the DPT to find the coordinating verb and then locate the nominal subject of the target verb. The nominal subject and the verb will be added to the actor list and the sharing/collection verb list, respectively. We also consider the data actor as users separately and the situation when the real data actor is the user rather than the supplier. For example, "*we will not collect the personal information you shared with us*". The data actor should be "you" while the data recipient is "us".

Post-process To eliminate false detection, e.g., conditional clauses, we design rules to avoid false data usage extractions. For sentences that start with condition or assumption hints, such as *'if'* (*"if you do not provide your personal information"*), we will not identify the data usage pattern inside.

C. Extract host apps' interaction with TPLs

To check whether host apps' privacy policies are consistent with their TPL usage, ATPChecker conducts static analysis to identify the host apps' interaction with TPLs. ATPChecker discloses what kinds of PI data (Tab. I) are shared with TPLs. ATPChecker uses flowdroid [51] to construct FCG of apps. Since flowdroid optimizes Android app analysis process [51], ATPChecker directly uses the FCG constructed from flowdroid. ATPChecker performs the data flow analysis (§III-A) to find the PI related data flow. To identify TPLs, ATPChecker identifies whether the statement contains the TPLs' name or package name or not. After obtaining TPLs and PI data, we identify whether the method exists in the data flows. Besides, we optimize the signature of method invocation (*SMI*) using the invoking statement. ATPChecker matches the *SMI* in both PIs' data flow and TPLs' data flow. Once the *SMI* is matched between TPL_i and PI_j, ATPChecker regards the host app sharing the PI_j with the TPL_i. If the *SMI* only exits in PI_j's data flow, ATPChecker regards that the host app collects PI_j without sharing it with TPLs.

D. Identify TPLs usage statements in host apps' privacy policy

To identify the TPL usage statements in host apps, we extract two descriptions from the privacy policy: a) the *data sharing* related statements and b) TPL-related descriptions. The *data sharing* related statements claim how the app discloses users' personal information under which situations. For example, the app (*flipboard.boxer.app*) claims that "*We share personal information with vendors and service providers that*

TABLE III: Data sharing and collecting words.

Action Type	Keywords
Collect	access, check, collect, gather, know, obtain, receive, save, store, use
Sharing	accumulate, afford, aggregate, associate, cache, combine, convert, connect, deliver, disclose, distribute, disseminate, exchange, gather, get, give, keep, lease, obtain, offer, post, possess, proxy, provide, protect against, receive, rent, report, request, save, seek, sell, share, send, track, trade, transport, transfer, transmit

help us offer and improve our service". The statement indicates that *flipboard.boxer.app* will share users' *personal information* with vendors and service providers. If ATPChecker also identifies the app's data flow sharing users' data with TPLs tagged as vendors or service providers, ATPChecker regards the app's privacy policy as consistent with its behavior.

After preprocessing privacy policies (§III-B1), we use NLP methods [63] to get the tokenization, POS tagging and DPS for each sentence. For sentences with verbs, we check whether the verb is in *sharing* word list (Tab. III). After obtaining the data sharing verb, we identify the data recipient to determine which TPL the data are shared with. Specifically, we iterate the words in the sentence. If the word is the object of a preposition (*pobj*) or an indirect object (*iobj*) of the verb, we identify the words as candidate data recipients, i.e., TPLs. For enumeration patterns, once we identify the sentence that a) starts with a noun or noun phrase (denoted as tar_{TPL}) that is involved in our TPL list, v) the noun ends with colon following only noun or noun phrases, we regard the noun (phrases) following colon as the shared data with tar_{TPL}.

E. TPL privacy compliance investigation

ATPChecker investigates TPL privacy compliance by performing *normativeness analysis*, *privacy policies legality analysis*, and *privacy non-compliance transmissibility analysis*.

Normativeness analysis identifies whether TPL provides privacy policies. According to GDPR Article 12(1) and Article 13(1), TPLs, who act as the data controller, or third party of personal data, should infer users about data access behaviors in a concise, transparent, intelligible and easily accessible form.

Legality analysis identifies conflicts between TPLs' behavior and privacy policy statements. Specifically, ATPChecker identifies whether TPLs' data access actions are clearly claimed in their privacy policies by combining the data flow results (§III-A) and PP ADUP (§III-B). If ATPChecker identifies data usage in TPL data flow but the results are not mentioned in $data_type$s of ADUP set, the TPL will be regarded as violating legal requirements.

Privacy non-compliance transmissibility analysis checks to what extent the TPLs (TPL_v), whose privacy policies violate the regulation requirements, affect other TPLs or apps. ATPChecker summarizes artifacts that use TPL_v from *Maven Repository* to investigate the impact of TPL's privacy non-compliance. Specifically, ATPChecker crawls *Usages* information in *Maven Repository* to count the number of artifacts that use TPL_v.

F. Regulation issue analysis

Regulation conflict analysis discovers whether the usage of TPLs in host apps comply regulation requirements. Two kinds of conflict will be identified:

• ATPChecker identifies whether usage of TPLs from data flow analysis (§III-C) is clearly claimed in their privacy policy statements (§III-D). ATPChecker investigates whether the TPL package names in data flow analysis match the $data\_entity$ or $data\_recipient$ in ADUP of host app privacy policy results.

TABLE IV: List of TPLs without Privacy Policies.

TPL Categories	TPL Name	Reasons
Ad Network	Fractional Media	NOS
	YuMe	NOS
Social library	Smack API	NPOS
	Twitter4j	NPOS
Development Tool	Android In-App Billing Library	GNP
	Apache Commons Codec	NPOS
	Apache Commons I/O	NPOS
	Apache Commons Lang	NPOS
	Apache Commons Logging	NPOS
	Apache Http Auth	NOS
	Apache HttpMime API	NOS
	Apache James Mime4j	NPOS
	Apache Thrift	NPOS
	AChartEngine	GNP
	FasterXML Jackson	GNP
	Android ViewBadger	GNP
	Kin	GNP
	Material App Rating	GNP
	OpenStreetMap tools for Android	GNP
	Android GIF Drawabl	GNP
	Crouton	GNP
	Google GData client	GNP
	Google Guava	GNP
	Google gson	GNP
	HttpClient for Android	GNP
	JSON.simple	GNP
	greenDAO	NPOS
	libgdx	NPOS
	Material DateTime Picker	GNP

NOS: No Official webSite; NPOS: No Privacy policies on Official webSite; GNP: Github project without Privacy policies

• ATPChecker discloses whether the host apps clearly state their data interaction with TPLs. ATPChecker identifies whether the apps' TPL interaction behavior (§III-C) is clearly stated in host apps' privacy policies (§III-D).

