On the (Un)Reliability of Privacy Policies in Android Apps

Luca Verderame[1], Davide Caputo[1], Andrea Romdhana[1,2], Alessio Merlo[1]
[1]*DIBRIS*, *University of Genoa*, Genoa, Italy
Email: {luca.verderame, davide.caputo, andrea.romdhana, alessio}@dibris.unige.it
[2]*Security & Trust Unit*, *FBK-ICT*, Trento, Italy

***Abstract*—The access to privacy-sensitive information on Android is a growing concern in the mobile community. Albeit Google Play recently introduced some *privacy guidelines*, it is still an open problem to soundly verify whether apps actually comply with such rules. To this aim, in this paper, we discuss a novel methodology based on a fruitful combination of static analysis, dynamic analysis, and machine learning techniques, which allows assessing such compliance. More in detail, our methodology checks whether each app i) contains a privacy policy that complies with the Google Play privacy guidelines, and ii) accesses privacy-sensitive information only upon the acceptance of the policy by the user. Furthermore, the methodology also allows checking the compliance of third-party libraries embedded in the apps w.r.t. the same privacy guidelines.**

We implemented our methodology in a tool, 3PDroid, and we carried out an assessment on a set of recent and most-downloaded Android apps in the Google Play Store. Experimental results suggest that more than 95% of apps access user's privacy-sensitive information, but just a negligible subset of them ($\approx 1\%$) fully complies with the Google Play privacy guidelines.

几个方法结合

***Index Terms*—Android, Privacy Guidelines, Static Analysis, Dynamic Analysis, Machine Learning.**

I. Introduction

According to Statista[1], the number of available Android applications (hereafter, apps) was lowering in 2019, for the first time. This fact suggests that the app market competition is becoming more fierce, where a lot of apps drop from the Google Play Store due to obsolescence or lack of interest by the users' community. In order to stay on top, apps need to keep monitoring the users' preferences and demands, by continuously harvesting and gathering both user's and device information during their execution. Unfortunately, most of such information are privacy-related (e.g., the user's location through GPS, the contact list, and the IMEI of the device), and raised privacy concerns from both corporate and personal users (see, e.g., [1], [2]). This is due to the recent discovery of severe data breaches involving mobile apps, like the one discovered in the Peekaboo Moments Android app which exposed more than 100 GB of images and videos of babies[2].

To deal with users' privacy demand, mobile apps should only access the minimum amount of *personal and sensitive information* (hereafter, PSI) which could be sufficient to provide the service they are offering. Furthermore, they should clearly state which PSI is accessed.

To try mitigating the privacy problem on the Android platform, the Google Play Store released a detailed document containing a set of *privacy guidelines* for Android apps [3]. This document contains the technical and legal requirements concerning *"the collection, use, and sharing of the data, and limiting the use of the data to the purposes disclosed, and the consent provided by the user"*. As a consequence, all apps on the Google Play Store that need to access PSI must have a *privacy policy* to notify the user about how they collect, use, share, and process such information. It is also mandatory that this policy is hosted inside the app, and is prompted to (and accepted by) the user before accessing any PSI.

Research challenge: Albeit the definition of a set of privacy guidelines is an important step towards providing privacy guarantees to Android users, a sound and complete methodology to assess whether each app actually complies with them is still missing. Furthermore, it is likewise unclear how many apps actually comply with the Google Play privacy guidelines.

A. State of the art

In 2018, Google carried out an extensive analysis in the Google Play Store, thereby removing all apps that did not have at least an external link to a valid privacy policy page on their Google Play Store page. To prevent Google from deleting their apps, several developers added such link, bringing the percentage of apps that satisfies this requirement from 41.7% in 2017 to 51.8% in 2018 [4]. However, no further checks on the content of the privacy policy page have been carried out, nor on the compliance between the privacy policy and the actual behaviors of the app.

To deal with previous issues, Zimmeck et al. [5] carried out an extensive analysis of 17.991 free Android apps on the Google Play Store by leveraging static analysis and machine learning techniques. The authors found that 71% of apps that lack a privacy policy should have one, as they access PSI. Also, for $9,050$ apps that have a privacy policy page, the authors found a lot of *potential* inconsistencies between the content of the policy page and the app behavior. Unfortunately, since the authors relied on static analysis only, the actual number of true positives cannot be verified. In [6], the authors proposed a tool, named Mobile App Privacy System (MAPS),

[1] https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[2] https://www.bankinfosecurity.com/babys-first-breach-app-exposes-baby-photos-videos-a-13603

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

which is able to carry out a more extensive analysis of Android apps. MAPS is based on a pipeline for retrieving and analyzing large app populations based on code analysis and machine learning techniques. MAPS analyzed $1,035,853$ apps taken from the Google Play Store, and found that only 50.5% of them actually have a privacy policy page. In [7], the authors propose PolicyLint, a privacy policy analysis tool able to identify contradictions inside the privacy policy page. The authors analyzed $11,430$ apps and found that 14.2% of the privacy policies contain contradictions that may suggest the presence of misleading statements.

Open challenges: We argue that previous proposals suffer from some limitations. First, they rely on static analysis techniques only, thereby making hard to identify actual true positives. Moreover, they focus on the privacy policy page published on the Google Play Store only, which may differ from the one contained in the app and prompted to the user at runtime. Furthermore, they are not able to *i)* identify the privacy policy page inside the app, *ii)* verify whether such page complies with the privacy guidelines of the Google Play Store, *iii)* detect whether the app begins to access PSI *before* the user explicitly accepts the privacy policy.

为什么要进行dyna分析？

Previous challenges require to assess the behavior of the app *dynamically* and are getting worse by the fact that recent apps made extensive usage of third-party libraries and frameworks [8], which boost the complexity in assessing the type of data that is collected and processed by the app.