IV. Evaluation

We evaluate the performance of ATPChecker by answering the following research questions:

RQ1: Normativeness Analysis of TPLs. *How many TPLs provide privacy policy documents?*

RQ2: Legality Analysis of TPLs. *Do TPLs' privacy policies meet regulation requirements, and do TPLs' privacy policies correctly claim data usage actions?*

RQ3: Host apps behavior analysis. *Do host apps conduct privacy data interaction with TPLs?*

RQ4: Legality of host app's privacy policies. *Do host app's privacy policies comply with requirements for disclosure of TPL usage?*

A. RQ1: Normativeness Analysis of TPLs.

Experiment Setup. This research question evaluates all Android TPLs listed in AppBrain [48] including three types: Ad networks, social libraries, and development tools. The list gives the TPLs that are widely used by top 500 installed apps in Google Play. The data set contains 458 TPLs that include 141 ad networks, 25 social libraries, and 292 development tools. Then, we gather the privacy policies of those TPLs by visiting the homepage provided by each TPL information page on AppBrain. We manually crawl information of TPLs whose

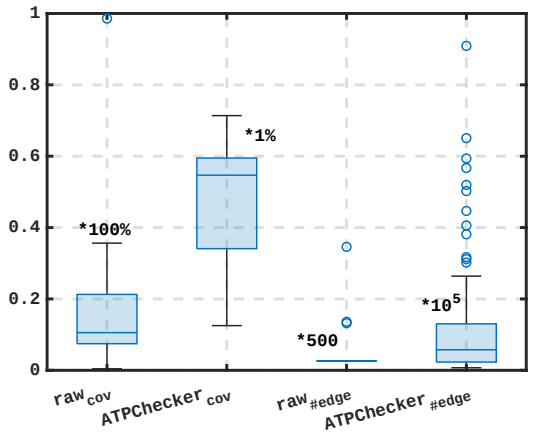

Fig. 2: Evaluation of function call graph construction.

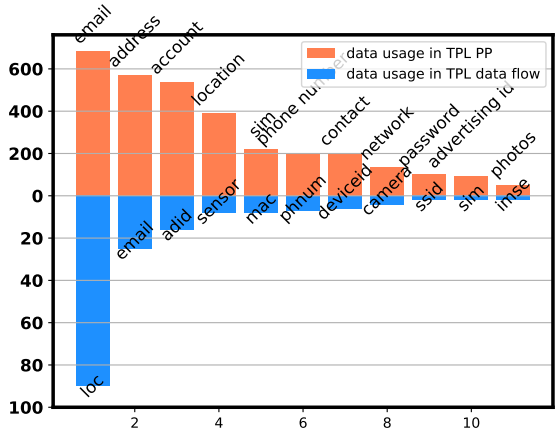

Fig. 3: Popularity of data types that are used in TPLs' data flow and PP.

homepages are not provided. For TPLs in Github or Bitbucket, we visit their public repositories and check whether privacy policies are provided. When visiting the homepage of TPLs, we search the homepage with keywords, such as *privacy, policy, legal* and *policies*, to find potential privacy policy links on the homepage. Besides, we also manually check whether policy links are given in the typical layout [65], such as the bottom of the website.

Results. According to regulation requirements (§II-C), data controllers should clearly disclose their data usage in privacy policies. This research question discloses whether TPLs satisfy the requirements of regulations, i.e., do TPLs correctly provide privacy policies. With the TPL list and privacy policies, we discover that 21 of 141 Ad network TPLs, 10 of 25 social libraries, and 180 of 292 development tools TPLs, which account for 31%, do not provide privacy policy websites. We find that some TPLs are created by individual developers and released on *GitHub* [66]. We manually crawl their privacy policies for *GitHub* repositories. Tab. IV summarizes partial TPL names and the reasons that the TPLs do not provide privacy policies. Tab. IV demonstrates that 16 of 22 TPLs that do not provide privacy policies are published as Github repositories (GNP). Five TPLs without privacy policies miss official websites (NOS), and the left does not provide privacy policy documents on their websites (NPOS). For development tool TPLs, we also observe that 66 TPLs from *Google Inc.* share the same privacy policy [67] that may lead to over-claiming the personal information usage for specific TPLs. For nine TPLs from *Apache*, six TPLs (6/9) do not provide privacy policies and two TPLs' (2/9) websites are Not Found.

Answer to RQ1: ATPChecker reveals that 31% TPLs do not provide privacy policies. Over 14% TPLs from the same company provide one general privacy policy.

B. RQ2: Legality Analysis of TPLs.

Experiment Setup. We collect the resources of TPLs from Maven Repository [49] based on the TPL list from AppBrain (§IV-A). We manually crawl TPL binary files from Maven Repository by searching the TPL's name in Appbrain's list and finally get 187 different TPLs including 87 "aar" and 100 "jar" files. More specifically, we get 45 ad networks, 132 development tool libraries and 10 social libraries.

Results. ATPChecker discloses whether the TPLs' privacy policies satisfy the regulation requirements by comparing TPLs' binary files analysis results (§III-A) and privacy policies analysis results (§ III-B). ATPChecker analyzes TPLs' privacy policies and obtains the ADUP of personal information usage-related statements. ATPChecker identifies personal information in the TPL through static analysis (§III-A). Finally, ATPChecker matches the consistency between AUDP and personal information.