3PDroid: In this paper, we propose a novel methodology to automatically verify the compliance of an Android app with the privacy guidelines of the Google Play Store and we implemented it in a tool, named 3PDroid[3] This tool combines static and dynamic analysis approaches with machine learning techniques to monitor the runtime behavior of an Android app in order to verify whether:

存在性 1) the app contains a privacy policy page;
完整性 2) the privacy policy page fully complies with the Google Play privacy guidelines;
滞后性 3) the app accesses PSI *only after* the user has accepted the privacy policy;
还考虑SDK PP与行为的一致性 4) the access to PSI - carried out by both the native app and its included third-party libraries/frameworks - complies with the app privacy policy.

Structure of the paper: The rest of the paper is organized as follows: Section II introduces some basics on Android and the Google Play privacy guidelines, while Section III presents a methodology to automatically assess the compliance of Android apps with the privacy guidelines. Section IV discusses the implementation of the proposed methodology in 3PDroid, as well as an experimental setup aimed at systematically analyzing sets of Android apps. Section V shows and discusses the experimental results, while Section VI concludes the paper and points out some extension of the work.

[3] The tool and the results are available at https://csec.it/3pdroid.

II. Background

Modern mobile devices gather a plethora of PSI, which could refer to different categories, like financial and payment data, authentication information, phonebook, contacts, SMS, call-related data, microphone, and camera sensor data, just to cite a few. Android apps extensively collect, store, and process PSI in order to improve their quality and reliability, as extensively discussed in [9]–[11]. From a technical standpoint, apps can retrieve PSI either by i) relying on the Android platform API or ii) embedding third-party libraries.

Android API and Permissions: The Android OS allows apps to get access to PSI through a set of well-defined API. At the same time, Android limits the access to PSI through a security mechanism based on the idea of *permission*. In a nutshell, each PSI-related API is associated with a set of privacy-sensitive permissions. The invocation of an API is then restricted to the sole apps having the required set of permissions. Android apps must require permissions by declaring them in their *AndroidManifest.xml* file in a specific tag named *uses-permission* [12].

According to the type of permission, the OS might grant it automatically or might prompt the user to approve the request. For example, if an app aims to read data from the contact list, it must declare the "`android.permission.READ_CONTACTS`" permission within the *AndroidManifest.xml* file in order to be allowed to invoke the corresponding API (e.g., `getPhoneNumbers`). Then, the first time the app tries to access the API, the user is prompted with a permission grant request. If the user accepts, the app can invoke the API, thereafter.

For the aim of this work, we mapped the standard permission set provided by the Android OS with the corresponding API methods and the PSI. An excerpt of such mapping, inspired by the works of [13] and [11], is provided in Table I.

Permission	**PSI**	**API method**
`ACCESS_NETWORK_STATE`	Device MAC	`getMacAddress()`
`ACCESS_FINE_LOCATION`	GPS Location	`getLocation()`
`READ_PHONE_STATE`	IMEI	`getDeviceId()`
`READ_PHONE_STATE`	Phone Number	`getLine1Number()`
`ACCESS_WIFI_STATE`	Router MAC	`getMacAddress()`
`ACCESS_WIFI_STATE`	Router SSID	`getBSSID()`

TABLE I: Mapping of Android permissions, PSI and API methods.

Third-party Libraries: The previous mapping provides only a partial coverage of the full PSI. In fact, there exists a wide set of PSI that is beyond the control of the OS and, therefore, it is outside the enforcement mechanisms of the Android permissions. Examples of this PSI include all the usage statistics like the commercial or usage preferences of the user, collected for profiling aim, or to build crash reporting activities. To collect this PSI, developers extensively include in their apps third-party libraries for analytics and advertising that can be triggered using a set of the API methods [14].

For instance, analytics libraries can retrieve and store several pieces of PSI, including the country as well as the type and the model of the mobile device. Most of these libraries can also track the user's activities. Furthermore, developers may use *Ad Network* libraries to deliver application ads in order to increase revenues. To this aim, the Ad Network library may access PSI to show personalized ads customized to the taste of the users. Notables examples of analytics and Ad Network libraries include AdMob[4], Google Analytics[5], InMobi[6], and Facebook Analytics[7].

In this work, we identified a set of the most used analytics and Ad Network libraries according to [15], and we mapped the API methods that involve the collection, process, and use of PSI. An excerpt of such mapping, presenting a subset of InMobi API methods, is presented in Table II.

API method	PSI
`setKeywords`	Keywords
`setSearchString`	Keywords
`setGender`	Gender
`setCurrentLocation`	Location
`setAge`	Age
`setRequestParams`	Multiple Factors
`setPostalCode`	Postal Code
`setLocationInquiryAllowed`	Enable Location
`setIncome`	Income
`setInterests`	Interests
`setAreaCode`	Area Code
`setEducation`	Education
`setEthinicity`	Ethnicity

TABLE II: Examples of API methods extracted from the InMobi library and their mapping with PSI.

A. The Google Play Privacy Guidelines

The growing concerns on PSI pushed Google Play to release a "Privacy, Security and Deception" guideline document [3] to grant transparency on the access and usage of PSI by Android apps. Such guidelines force each developer to restrict the collection and the use of sensitive data only for aims directly related to the supply and improvement of the functionality of the app. Furthermore, the developer must handle all of these pieces of data safely and transmit them using modern encryption mechanisms (i.e., via HTTPS).

Still, in case PSI are gathered at runtime, the app must provide an in-app disclosure regarding data collection and usage, i.e., a *privacy policy page*. Such page must meet a set of *technical* (*TR*) and *content* (*CR*) requirements, as detailed in Table III, in order to be compliant with the guidelines. In a nutshell, the application must include the policy within the app, and prompt it to the user without requiring her to open a menu or the settings. Moreover, the app must require the explicit consent of the user, e.g., by avoiding timeout or automatic acceptance actions. Finally, the privacy policy page must precisely describe both the set of collected PSI and for which aim. At the same time, the app must not collect PSI before obtaining the user's consent.

Id	Description
`CR1`	The privacy policy page must clearly state the set of PSI it collects, as well as how the same PSI will be used.
`CR2`	The privacy policy page must be prompted to the user in a proper document, which is different from the terms of service or other documents that do not deal with personal or sensitive information.
`TR1`	The privacy policy page must be stored within the app, i.e., it is not sufficient to store it on a website or on the app page in the Google Play Store.
`TR2`	The privacy policy page must be shown during the execution of the app once and automatically prompted, i.e., the user must not search for it in a menu or a settings page.
`TR3`	The user must explicitly accept the privacy policy, e.g., it must click on an acceptance widget.
`TR4`	The app must not collect PSI before the user accepts the privacy policy.
`TR5`	The app must not consider the privacy policy as accepted if the user leaves the privacy policy page by pressing the home or back button.
`TR6`	Once prompted, the privacy policy page must not expire before the user could accept it.

TABLE III: Technical (TR) and content (CR) requirements of the Google Play privacy guidelines.

III. Assessing the Privacy Guidelines Compliance

We discuss here a novel methodology based on a combination of static analysis, dynamic analysis, and machine learning techniques to address the open research challenges described at the end of Section I. This methodology can be applied to any Android application package without requiring the source code nor any additional information, and it is composed of 7 different modules that cooperate according to the workflow sketched in Figure 1. The rest of this section details the modules and their interactions.

A. PSI Mapping

The *PSI Mapping* contains the API methods belonging to both the Android OS and the third-party libraries that are relevant for the analysis, i.e., that collect, use, or process PSI. In detail, the *PSI Mapping* contains the mapping of all the privacy-related Android permissions (see Table I) and the third-party libraries (see Table II) with their corresponding PSI-related API methods.

B. App Analyzer

The *App Analyzer* decompiles an Android application package (APK) and extracts *i)* the list of permissions requested by the app and *ii)* the list of third-party libraries included in the app. For each finding, the module queries the *PSI Mapping* to determine if the permission or the library is privacy-sensitive. If this is the case, it requests the PSI and the corresponding list of API methods that are involved. Then,

[4] https://admob.google.com/intl/it/home/

[5] ttps://firebase.google.com/docs/analytics

[6] https://www.inmobi.com/

[7] ttps://analytics.facebook.com/

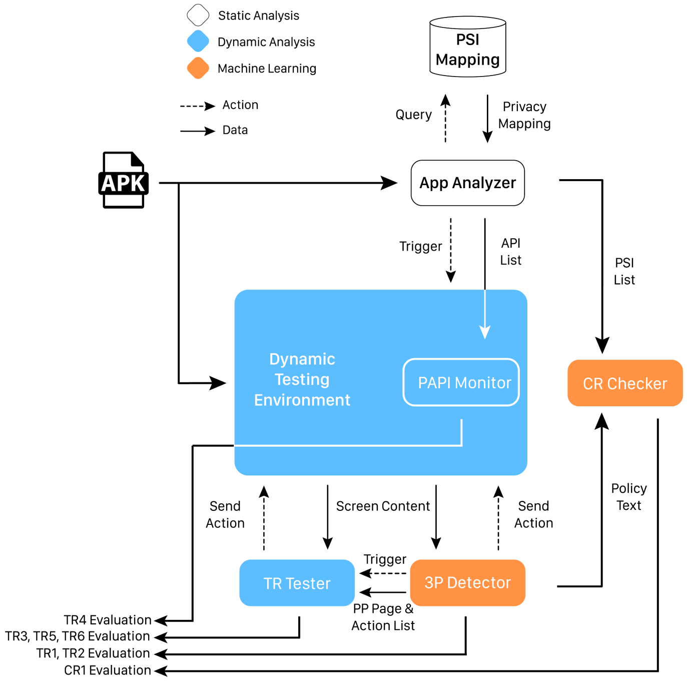

Fig. 1: Analysis Workflow.

the *App Analyzer* dispatches the list of PSI to the *Content Requirement Checker (CR Checker)* and the list of involved API methods to the *Privacy API Monitor (PAPI Monitor)*. Finally, the module triggers the *Dynamic Testing Environment (DTE)* for the dynamic analysis phase. Instead, if the app does not contain any privacy-related permission or library, it is marked as compliant and no further analyzed.

C. *Dynamic Testing Environment*

The *Dynamic Testing Environment (DTE)* is the module in charge of the installation and the execution of the app inside a device emulator. First, the DTE initializes an Android emulated device and installs the APK. Then, the module executes the app, and notifies both the PAPI Monitor and the *Privacy Policy Page Detector (3P Detector)* to start the analysis.

D. *Privacy Policy Page Detector (3P Detector)*

The goal of the *Privacy Policy Page Detector (3P Detector)* module is to identify the privacy policy page screen inside the Android app. First, the 3P Detector connects to the DTE and searches for the privacy policy page.

Following the algorithm depicted in Algorithm 1, the module retrieves the XML content of the app screen which is currently displayed to the user (row 5). Then, the module aims to determine whether the textual content of the extracted page includes a privacy policy (row 6-8). To do so, the 3P Detector enforces a text classification algorithm based on machine learning techniques. The classification procedure requires two phases, namely *preprocessing* and *classification*.

Preprocessing: At first, the preprocessing extracts the text content (row 6) from the XML of the page. Then, the extracted text is polished to normalize whitespace and punctuation, remove non-ASCII characters, and lowercase all the content (row 7).