TPL privacy policies analysis. ATPChecker analyzes TPLs' privacy policy documents. We analyze 52,523 sentences with sharing and collection (SAC) words. Among sentences with SAC words, 15,270 sentences start with 5W1H (who, why, when, whether, what, how) words that are considered as data usage actions. Besides, 5,205 sentences only vaguely state that the TPL will collect personal information without specific data types. For example, $com.xiledsystems$ only claims *"We collect your personal information in order to provide and continually improve our products and services,"* but no specific personal information is given. In data usage-related actions, ATPChecker analyzes the personal information that is mostly used by TPLs. Fig. 3 shows the times that are mentioned in TPLs' privacy policies. It can be observed that `contact` is the most used personal information. This may be caused by the fact that, by getting contact, the TPLs can easily expand and prompt their services. Account, address and email are the second most popular data mentioned in TPLs' privacy policies. Among the statements that claim to collect accounts, addresses, and email, 21.13% sentences claim that they collect related data for contacting users. For example, one of them claims that *"we will use your email, phone number, or other contact information you provide us by written or oral means for contacting you and providing you with the services and information that you request"*.

TPL data usage analysis. We first evaluate the effectiveness of ATPChecker in improving the FCG construction performance.

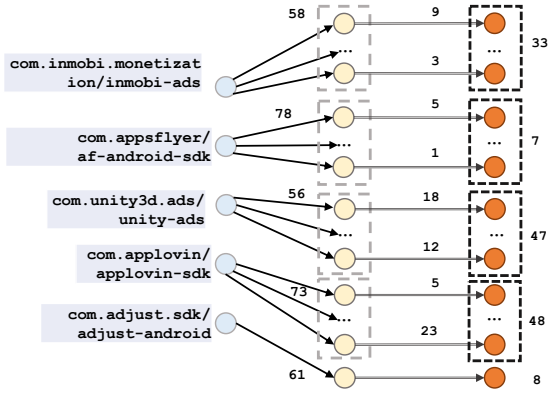

Fig. 4: TPL Privacy Policy Non-compliance Propagation. The first column is raw TPLs identified by ATPChecker that potentially violate regulation requirements. The second column is TPLs affected by raw TPLs and the number above the arrow is the number of TPLs infected by raw TPLs. The third column is TPLs affected by the TPLs in the second column.

We analyze TPLs' FCG quality through comparing the a) FCG coverage and b) the number of edges between the raw FCG built by soot and our optimized FCG. FCG coverage is calculated by:

$$cov_{FCG} = \frac{\#Nodes_{FCG}}{\#Nodes_{TPL}}, \quad (1)$$

where $\#Nodes_{FCG}$ is the number of nodes (methods) in FCG and $\#Nodes_{TPL}$ is the total number of TPLs' methods. Fig. 2 illustrates the box plots of ATPChecker's performance in optimizing FCG construction, and the number at the top of boxes denotes measuring units of the data for better illustration. Fig. 2 demonstrates that ATPChecker's FCG coverage achieves an average performance of 55%, while the FCG coverage of raw FCG only achieves an average of 0.34%, which means it nearly fails to construct the FCG of TPLs. This could be caused by the fact that soot can not identify comprehensive entry points of TPLs to construct the FCG and thus cause many nodes and edges are missed. $raw_{\#edge}$ denotes the raw FCG only contains the 159 average edges, while the $ATPChecker_{\#edge}$ shows the improved FCG can extract 12,960 edges. The results indicate that the soot nearly fails to analyze the FCG of TPLs because of the misidentification of TPL's entry points for FCG construction, and ATPChecker optimizes construction performance and improves the FCG quality.

ATPChecker analyzes the data usage of TPLs to identify the consistency with their privacy policy statements. ATPChecker analyzes the data flow of TPLs and discovers that 38 out of 187 TPLs access users' personal information. ATPChecker identifies 176 data access traces. Among data usage behaviors, 90 out of 176 traces access location information. Fig. 3 shows the data frequencies that are mostly used in TPLs' data flow (shown in orange color). It can also be observed that TPLs' privacy policies tend to disclose non-identical information while their actions tend to access identical information, such as location.

ATPChecker checks the compliance of TPLs by combing the results of privacy policies analysis and data flow analysis. Specifically, ATPChecker identifies how many TPLs comply with regulations by checking whether the PI in data flow is also mentioned in ADUP. ATPChecker identifies 38 TPLs access users' data. Among those 38 TPLs, ATPChecker discovers that 15 out of 38 (39.5%) TPLs violate the regulation requirements, i.e., the TPL collects at least one users' data without clearly disclosing them in the TPL's privacy policy.

Investigation on perniciousness of TPLs' non-compliant behavior. After identifying TPLs that do not satisfy the regulation requirements, we also study the effect of TPLs' non-compliant behavior on a large scale. TPLs may integrate other TPLs to enhance their functionalities or facilitate usability [2]. Thus, once one TPL (TPL_v) violates the regulation, other artifacts using TPL_v may spread the threats and make their privacy policies violate regulation requirements. To expose the impacts, ATPChecker analyzes the dependencies among TPLs. Specifically, ATPChecker crawls the TPLs (TPL_U) that use TPL_v, which are analyzed in §III-A-III-B, from Maven Repository. Then starting from TPL_U, ATPChecker crawls the list of artifacts that use TPL_U. In this way, ATPChecker investigates the propagation of TPL_v's threats under two times integration and visualizes it in Fig. 4. ATPChecker starts the analysis with 15 unconventional TPLs. Fig. 4 shows the effect of five TPLs whose privacy policies have inconsistency issues. It can be observed that after one round of propagation, even five TPLs will affect 321 TPLs (15 TPLs affects 434 TPLs). After two rounds of propagation, the threats even spread to extra 143 TPLs (15 to 168). Fig. 4 also illustrates that both popular and minority TPLs can have a significant impact on the propagation of privacy non-compliance. Popular TPLs are widely used by other TPLs, making them highly infective, while minority TPLs can also affect a large number of TPLs. After two rounds of propagation, the number of TPLs they affect increases exponentially. This observation indicates that developers should pay attention to the usage data of TPLs, especially the functions that are related to privacy.

Answer to RQ2: ATPChecker identifies that over 39.5% TPLs miss disclosing their data usage in privacy policy documents. ATPChecker investigates that the effect of the privacy policies with non-compliance issues spreads widely.

C. RQ3: Analysis of host apps' interaction with TPLs

Experiment Setup. We collect the host apps of TPLs from AppBrain to investigate host apps' interaction with TPLs. We gather the host app list using the mapping relations between host apps and used in-app TPLs provided in AppBrain. For TPL's host apps, we can only access 10 apps which are mostly downloaded in Google Play. Finally, we gather a total of 641 distinct apps because some apps may use multiple TPLs and one TPL can be used by different apps.

Results. ATPChecker performs data flow analysis to investigate whether host apps conduct personal information interaction with TPLs and what kinds of PI are shared with

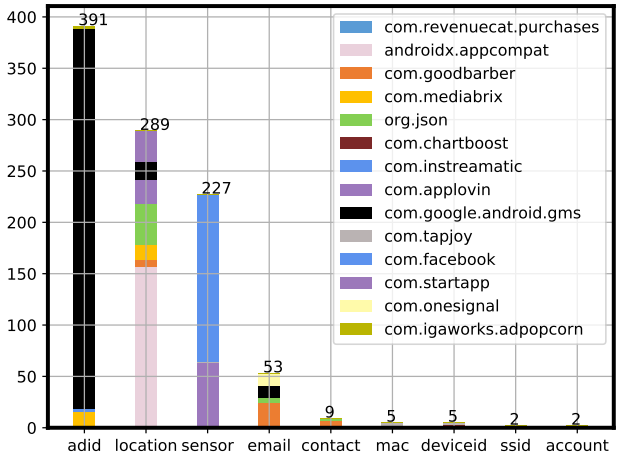

Fig. 5: Statistics of host apps' data interaction with TPLs.

TPLs. ATPChecker analyzes 641 distinct apps. Due to the restraint of computational resources and the limitations of flowdroid, we successfully analyzed 459 apps. 174 out of 641 apps are in the form of ".xapk" [68] which cannot be analyzed by flowdroid, and the others cannot be analyzed because some apps apply protection methods and some apps are too large which exhaust our computation resources [69]. ATPChecker identifies that over 47.9% (220/459) host apps share PI with TPLs. ATPChecker measures the suspicious behavior by counting the times of PI-related invocations in apps' data flow and traces 973 times data-sharing behaviors with TPLs and finds that over 40.2% traces share users' *advertising ID* with TPLs such as *"com.google.android.gms"*. Fig. 5 shows the nine types of data shared by host apps with TPLs. Fig. 5 shows that the second popular data the host app need to share with TPLs is location. ATPChecker also observes that *"com.google.android.gms"* is the TPL that accesses the most PI from host apps. Although *"com.google.android.gms"* provides abundant functionalities, such as Gmail and music, TPLs should avoid collecting information that can be used to easily identify an individual and provide privacy policies to state the purpose of data access. TPLs should not ask for data which is unnecessary for their services [30]–[33].

Answer to RQ3: ATPChecker identifies that over 47.9% of host apps share personal information with TPLs. Among shared PI, advertising ID accounts for over 40.2% of traces in identified user data access actions. ATPChecker also discovers that while well-known TPLs provide rich functionalities, they also request personal information.

D. RQ4: Analysis of host apps' privacy policy.

Experiment Setup. We collect the privacy policies of host apps to determine whether the apps' data usage behavior is consistent with the privacy policies. For each host app in our dataset, we get its privacy policy by crawling its homepage from Google Play. To evaluate the performance of ATPChecker, we collect extra 254 apps, which can be successfully analyzed, and their privacy policies. To annotate these privacy policies, one expert and senior researcher, who have conducted research on privacy policy and software engineering for over seven years and two years respectively, manually label the sentences that include data usage, TPL usage, and data interaction with TPLs. The annotation process is conducted in an in-house manner [70], [71] that guarantees the quality and agreement of the labels.

Metrics. We conducted an evaluation of ATPChecker's performance in identifying the accuracy of host apps' claims regarding i) the usage of TPLs in their privacy policies, and ii) TPLs' data usage in host apps. Considering ATPChecker's purpose in identifying compliance, we define metrics as follows. For i), which focuses on whether ATPChecker can correctly identifies the host apps' usage and declaration of TPL usage in their privacy policies:

True Positive: The host app uses the TPL without clarifying it in the host apps' privacy policies (non-compliance). ATPChecker identifies the host app's usage of TPL and cannot identify the claim of the TPL's usage in the host app's privacy policy. It is worthy noticing that if ATPChecker cannot identify the usage of TPL in host app's Code, ATPChecker cannot identify the positive behavior, i.e., the host app uses the TPL without claiming the usage of TPL in app's privacy policy.

True Negative: The host app declares the usage of TPLs in their privacy policies (compliance). i) ATPChecker identifies the host app's TPL usage in the host app's binary files and identify the claim of TPL usage in the host app's privacy policy. ii) ATPChecker identifies the claim of TPL usage in the host app's privacy policy.

False Positive: The host app declares the usage of TPLs in their privacy policies (compliance). i) ATPChecker identifies the host app's usage of TPL and cannot identify the claim of the TPL's usage in the host app's privacy policy. ii) ATPChecker cannot identify the usage of TPL in app's code and cannot identify the claim of TPL usage in app's privacy policy.

False Negative: The host app does not claim the usage of TPL in the app's privacy policy. i) ATPChecker identifies TPL usage in host app's binary files and also identify the claim of TPL usage in the app's privacy policy. ii) ATPChecker identifies the claim of TPL usage in the app's privacy policy.