Classification: The preprocessed text is then evaluated to determine if it contains a privacy policy page. For such a task, 3P Detector relies on a multilayer perceptron classifier (row 8). If the classifier marks the text as a privacy policy, the 3P Detector sets the flag `detected` and exits the research loop. Otherwise, the module sends an input to the app (e.g., a swipe or a press command) in order to evaluate a new screen at the next iteration (row 12-13). Also, 3P Detector stores a list of the input actions (row 14) used to reach the privacy policy page. This information will be sent to the *Technical Requirement Tester* to evaluate the list of `TRs` (see Table III). Finally, if the module successfully detects the policy page, it returns the text of the policy, the XML file, and the list of actions required to reach the page (row 18). On the contrary, if the module cannot find the policy page within a maximum limit of input actions (i.e., *MA*), the app is marked as not compliant, and the analysis terminates. Indeed, since apps must satisfy `TR2`, the module expects to reach the policy page within such a threshold.

Algorithm 1: 3P Detector

```
Input  : Emulator, MA
Output: textContent, PPPXml, listActions
Output: TR1-TR2 Evaluation
1  listActions ← makeList ();
2  detected ← False;
3  emulator.startAPK ();
4  while not detected and listActions.len () < MA do
5      xmlContent ← emulator.getPageContent ();
6      textContent ← extractFromXML (xmlContent);
7      preprText ← dataPreprocessing (textContent);
8      detected ← MLPClassifier.isPolicyPage (preprText);
9      if detected then
10         break;
11     end
12     nextAction ← getNextAction (pageContent);
13     emulator.performAction (nextAction);
14     listActions.add (nextAction);
15 end
16 if detected then
17     PPPXml ← xmlContent;
18     return textContent, PPPXml, listActions;
19 end
20 return Fail_TR1_TR2;
```

E. *Privacy API Monitor (PAPI Monitor)*

This module verifies whether the app does invoke any PSI-related API before the user explicitly accepts the privacy policy page. Once it receives the start notification from the DTE, the *Privacy API Monitor* connects to the testing environment and begins to monitor the execution of any of the PSI-related API method received from the *App Analyzer*. If the module logs any of these methods, then the app does not fulfill the TR4 rule, and thus it is marked as not compliant.

F. Technical Requirement (TR) Tester

The aim of the *Technical Requirement (TR) Tester* module is to identify whether the privacy policy page satisfies the technical requirements described in Section II-A. To do that, the TR Tester implements Algorithm 2.

First, the module evaluates the TR3 requirement by analyzing the XML of the page to detect any element allowing the user to accept the policy explicitly (e.g., a button or a checkbox); if no match is found, the app is marked as not compliant, and the evaluation terminates (rows 1-3). Otherwise, the module checks if the page expires or closes automatically within a given threshold (row 4), in order to verify the TR6 requirement. If the policy page is still displayed when the threshold is reached, the module then proceeds with the evaluation of the TR5 requirement. To this aim, it triggers the home button and then re-opens the app (rows 8-10): if the app screen is different from the privacy policy page, then the app considers leaving the policy windows as an act of acceptance, and thus it is marked as not compliant (row 14-15). The experiment is repeated with the back button (rows 11-13).

Algorithm 2: TR Tester

```
   Input  : Emulator, PPPXml, listActions
   Output: TR3, TR5, TR6 Evaluation
1  if not PPPXml.containsExplAccept () then
2  |   return Fail_TR3;
3  end
4  wait(timeout);
5  if emulator.getPageContent () != PPPXml then
6  |   return Fail_TR6;
7  end
8  emulator.pressHomeButton();
9  emulator.startAPK();
10 pageHomeButton ← emulator.getPageContent();
11 emulator.pressBackButton();
12 emulator.startAppWithPrivacyPolicyPage();
13 pageBackButton ← emulator.getPageContent();
14 if pageBackButton != PPPXml or pageHomeButton != PPPXml
     then
15 |   return Fail_TR5;
16 end
17 return Pass;
```

G. Content Requirement (CR) Checker

The *Content Requirement (CR) Checker* verifies whether the privacy policy page successfully declares all PSI requested by the app, following the CR1 requirement. Given the list of PSI received by the App Analyzer, the module executes several machine learning-based classifiers.

For this task, CR Checker identifies for each PSI the most meaningful keywords and creates two distinct sets, as suggested in [5]. The first one is used to preprocess the policy text and extract all the sentences that contain at least one of the keywords associated with the data type of the PSI under investigation (e.g., for the location PSI, we use terms like 'position' or 'GPS'). On the resulting sentences, CR Checker uses a second set of keywords that refer to the actions available for a given PSI (e.g., for the location PSI, we use terms like 'share' or 'partner'), to construct unigram and bigram feature vectors [16].

The feature vectors are then used to classify the policy. CR Checker uses a set of machine learning models (one for each PSI, see Table IV) to determine whether all the pieces of PSI are included in the policy. If this is not the case, the module raises a warning regarding the CR1 requirement. In this case, the app must pass through a manual inspection phase.

IV. Implementation and Experimental Setup

We implemented a prototype of our methodology, called *3PDroid*, to evaluate both the detection accuracy and the performance on a dataset of Android apps. The rest of this section describes the implementation choices and the corresponding experimental setup.

App Analyzer: 3PDroid implements the *App Analyzer* as a Python script based on the Androguard library[8]. Androguard makes available several APIs allowing to parse the *AndroidManifest.xml* file and retrieve all the privacy-related libraries. Furthermore, the *App Analyzer* queries the *PSI Mapping* DB to build the list of API to be monitored during the dynamic analysis phase.