For the second aspect (ii), which assesses whether ATPChecker can correctly identify the host apps' declaration of TPLs' data usage in the app's privacy policies:

True Positive: The host app shares user data with TPL without clarifying it in the host app's privacy policies (non-compliance). ATPChecker identifies the host app's data-sharing behavior in host app's code and cannot identify the claim of data-sharing behavior in the app's privacy policy. Please note that if ATPChecker cannot identify the host app's data-sharing behavior with TPL in its Code, ATPChecker cannot identify the positive behavior, i.e., the host app shares user data with TPL without clarifying it in the host app's privacy policies.

True Negative: The host app shares user data with TPL and clarifies it in the host app's privacy policies (compliance). ATPChecker identifies the host app's data-sharing behavior

TABLE V: Evaluation of ATPChecker for identifying host apps' compliance.

	TPL List				TPL Data			
Trace Level	TP	TN	FP	FN	TP	TN	FP	FN
	87	219	0	22	143	123	/	39
	A	P	R	F1	A	P	R	F1
	93.29%	1	79.82%	88.78%	/	/	78.57%	/
App Level	TP	TN	FP	FN	TP	TN	FP	FN
	56	118	0	24	62	57	/	22
	A	P	R	F1	A	P	R	F1
	87.88%	1	70%	82.35%	/	/	73.81%	/

and identifies the claim of the data-sharing behavior in the app's privacy policy. Please note that if ATPChecker cannot identify the host app's data-sharing behavior with TPL in its Code, ATPChecker cannot identify the positive behavior, i.e., the host app shares user data with TPL and clarifies it in the host app's privacy policies.

False Positive: The host app demonstrates compliance by sharing user data with TPLs and explicitly stating this behavior in the host app's privacy policies. i) ATPChecker successfully identifies the host app's data-sharing behavior within the host app's code; however, it is unable to identify the explicit claim of data-sharing behavior in the app's privacy policy. ii) ATPChecker does not have the capability to identify the app's data-sharing behavior as described in the app's privacy policy. Furthermore, it is important to note that due to the unavailability of ground truth regarding the host apps' data-sharing behavior and their claims in the privacy policies, the assessment of false positives by ATPChecker cannot be evaluated in the current version.

False Negative: The host app shares user data with TPL without clarifying it in the host app's privacy policies or the app does not claims the data sharing with TPLs. i) ATPChecker identifies the host app's data-sharing behavior and identifies the claim of the data-sharing behavior in the app's privacy policy. ii) ATPChecker only identifies the the claim of the data-sharing behavior in the app's privacy policy

Results. Table V presents the performance of ATPChecker in identifying two aspects: i) whether host apps accurately declare the usage of TPLs in their privacy policies, referred to as "TPL list", and ii) whether host apps accurately declare the data usage of TPLs in their privacy policies, denoted as "TPL data". The metrics used for evaluation include Accuracy (A), Precision (P), Recall (R), and F1-score. The evaluation assesses ATPChecker's performance from two perspectives: Trace Level, which measures the number of metrics based on the behaviors identified by ATPChecker in each app, and App Level, which counts the metrics based on the number of apps. For instance, suppose ATPChecker identifies that an app conceals the usage of two TPLs in its privacy policies and the app indeed conceals the usage accurately. In this case, we record TP (True Positive) as 2 for Trace Level in TPL List and count TP as 1 for App Level in TPL list.

Table V reveals that ATPChecker achieves an accuracy of over 93.29% at the trace level and 87.88% at the app level for identifying the declaration of TPL usage in host apps' privacy policies. As for the false negatives related to ATPChecker's ability to identify the declaration of TPL usage in host apps' privacy policies, ATPChecker utilizes keyword matching in the compliance identification modules. However, mismatches may occur if the TPL's name coincides exactly with certain words within the sentence. Although ATPChecker cannot identify the false positives for declaration of host apps' data sharing with TPLs in their privacy policies, ATPChecker still achieves 78% recall at trace level and 73% recall at app level. The occurrence of false negatives primarily arises in cases where host apps make data-sharing claims within lengthy sentences, exceeding 70 words. In such instances, ATPChecker may incorrectly break down the subject, predicate, and object, leading to inaccurate identification of the subject involved in the data-sharing behaviors.

Table V reveals that ATPChecker identifies 56 out of 254 (22%) host apps, i.e., True Positives for identifying declaration of TPL usage in host apps' privacy policies at app level, do not clearly claim their TPL usage in privacy policies and 62 out of 254 (24%) host apps, i.e., True Positives for identifying declaration of data sharing with TPLs in host apps' privacy policies, do not clearly claim the data sharing with TPLs in host apps' privacy policies. This situation may arise due to the lack of awareness among the developers of these host apps regarding the usage of TPLs, such as certain development tools or advertising libraries, integrated within their apps. It is also possible that these developers may not have specifically recognized the extent of data sharing that occurs with TPLs.

Answer to RQ4: ATPChecker identifies that more than 22% of apps fail to comply with regulatory requirements by not disclosing their TPL usage in their privacy policies. Additionally, it identifies that over 20% of apps violate regulatory requirements by not adequately disclosing their data interactions with TPLs.

V. Discussion

This section discusses the limitation of ATPChecker, the discussion of assumption, threats to validity, and future work.

A. Limitations

ATPChecker was designed to identify whether TPLs' personal information usage complies with regulations and whether host apps' TPL usage complies with regulations. ATPChecker can only assess the binary files of TPLs and their corresponding privacy policies if they were collected simultaneously. It cannot guarantee that the binary files and privacy policies are of the same version. Note that TPLs' functions may be changed with the updates of the TPL versions. It is possible to trace TPL versions from Maven Repository, but it is not easy to trace the privacy policy of corresponding versions.

ATPChecker is based on static analysis tools like soot and flowdroid. ATPChecker cannot handle some dynamic

behaviors of TPLs (e.g., reflection, dynamic class loading), and it also cannot process large apk [51] files due to the limited memory space. Furthermore, ATPChecker can neither handle host apps in the format of *xapk* which is created by Apkpure [68] nor process native libraries.