PSI Mapping: The *PSI Mapping* is a MongoDB database storing the mappings among PSI, API Methods, Android permissions, and third-party libraries. This dataset has been built by parsing the Android API reference website[9] as well as the websites and the documentation provided by third-party developers. The dataset is composed of a set of JSON object documents. The current version of the *PSI Mapping* includes the most used libraries according to [17], like, e.g., AdMob and Google Analytics.

The first column of Table IV shows all PSI taken into consideration in this work and inspired by [4]. It is worth noticing that a privacy policy page could combine both coarse-grained (e.g., "contact_information") and fine-grained (e.g. "contact_email_address") PSI.

Dynamic Testing Environment: The app is executed on an emulated x86[10] device based on Android 6.0 with root permissions, and equipped with Frida, a dynamic code instrumentation toolkit[11]. The DTE module is attached to the emulator and orchestrates the installation, execution, and stimulation of the app under test by leveraging the Android Debug Bridge (ADB)[12]. The *Privacy API Monitor* relies on Frida to instrument the app at runtime. This module is configured to log only the privacy-related API identified by

[8] https://androguard.readthedocs.io/

[9] https://developer.android.com/reference/packages?hl=en

[10] https://www.android-x86.org/

[11] https://frida.re/docs/home/

[12] https://developer.android.com/studio/command-line/adb

CR Checker task	Model	Best Parameters	P	A	R
contact_address	RF	entropy, log2, 1000	85%	85%	85%
contact_city	RF	entropy, log2, 100	73%	73%	73%
contact_email_address	SVM	1, 0.1, rbf	77%	77%	77%
contact_information	SVM	0.1, linear	82%	82%	82%
contact_password	RF	entropy, auto, 100	83%	83%	83%
contact_phone_number	LR	5, 20, l2, 10, lbfgs	78%	78%	78%
contact_postal	MNB	0.5, True	80%	80%	80%
contact_zip	AB	SAMME, 500	81%	81%	81%
demographic_age	RF	gini, auto, 50	83%	83%	83%
demographic_gender	AB	SAMME.R, 265	78%	78%	78%
demographic_information	RF	gini, auto, 150	82%	82%	82%
identifier_ad_id	RF	entropy, auto, 10	83%	83%	83%
identifier_cookie	SVM	1, 1, rbf	88%	88%	88%
identifier_device	RF	gini, auto, 50	79%	79%	79%
identifier_imei	RF	entropy, log2, 50	91%	91%	91%
identifier_imsi	RF	gini, auto, 5	100%	100%	100%
identifier_information	RF	gini, log2, 10	79%	79%	79%
identifier_ip_address	RF	entropy, log2, 1000	73%	73%	73%
identifier_mac	RF	gini, auto, 50	88%	88%	88%
identifier_sim_serial	MNB	1.5, True	100%	100%	100%
identifier_SSID_BSSID	RF	gini, auto, 50	92%	92%	92%
location_bluetooth	RF	entropy, log2, 10	83%	83%	83%
location_cell_tower	RF	gini, auto, 100	87%	87%	87%
location_gps	RF	gini, log2, 300	82%	82%	82%
location_information	SVM	0.01, linear	73%	73%	73%
location_ip_address	MNB	0.5, False	78%	78%	78%
location_wifi	RF	entropy, auto, 25	82%	82%	82%
performed_not_performed	RF	gini, auto, 500	96%	96%	96%
third_party_first_party	RF	gini, log2, 300	98%	98%	98%

TABLE IV: CR Checker tasks: model parameters and evaluation in terms of precision (P), accuracy (A), and recall (R).

the *App Analyzer*, and to analyze an app for ten minutes at most.

Privacy Policy Page (3P) Detector: The 3P Detector stimulates the app using the DTE module and extracts the XML of the app screens using the UI Automator tool[13]. Then, the module pre-processes the policy text using the NLTK[14] libraries, and relies on a multilayer perceptron (MLP) based text classifier based on the TensorFlow library [18] to identify the policy page. The MLP is composed of two hidden layers made by 64 perceptrons each. The activation function of the output layer is a sigmoid function, and the loss function used to train the model is a binary cross-entropy function. Moreover, we set the learning rate to 0.0001, the number of epochs to 1000, a batch size to 128, and a dropout rate to 0.2 in order to prevent overfitting.

We leveraged the APP-350 [6] and a subset of the News Summary[15] datasets - labeled as "`policy`" and "`not_policy`" respectively - as a basis to build a new dataset for training and validating the MLP model. Then, we carried out the following operations on this dataset: i) we tokenized the documents in n-grams (with $n = 1$ and $n = 2$), ii) we computed the importance of each n-gram using the $tf{-}idf$ function [19] and iii) we selected the best $20k$ n-grams based on the ANOVA F-value [20] statistical test.

The output of the MLP model is a *probability score* that indicates the likelihood that the analyzed privacy page belongs to the "`policy`" class. We defined a confidence threshold to 90%, i.e., the page is classified as "`not_policy`" if the score is less than the threshold; otherwise, it is classified as "`policy`". We selected a high threshold in order to automatically discard all text pages which may resemble a privacy policy page (e.g., the "Terms & Conditions" pages), thereby reducing the likelihood of false positives.

Technical Requirement (TR) Tester: The TR Tester interacts with the emulator using the DTE module and the UI Automator tool to evaluate the compliance of the privacy policy page w.r.t. the list of TRs (Table III). As described in Section III, we defined a maximum number of 20 actions to check the compliance with `TR2`, and we set a timeout of 10 seconds for `TR6`.

Content Requirement (CR) Checker: The CR Checker analyzes each sentence in a privacy policy page with the aim to i) identify the set of PSI therein, ii) check whether the sentence is affirmative or not (e.g., "We access your contacts" or "We do not access your contacts"), and iii) verify if the PSI is accessed by third-party libraries. The CR Checker pre-processes the privacy policy page by splitting it into sentences using the NLTK libraries. Following the same approach of 3P Detector, the sentences are thus tokenized in n-grams (n=1 and n=2). Then, the $tf - idf$ function is applied to evaluate the importance of each n-gram. Moreover, the CR Checker leverages 27 ML models to identify the PSI in the sentence (i.e., one for each PSI). Finally, the CR Checker leverages two ad hoc binary classifiers, i.e., *performed_not_performed* and *third_party_first_party*, to identify whether i) a sentence is affirmative, and ii) the third-party libraries access PSI, respectively.

We selected and tested a set of ML algorithms (i.e., *MultinomialNB*, *RandomForest*, *SVM*, *kNN*, *LogisticRegression*, *DecisionTree*, and *AdaBoost*) in order to find the best model for each CR Checker task. In addition, we carried out a hyperparameters optimization phase using a grid search strategy (i.e., exhaustive searching strategy) to find the best parameters for all ML algorithms; the tested parameters are summarized in Table V.

ML Algorithm	Parameters
MultinomialNB (MNB)	*alpha*, *fit_prior*
RandomForest (RF)	*critierion*, *max_features*, *n_estimators*
Support Vector Machine (SVM)	*C*, *gamma*, *kernel*
Logistic Regression (LR)	*max_iter*, *penalty*, *solver*
k-Nearest Neighbors (kNN)	*n_neighbors*, *weights*, *algorithm*
Decision Tree (DT)	*criterion*, *splitter*
AdaBoost (AB)	*algorithm*, *estimators*

TABLE V: ML algorithms and their tested parameters.

The set of the best parameters for each model, as well as the evaluation of the models in terms of precision (P), accuracy (A), and recall (R), are reported in Table IV.

V. Experimental Results

We empirically assessed the reliability of the proposed methodology by systematically analyzing a dataset of $5,473$ apps with 3PDroid. Such apps are the top free Android apps

[13] https://developer.android.com/training/testing/ui-automator

[14] https://www.nltk.org/

[15] https://github.com/sunnysai12345/News_Summary

ranked by the number of installations and average ratings according to Androidrank [21], and have been downloaded from the Google Play Store between Dec. 2019 and Jan. 2020. It is worth noticing that $4,567$ apps (i.e., $\approx 84.4\%$) have a valid link to a privacy policy page on the Google Play Store. Our experiments were conducted using a laptop equipped with an Intel Core i7-3770@3.40 GHz, 16GB RAM, and Ubuntu 18.04.

A. Overview of Apps

3PDroid allowed categorizing the dataset according to the privacy-sensitive permissions and the usage of third-party libraries for accessing PSI.

The distribution of the privacy-relevant permissions in the app dataset is shown in Table VI. The most requested privacy-sensitive permission is `ACCESS_WIFI_STATE` which allows accessing information about Wi-Fi networks and enables the extraction of tracking details about the users as explained in [22]. Other widely-used permissions include `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION` that enable the access to GPS location and `READ_PHONE_STATE` that gives access to the phone state, including the phone number and information on the cellular network. A complete description of all Android permissions, including those listed in Table VI, can be found in [12].

Permission	**Percentage**	**Ratio**
ACCESS_WIFI_STATE	47.1%	2579/5473
READ_EXTERNAL_STORAGE	45.7%	2503/5473
ACCESS_FINE_LOCATION	22.8%	1250/5473
READ_PHONE_STATE	22.5%	1233/5473
ACCESS_COARSE_LOCATION	21.4%	1157/5473
CAMERA	18.8%	1031/5473
GET_ACCOUNTS	17.2%	942/5473
RECORD_AUDIO	11.1%	606/5473
READ_CONTACTS	9.5%	518/5473
CALL_PHONE	3.9%	212/5473
READ_CALENDAR	2.0%	111/5473
READ_SMS	0.8%	43/5473
RECEIVE_SMS	0.8%	42/5473
READ_CALL_LOG	0.7%	38/5473

TABLE VI: Distribution of privacy-related permissions in the experimental dataset.

The distribution of third-party libraries for analytics and advertising is shown in Table VII. It is important to emphasize that a single PSI can be shared with one or more third-party libraries, as an app can import any number of third-party libraries. It is also worth pointing out that the most widespread libraries belong to Google (i.e., Google Ads, Google Firebase Analytics, Google DoubleClick, Google CrashLytics, and Google Analytics) and Facebook (i.e., Facebook Ads and Facebook Analytics).

B. Success Rate and Performance Analysis

3PDroid was able to analyze 92.4% of apps (i.e., $5057/5473$) successfully. The analysis of the remaining 416 apps failed due to one of the following reasons:

Library Name	**Percentage**	**Ratio**
Google Ads	82.8%	4534/5473
Google Firebase Analytics	63.1%	3454/5473
Google DoubleClick	55.6%	3044/5473
Google CrashLytics	40.2%	2201/5473
Facebook Ads	29.3%	1605/5473
Google Analytics	28.8%	1581/5473
Facebook Analytics	26.9%	1472/5473
Unity3d Ads	21.1%	1156/5473
Moat	18.9%	1036/5473
Flurry	16.3%	894/5473
Inmobo	15.9%	869/5473
AppLovin	14.8%	810/5473
Twitter MoPup	14.3%	783/5473
Vungle	12.9%	711/5473
Integral Ad Science	11.7%	640/5473
AdColony	11.5%	627/5473

TABLE VII: Distribution of third-party libraries for advertising and analytics.