Moreover, ATPChecker is limited to the pre-defined patterns and string matching for identifying the statements and TPL usage in collected privacy policies. The string matching method may result in some incorrect matches, for example, some matches contain the keyword but trace does not have the function of data collection. ATPChecker will fail to detect those issues if adversaries use novel patterns to hide their data usage statements [7], [72] .

B. Discussion of assumption

ATPChecker was designed to identify the compliance issues between TPLs' behavior and privacy policies, and host apps' TPL usage and privacy policies. TPLs are mainly used as an additional part of apps to enhance app functionalities and may not be responsible for data access behaviors or work in the role of data controllers as defined in GDPR. However, we cannot assume all TPLs only conduct user data access behavior for assisting apps. Existing work [6], [73], [74] has demonstrated that development tool TPLs, such as firebase [6], may leak users' privacy without developers' consciousness. Besides, regulations, such as GDPR Article 35 and SGSDK Article 5.3, request clearly giving the purpose of data processing. Even if we could assume TPLs only access user data for specific functionalities and would never share or collect the data, we still recommend TPLs provide privacy policies to clearly state their user data access behavior and related purpose. It can help not only app developers better understand TPL's functionalities but also TPL developers avoid legal disputes. Thus, we assume all TPLs should provide privacy policies when evaluating ATPChecker.

C. Threats to validity

The first threat comes from the language used in privacy policies, it is not trivial to identify and switch the language of privacy policy during the collecting phase, which may lead to a missing collection of some privacy policies. As TPLs or apps are published in different countries, such as China or Korea, the default language used in the privacy policy website is provided using their native language.

Another threat comes from inconsistent versions of TPLs and their privacy policies. Notice that all our data was collected from Feb-2022 to Apr-2022. There may be cases where the TPLs' or apps' functions have been updated, but the privacy policies have not been updated in time. This can lead our system to misidentify that the behavior of the software is inconsistent with the privacy policy and violates regulation requirements. We will mine software and privacy policy version issues in future work.

Moreover, the third threat is due to the lack of a large-scale labeled data set. We only crawl the TPL list and the top 10 host apps from AppBrain. It is very laborious to label privacy policies and collect the ground truth of data usage in software and privacy policies.

D. Future work

We will equip ATPChecker with the capability of analyzing the data collection purpose, because such information can help researchers better understand the code and detect violations. Furthermore, writing legal privacy policies remains challenging and time-consuming work for TPL developers. Automatic privacy policy generation methods are in urgent need. Although there are privacy policy generation methods [54], [75] for apps, those methods are not suitable for generating privacy policies for TPLs. In future work, we will develop automatic TPL privacy policy generation methods by combing regulations requirements analysis [31]–[33], natural language processing methods [71] and TPL analysis techniques [2].

VI. Related Work

Privacy policy conflict identification. XFinder [44] identifies the cross-library data harvesting in Android apps with dynamic analysis. XFinder identifies third-party libraries' usage by comparing the caller's and callee's package names. XFinder also restores reflection invocations using two predefined patterns. For conflict identification, XFinder manually parses the term-of-service of 40 TPLs and then uses NLP techniques to extract data sharing policies. Nguyen et al. [43] investigate whether apps achieve users' consent before sharing personal information. The authors use dynamic analysis to identify the network traffic and data sharing behaviors. They determine whether the shared data are identifiable personal data by comparing the same traffic collected from different times or the same traffic from different devices. The ablation experiments are designed to determine whether the data-sharing action achieves users' explicit consent.

PAMDroid [27] analyzes the impact of misconfigurations of analytic services in Android. After analyzing 1000 popular apps, PAMDroid finds 52 of 120 apps misconfigure the services and lead to a violation of either the service providers' term-of-service or the app's privacy policy. PPChecker [40] detects the conflicts in apps' privacy policies, but only determines whether apps' privacy policies provide TPLs' privacy policy links and interactions of five permission related personal information with 81 TPLs. POLICHECK [39] identifies the app's data sharing with third parties using dynamic analysis. POLICHECK finds that 49.5% of apps disclose their third-party sharing practices using vague terms and 31.1% of data flows as omitted disclosures. Existing works ignore analyzing whether TPLs satisfy the regulation of requirements.

TPL data leakage identification. Razaghpanah et al. [76] detect third-party advertising and tracking services using dynamic analysis. They use dynamic analysis to identify the advertising and tracking services. Specifically, they use a free app, namely Lumen Privacy Monitor, to collect all network traffic generated by all apps installed on the device. With limitations of Lumen, the proposed system can only identify

limited personally identifiable information and unique identifier, e.g., IMEI. He et al. [77] use dynamic analysis to analyze the invocation path between predefined source and sink to identify the privacy leakage of third-party libraries. Their system only concentrates on Android permission-related personal information. Their experiments on 150 popular apps demonstrate that their proposed dynamic methods achieve real-time detection and 97.4% accuracy. Ekambaranathan et al. [78] concentrate on children's apps data usage and disclosure. The researchers conduct surveys and interview with app developers to understand why apps disclose children's personal data. Liu et.al. [79] analyze the data leaking of nine analytics libraries in 300 apps. They conduct static and dynamic analyses to mitigate the privacy risk caused by analytics libraries.

VII. Conclusion

We propose an automatic third-party library regulation compliance checker, namely ATPChecker. ATPChecker was designed to identify whether TPLs satisfy regulation requirements, i.e., whether TPLs provide privacy policies and correctly claim their data usage, and whether host apps correctly disclose their usage and data interaction with TPLs. ATPChecker discovered that over 23.4% TPLs incorrectly provide the privacy policies, 37% TPLs do not disclose all of their data usages, and over 65.64% apps miss disclosing their personal information interaction with TPLs.

VIII. Data Availability

We make our dataset and tool publicly available to facilitate research in this area. We release the code and data to other researchers by responsibly sharing a private repository. The project website with instructions to request access is at: https://doi.org/10.5281/zenodo.7932665. Besides, our data set is constructed by gathering publicly available privacy policy websites and apps without posing any ethical problems.