- *Technical.* The dynamic analysis of 3PDroid is based on an Android with root permissions, and mounted on an emulated x86 architecture equipped with the Houdini ARM translator library. However, some apps did not execute in an emulator or on a rooted device. Other apps failed due to compatibility issues related to the Houdini library.
- *Geographical.* Some apps (e.g., banking) executes only in specific locales.

Concerning performance, 3PDroid took 413 hours for analyzing all the apps (i.e., 5473), with a mean of 272 seconds for each app, on a mid-level laptop, thereby suggesting that the approach is viable. Further optimizations like porting the 3PDroid approach on some Cloud IaS could be an interesting technical deployment to delve further that could consequently allow increasing the current thresholds for the dynamic analysis.

C. Analysis Results

The results obtained by 3PDroid are depicted in Fig. 2 and indicate that only 5.5% of the analyzed apps (i.e., $279/5057$) are compliant with the Google Play privacy guidelines.

Among these, 4.6% (i.e., $233/5057$) are *clean* apps, i.e., apps that do not access any PSI, and thus do not require a privacy policy.

It is worth pointing out how such a value suggests that only a minimal set of the current apps do not access any personal information.

The remaining 0.9% (i.e., $46/5057$) are apps that actually provides a privacy policy page that fulfills the Google Play privacy guidelines.

The amount of non-compliant apps is worrisome (94.5%, $4778/5057$), and suggests that the privacy problem could be more severe than foreseen in previous work. Furthermore, it is worth recalling that the vast majority of such apps have a valid link to a privacy policy page on the Google Play Store.

A more detailed analysis of this set shows that most of the apps (82.1%, $4150/5057$) lack an internal privacy policy page

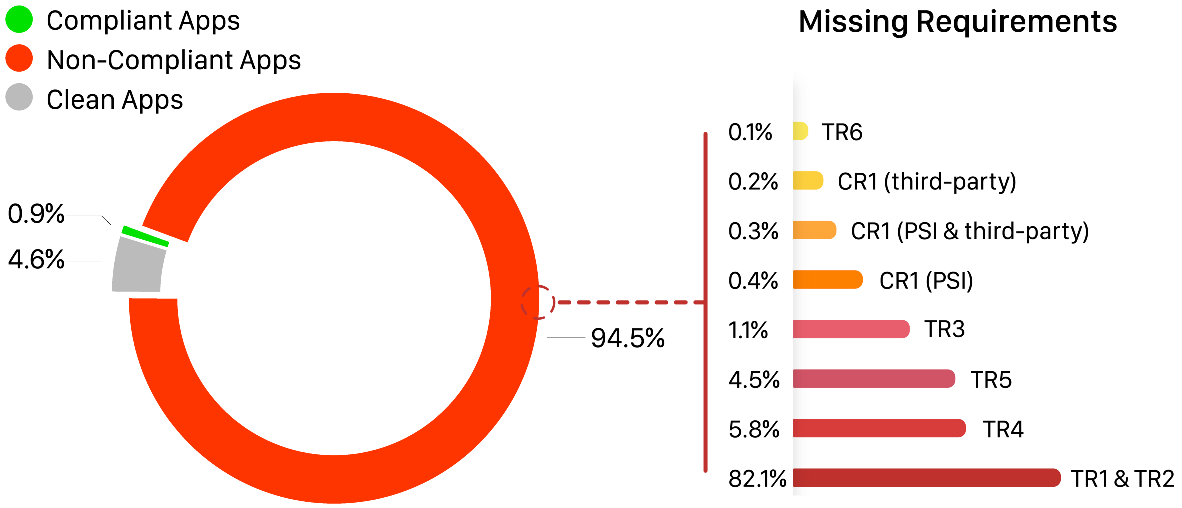

Fig. 2: Overview of 3PDroid Results on 5057 apps.

- although they get access to PSI - and thus, they do not fulfill the *TR1* and *TR2* requirements. The remaining apps (12.4%, 628/5057) do not fulfill either *TR3-TR6* or *CR1*. It is worth pointing out that the analysis process is sequential and the TRs are evaluated before $CR1$. As a consequence, all apps that are tested against $CR1$ have been previously recognized as $TR3$-$TR6$ compliant, also.

Concerning TRs, 292 apps collect PSI before an explicit acceptance of the privacy policy (*TR4*), while 228 assume that the user implicitly accepts the policy by leaving the privacy policy page (*TR5*). Moreover, 54 apps do not require an explicit acceptance from the user (*TR3*), while 6 of them have a self-expiring privacy policy page (*TR6*). Concerning the CRs, 22 apps provide just a partial description of the PSI they collect during normal execution (i.e., *CR1 PSI*), while 10 apps do not warn the user about the usage of PSI-related third-party libraries (i.e., *CR1 third-party*). Finally, 16 apps do not fulfill any of the previous requirements (i.e., *CR1 PSI & third-party*).

D. Manual Validation of 3PDroid

We manually analyzed some subsets of apps, in order to assess the reliability of the machine learning-based components (i.e., 3P Detector and CR Checker) of 3PDroid.

Regarding 3P Detector, we manually analyzed a set of 1348/5057 apps (i.e., $\approx 26.6\%$) made by two subsets. The first subset contains all 674 apps in which 3P Detector recognized an internal privacy policy page. The latter is made by other 624 apps randomly selected among the remaining ones (i.e., without a detected privacy policy page). Regarding the first subset, the manual analysis showed that 648 apps are actually policy pages (true positive - TP), while the remaining 26 hosted a different kind of page (false positive - FP). Regarding the second subset, we found that 656 apps actually lack a privacy-related content (true negative - TN), while the remaining 18 apps have an undetected privacy policy page (false negatives - FN).

Concerning CR Checker, we took into consideration the 94 apps, which fulfilled all TRs. We manually analyzed all sentences (i.e., 2961) contained in their privacy policy pages. CR Checker recognized 52/94 apps as *CR1 (PSI)* compliant, and 64/94 as *CR1 (third-party)* compliant. Regarding *CR1 (PSI)*, our manual analysis revealed that: i) 48/52 are actually compliant (TP), while 4/52 are not (FP). Furthermore, 28/42 are indeed not compliant with *CR1 (PSI)* (TN), while 14/42 revealed to be compliant (FN). Regarding *CR1 (third-party)*, we found that 63/64 are TPs (and 1/64 is a FP), while 24/30 are TNs (and 6/30 are FNs). The results of the previous analysis indicate that the adopted classifiers have a good level of performance in terms of accuracy, sensitivity, specificity, and precision, as summarized in Table VIII.

Metric		Model	Percentage	Ratio
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	3P Detector CR1 (PSI) CR1 (third-party)	96.7% 80.9% 92.6%	1304/1348 76/94 87/96
Sensitivity	$\frac{TP}{TP+FN}$	3P Detector CR1 (PSI) CR1 (third-party)	97.3% 77.4% 91.3%	648/666 48/62 63/69
Specificity	$\frac{TN}{TN+FP}$	3P Detector CR1 (PSI) CR1 (third-party)	96.2% 87.5% 96.0%	656/682 28/32 24/25
Precision	$\frac{TP}{TP+FP}$	3P Detector CR1 (PSI) CR1 (third-party)	96.1% 92.3% 98.4%	648/674 48/52 63/64

TABLE VIII: Performance of ML-based analysis in 3PDroid.

VI. Conclusion

In this paper, we introduced the first methodology which allows assessing the compliance of Android apps with the recently released Google Play privacy guidelines *at runtime*.

Our approach can be combined with the previous proposals based on static analysis, in order to build more reliable analysis workflows for evaluating the access to PSI by Android apps. Our results suggest that the vast majority of actual Android apps (i.e., 95.4% of the analyzed apps) access PSI, but just a negligible part of them (i.e., $\approx 1\%$) fully complies with the Google Play privacy guidelines.

Future extensions of this work could be i) an extensive empirical assessment of the methodology in the wild, by analyzing a higher number of Android apps, ii) the evaluation of other machine learning techniques for the detection of the privacy policy page and the classification of its contents, in order to further improve the precision, the recall, and the accuracy of the analysis, iii) extend the number of supported third-party libraries.

References

[1] P. B. Brandtzaeg, A. Pultier, and G. M. Moen, "Losing control to data-hungry apps: A mixed-methods approach to mobile app privacy," *Social Science Computer Review*, 2019.

[2] Y. Wang, Y. Chen, F. Ye, H. Liu, and J. Yang, "Implications of smartphone user privacy leakage from the advertiser's perspective," *Pervasive and Mobile Computing*, 2019.

[3] "Privacy Policy on Google Play Store," https://play.google.com/about/privacy-security-deception/, accessed: 2020-01-15.

[4] P. Story, S. Zimmeck, and N. Sadeh, "Which apps have privacy policies?: An analysis of over one million google play store apps," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018.

[5] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. Sadeh, S. M. Bellovin, and J. Reidenberg, "Automated Analysis of Privacy Requirements for Mobile Apps," 2017.

[6] S. Zimmeck, P. Story, D. Smullen, A. Ravichander, Z. Wang, J. Reidenberg, N. C. Russell, and N. Sadeh, "MAPS: Scaling Privacy Compliance Analysis to a Million Apps," *Proceedings on Privacy Enhancing Technologies*, 2019.

[7] 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 (USENIX Security 19)*. USENIX Association, 2019.

[8] Y. He, X. Yang, B. Hu, and W. Wang, "Dynamic privacy leakage analysis of android third-party libraries," *Journal of Information Security and Applications*, 2019.

[9] 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*, ser. WISEC '12. New York, NY, USA: Association for Computing Machinery, 2012.

[10] M. Huber, M. Mulazzani, S. Schrittwieser, and E. Weippl, "Appinspect: Large-scale evaluation of social networking apps," in *Proceedings of the First ACM Conference on Online Social Networks*. New York, NY, USA: Association for Computing Machinery, 2013.

[11] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "Recon: Revealing and controlling pii leaks in mobile network traffic," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: Association for Computing Machinery, 2016.

[12] "Android manifest," https://developer.android.com/guide/topics/manifest/manifest-intro, accessed: 2020-01-20.

[13] I. Reyes, P. Wijesekera, J. Reardon, A. Elazari, A. Razaghpanah, N. Vallina-Rodriguez, and S. Egelman, ""won't somebody think of the children?" examining coppa compliance at scale," *Proceedings on Privacy Enhancing Technologies*, 2018.

[14] Y. He, X. Yang, B. Hu, and W. Wang, "Dynamic privacy leakage analysis of android third-party libraries," *Journal of Information Security and Applications*, 2019.

[15] T. Book and D. S. Wallach, "A case of collusion: A study of the interface between ad libraries and their apps." New York, NY, USA: Association for Computing Machinery, 2013.

[16] S. Zimmeck and S. M. Bellovin, "Privee: An architecture for automatically analyzing web privacy policies," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[17] "App brain analytics," https://www.appbrain.com/stats/libraries/tag/analytics/android-analytics-libraries, accessed: 2020-01-20.

[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[19] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*. Piscataway, NJ, 2003.

[20] E. R. Girden, *ANOVA: Repeated measures*. Sage, 1992, no. 84.

[21] AndroidRank. (2019) Androidrank market data. [Online]. Available: https://www.androidrank.org/

[22] J. P. Achara, M. Cunche, V. Roca, and A. Francillon, "Short paper: Wifileaks: underestimated privacy implications of the access_wifi_state android permission," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*, 2014, pp. 231–236.