IX. Acknowledgment

We thank the anonymous reviewers for their helpful comments. This work was partially supported by Hong Kong RGC Projects (No. PolyU15219319, and No. PolyU15224121), HKPolyU Start-up Fund (BD7H), and National Natural Science Foundation of China (No. 62202406). Research Grant from Huawei Technologies Co., Ltd.

References

[1] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, and J. Lu, "Deepintent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS), London, UK, November 11-15*, 2020.

[2] X. Zhan, L. Fan, S. Chen, F. We, T. Liu, X. Luo, and Y. Liu, "Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021.

[3] A. Short and F. Li, "Android smartphone third party advertising library data leak analysis," in *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 2014, pp. 749–754.

[4] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. A. Gunter, "Free for all! assessing user data exposure to advertising libraries on android." in *NDSS*, 2016.

[5] J. Harty, H. Zhang, L. Wei, L. Pascarella, M. Aniche, and W. Shang, "Logging practices with mobile analytics: An empirical study on firebase," in *2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft)*. IEEE, 2021.

[6] Y. Tang, H. Wang, X. Zhan, X. Luo, Y. Zhou, H. Zhou, Q. Yan, Y. Sui, and J. W. Keung, "A systematical study on application performance management libraries for apps," *IEEE Transactions on Software Engineering*, 2021.

[7] K. Zhao, H. Zhou, Y. Zhu, X. Zhan, K. Zhou, J. Li, L. Yu, W. Yuan, and X. Luo, "Structural attack against graph based android malware detection," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[8] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang, "Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing," in *Proceedings of the 2012 ACM conference on ubiquitous computing*, 2012.

[9] H. Kawabata, T. Isohara, K. Takemori, A. Kubota, J. Kani, H. Agematsu, and M. Nishigaki, "Sanadbox: Sandboxing third party advertising libraries in a mobile application," in *2013 IEEE International Conference on Communications (ICC)*. IEEE, 2013, pp. 2150–2154.

[10] X. Zhang, A. Ahlawat, and W. Du, "Aframe: Isolating advertisements from mobile applications in android," in *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013, pp. 9–18.

[11] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient privilege de-escalation for ad libraries in mobile apps," in *Proceedings of the 13th annual international conference on mobile systems, applications, and services*, 2015, pp. 89–103.

[12] S. Shekhar, M. Dietz, and D. S. Wallach, "{AdSplit}: Separating smartphone advertising from applications," in *21st USENIX Security Symposium*, 2012, pp. 553–567.

[13] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, 2012, pp. 71–72.

[14] B. He, H. Xu, L. Jin, G. Guo, Y. Chen, and G. Weng, "An investigation into android in-app ad practice: Implications for app developers," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2465–2473.

[15] P. Salza, F. Palomba, D. Di Nucci, C. D'Uva, A. De Lucia, and F. Ferrucci, "Do developers update third-party libraries in mobile apps?" in *Proceedings of the 26th Conference on Program Comprehension*, 2018, pp. 255–265.

[16] J. Zhan, Q. Zhou, X. Gu, Y. Wang, and Y. Niu, "Splitting third-party libraries' privileges from android apps," in *Australasian Conference on Information Security and Privacy*. Springer, 2017.

[17] Y. Wang, S. Hariharan, C. Zhao, J. Liu, and W. Du, "Compac: Enforce component-level access control in android," in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, 2014.

[18] M. Sun and G. Tan, "Nativeguard: Protecting android applications from third-party native libraries," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*, 2014.

[19] J. Qiu, X. Yang, H. Wu, Y. Zhou, J. Li, and J. Ma, "Libcapsule: Complete confinement of third-party libraries in android applications," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[20] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, 2012, pp. 101–112.

[21] C. Birendra, "Android permission model," *arXiv preprint arXiv:1607.04256*, 2016.

[22] G. Xu, S. Li, H. Zhou, S. Liu, Y. Tang, L. Li, X. Luo, X. Xiao, G. Xu, and H. Wang, "Lie to me: Abusing the mobile content sharing service for fun and profit," in *Proceedings of the ACM Web Conference*, 2022.

[23] B. Li, Q. He, F. Chen, X. Xia, L. Li, J. Grundy, and Y. Yang, "Embedding app-library graph for neural third party library recommendation," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 466–477.

[24] M. Di, S. Nazir, and F. Deng, "Influencing user's behavior concerning android privacy policy:an overview," *Mobile Information Systems*, 2021.

[25] S. Seneviratne, H. Kolamunna, and A. Seneviratne, "A measurement study of tracking in paid mobile applications," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2015.

[26] E. Mikalauskas, "Popular android apps with 142.5 million collective installs leak user data," 2021, https://cybernews.com/security/.

[27] X. Zhang, X. Wang, R. Slavin, T. Breaux, and J. Niu, "How does misconfiguration of analytic services compromise mobile privacy?" in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1572–1583.

[28] T. Chen, I. Ullah, M. A. Kaafar, and R. Boreli, "Information leakage through mobile analytics services," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, 2014, pp. 1–6.

[29] M. Kiranmayi and N. Maheswari, "Reducing attribute couplet attack in social networks using factor analysis," in *2018 International Conference on Recent Trends in Advance Computing (ICRTAC)*. IEEE, 2018.

[30] S. A. o. t. P. R. o. C. State Administration for Market Supervision of the People's Republic of China, "Information security technology – personal information security specification," https://www.tc260.org.cn/front/postDetail.html?id=20200918200432, 2020.

[31] "Cybersecurity practices guidelines – security guidelines for using software development kit (sdk) for mobile internet applications (app) (tc260-pg-20205a)," https://www.tc260.org.cn/front/postDetail.html?id=20201126161240, 2015.

[32] "General data protection regulation," https://gdpr-info.eu, 2016.

[33] "California consumer privacy act regulations," https://govt.westlaw.com/calregs, 2016.

[34] "Code of federal regulations," https://www.ecfr.gov/reader-aids/using-ecfr/getting-started, 2017.

[35] "Developer Program Policy: April 6, 2022 announcement," 6, April, 2022, https://support.google.com/googleplay/android-developer/answer/11498144?hl=en&ref_topic=9877065.

[36] "Google Developer Program Policy," 2022, https://support.google.com/googleplay/android-developer/answer/11498144?hl=en.

[37] "Google play policies," 2022, https://developer.android.com/distribute/play-policies.

[38] "Appgallery review guidelines," https://developer.huawei.com/consumer/en/doc/30202, 2022.

[39] 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 *29th USENIX Security Symposium*, Aug. 2020.

[40] L. Yu, X. Luo, J. Chen, H. Zhou, T. Zhang, H. Chang, and H. K. N. Leung, "Ppchecker: Towards accessing the trustworthiness of android apps' privacy policies," *IEEE Transactions on Software Engineering*, 2021.

[41] 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 *28th USENIX Security Symposium*, 2019.

[42] L. Yu, X. Luo, X. Liu, and T. Zhang, "Can we trust the privacy policies of android apps?" in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016.

[43] T. T. Nguyen, M. Backes, N. Marnau, and B. Stock, "Share first, ask later (or never?) studying violations of gdpr's explicit consent in android apps," in *30th USENIX Security Symposium*, 2021.

[44] J. Wang, Y. Xiao, X. Wang, Y. Nan, L. Xing, X. Liao, J. Dong, N. Serrano, H. Lu, X. Wang, and Y. Zhang, "Understanding malicious cross-library data harvesting on android," in *30th USENIX Security Symposium*, 2021.

[45] "California privacy rights act," https://www.weil.com/-/media/the-california-privacy-rights-act-of-2020-may-2021.pdf, 2020.

[46] F. Petitcolas, "La cryptographie militaire," 1883.

[47] "International covenant on civil and political rights," 2020, https://www.justice.govt.nz/justice-sector-policy/constitutional-issues-and-human-rights/human-rights/international-human-rights/international-covenant-on-civil-and-political-rights/.

[48] "Appbrain," https://www.appbrain.com/stats/libraries, 2021.

[49] "Maven repository," https://mvnrepository.com/, 2022.

[50] S. R. Group, "Soot," https://github.com/soot-oss/soot.

[51] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *Acm Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[52] C. Schindler, M. Atas, T. Strametz, J. Feiner, and R. Hofer, "Privacy leak identification in third-party android libraries," in *2022 Seventh International Conference On Mobile And Secure Services*, 2022.

[53] Y. Tang, X. Zhan, H. Zhou, X. Luo, Z. Xu, Y. Zhou, and Q. Yan, "Demystifying application performance management libraries for android," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 682–685.

[54] L. Yu, T. Zhang, X. Luo, and L. Xue, "Autoppg: Towards automatic generation of privacy policy for android applications," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015.

[55] "Measures for determining the illegal collection and use of personal information by apps," https://www.chinajusticeobserver.com/law/, 2019.

[56] M. J. Harrold and M. L. Soffa, "Efficient computation of interprocedural definition-use chains," *ACM Trans. Program. Lang. Syst.*, 1994.

[57] "Android developers," 2021, https://developer.android.com/.

[58] "Google play," https://play.google.com/, 2022.

[59] S. Gojare, R. Joshi, and D. Gaigaware, "Analysis and design of selenium webdriver automation testing framework," *Procedia Computer Science*, 2015.

[60] "html2text," https://github.com/Alir3z4/html2text/blob/master/docs/usage.md, 2011.

[61] "Nlp preprocess," https://stackoverflow.com/questions/54396405/, 2020.

[62] E. Loper and S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the ACL Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics*, 2002.

[63] H. He and J. D. Choi, "The stem cell hypothesis: Dilemma behind multi-task learning with transformer encoders," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021.

[64] Y. Zhang, H. Zhou, and Z. Li, "Fast and accurate neural crf constituency parsing," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021.

[65] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer, "Polisis: Automated analysis and presentation of privacy policies using deep learning," in *27th USENIX Security Symposium*, 2018.

[66] "Github," https://github.com, 2022.

[67] "Google privacy," https://policies.google.com/privacy?hl=en, 2022.

[68] "Apkpure," https://apkpure.com, 2020.

[69] L. Qiu, Y. Wang, and J. Rubin, "Analyzing the analyzers: Flowdroid/iccta, amandroid, and droidsafe," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 176–186.

[70] A. Sorokin and D. Forsyth, "Utility data annotation with amazon mechanical turk," in *2008 IEEE computer society conference on computer vision and pattern recognition workshops*. IEEE, 2008, pp. 1–8.

[71] K. Zhao, L. Yu, S. Zhou, J. Li, X. Luo, Y. F. A. Chiu, and Y. Liu, "A fine-grained chinese software privacy policy dataset for sequence labeling and regulation compliant identification," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP*. Association for Computational Linguistics, Dec. 2022.

[72] N. Boucher, I. Shumailov, R. Anderson, and N. Papernot, "Bad characters: Imperceptible nlp attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022.

[73] W. Meng, R. Ding, S. P. Chung, S. Han, and W. Lee, "The price of free: Privacy leakage in personalized mobile in-apps ads." in *NDSS*, 2016.

[74] X. Liu, S. Zhu, W. Wang, and J. Liu, "Alde: privacy risk analysis of analytics libraries in the android ecosystem," in *International Conference on Security and Privacy in Communication Systems*, 2016.

[75] S. Zimmeck, R. Goldstein, and D. Baraka, "Privacyflash pro: Automating privacy policy generation for mobile apps." in *NDSS*, 2021.

[76] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, P. Gill *et al.*, "Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem," in *The 25th Annual Network and Distributed System Security Symposium*, 2018.

[77] Y. He, B. Hu, and Z. Han, "Dynamic privacy leakage analysis of android third-party libraries," in *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, 2018.

[78] A. Ekambaranathan, J. Zhao, and M. Van Kleek, ""money makes the world go around": Identifying barriers to better privacy in children's apps from developers' perspectives," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.

[79] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the android ecosystem," *IEEE Transactions on Mobile Computing*, vol. 19, no. 5, pp. 1184–1199, 2019.