# An Empirical Evaluation of GDPR Compliance Violations in Android mHealth Apps

Ming Fan*, Le Yu†, Sen Chen‡§, Hao Zhou†, Xiapu Luo†, Shuyue Li*, Yang Liu§, Jun Liu*, Ting Liu*

*School of Cyber Science and Engineering, MoE KLINNS, Xi'an Jiaotong University, China
†Department of Computing, The Hong Kong Polytechnic University, China
‡ College of Intelligence and Computing, Tianjin University, China
§School of Computer Science and Engineering, Nanyang Technological University, Singapore
mingfan@mail.xjtu.edu.cn; yulele08@gmail.com; ecnuchensen@gmail.com; cshaoz@comp.polyu.edu.hk;
luoxiapu@gmail.com; lishuyue1221@stu.xjtu.edu.cn; yangliu@ntu.edu.sg; liukeen@xjtu.edu.cn; tingliu@mail.xjtu.edu.cn

*Abstract*—The purpose of the General Data Protection Regulation (GDPR) is to provide improved privacy protection. If an app controls personal data from users, it needs to be compliant with GDPR. However, GDPR lists general rules rather than exact step-by-step guidelines about how to develop an app that fulfills the requirements. Therefore, there may exist GDPR compliance violations in existing apps, which would pose severe privacy threats to app users. In this paper, we take mobile health applications (mHealth apps) as a peephole to examine the status quo of GDPR compliance in Android apps. We first propose an automated system, named HPDROID, to bridge the semantic gap between the general rules of GDPR and the app implementations by identifying the data practices declared in the app privacy policy and the data relevant behaviors in the app code. Then, based on HPDROID, we detect three kinds of GDPR compliance violations, including the incompleteness of privacy policy, the inconsistency of data collections, and the insecurity of data transmission. We perform an empirical evaluation of 796 mHealth apps. The results reveal that 189 (23.7%) of them do not provide complete privacy policies. Moreover, 59 apps collect sensitive data through different measures, but 46 (77.9%) of them contain at least one inconsistent collection behavior. Even worse, among the 59 apps, only 8 apps try to ensure the transmission security of collected data. However, all of them contain at least one encryption or SSL misuse. Our work exposes severe privacy issues to raise awareness of privacy protection for app users and developers.

*Index Terms*—GDPR, Privacy policy, Data flow, GUI

## I. INTRODUCTION

The General Data Protection Regulation (GDPR) is an important data and privacy law, enforced since May 2018. The purpose of GDPR is to provide improved privacy protection based on a set of standardized data protection laws. For mobile apps, the GDPR applies to ones that collect and process personal data of European Union (EU) citizens. It does not matter if the app is operated from outside of the EU. The GDPR will still apply. Therefore, the GDPR is of considerable significance to mobile apps.

However, the regulation in GDPR itself does not contain any exact step-by-step guidelines about how to develop an app that fulfills all the requirements. It only gives us a list of the general rules that we must keep in mind when creating apps. Therefore, the semantic gap between the general rules and the app implementations may result in compliance violations

between GDPR and apps, which would pose severe privacy threats to app users. Once that happens, the developers would face administrative fines of up to to 20 million EUR, or in the case of an undertaking, up to 4% of the total worldwide annual turnover of the preceding financial year, whichever is higher [1].

In this paper, we take mHealth apps as a peephole to investigate the status quo of GDPR compliance in the Android apps, which would help developers identify and fix problems before releasing apps, and increase the apps' reliability when users prefer to download or use them. The rationale is that the mHealth apps, which are developed to perform health-related activities to help users monitor and manage their state of health, usually collect a broad range of more critical health-related information (PHI) [2] compared with the apps of other categories. Moreover, PHI is an important special category of personal data protected by GDPR.

We carefully read the articles in GDPR and summarize the following three necessary regulations based on three basic requirements for data protection. (see details in Section II-A).

• **Completeness of Privacy Policy.** The app should provide a complete privacy policy [3] to inform users about how their data is collected and used before app installation.

• **Consistency of Data Collection.** The app should not access more data than what its privacy policy declares.

• **Security of Data Transmission.** The data transmission of an app should adopt reasonable measures to keep secure.

However, it is difficult to investigate compliance violations with the three regulations due to three challenges.

First, for the detection of the incomplete privacy policy, since the policies are written in natural language without a uniform structure, it is difficult to understand the relations between the semantics declared in the privacy policy and the notices declared in GDPR. Most of the existing approaches only focus on the analysis of privacy policy with the natural language processing (NLP) techniques [4], [5], [6], [7] without considering the GDPR. To address this challenge, we combine the NLP techniques with the machine learning algorithms to generate six notice classifiers to detect whether a privacy policy is complete or not (see Section III-A for details).

Second, for the analysis of the inconsistent data collection, the PHI of mHealth apps is usually entered by users through the graphical user interface (GUI), making the related studies that only concentrate on the system-managed user data (e.g., device id, IMEI, and IP address) obtained by API calls inapplicable [8], [9], [10], [11], [12], [13], [14]. To address this challenge, we analyze the app GUI to recognize the PHI inputs and record its semantics based on a predefined PHI keyword database. Then, we leverage the static analysis technique to track the data flow of PHI to identify whether the mHealth app collects it by writing to files or sending out. Finally, we match the collected PHI with the those declared in privacy policy to identify whether there exists an inconsistent behavior of data collection (see Section III-B for details).

Third, for the identification of the data transmission security, it is challenging to map abstract cryptographic concepts (i.e., standard security rules) to concrete program properties. Existing approaches [15], [16], [17] can only check violations of security rules regardless of what data is protected; thus, they cannot be directly adopted by our work. Therefore, we initially analyze the tracked data flow information and identify which PHI is protected by cryptographic implementations. Then we study whether the implementations are compliant with standard security rules (see Section III-C for details).

We implement the above ideas in a new system called HPDROID to detect the GDPR compliance violations in mHealth apps. We conduct an empirical study on 796 real mHealth apps to examine whether they are compliant the regulations. The main contributions are as follows:

(i) To our best knowledge, this is the first systematic investigation on automatically detecting the compliance violations between the GDPR and mHealth apps.

(ii) We propose and develop HPDROID, an automated system to effectively detect whether mHealth apps are compliant with three privacy regulations summarized from GDPR.

(iii) We conduct an empirical evaluation with HPDROID on 796 real mHealth apps. The experimental results show that HPDROID can effectively detect the regulation violations for mHealth apps, which has exposed severe privacy issues to raise the awareness of privacy protection for the mHealth app users and developers.

## II. BACKGROUND AND PROBLEM DEFINITION

### A. GDPR

The GDPR agreed upon by the European Parliament and Council in April 2016, has replaced the Data Protection Directive 95/46/ec in May 2018 as the primary law [18]. It consists of 11 chapters and 99 articles that regulate how organizations collect, use, share, secure, and process their personal data and privacy of EU citizens for transactions that occur within EU member states. Organizations that fail to achieve GDPR compliance before the deadline (i.e., May 2018) will be subject to stiff penalties and fines. Note that even organizations outside the EU need to be compliant if they offer services to EU citizens. Never before have the needs of app users in this area been so forcefully and comprehensively protected. The seriousness of the GDPR should not be underestimated, which is why we have to take a fresh look at the compliance problem between the GDPR and Android apps. The analysis scope of this work focuses on three regulations that are summarized from three corresponding basic requirements in GDPR Article 5, i.e., transparency, data minimization, and confidentiality.

Transparency: the GDPR requires that *all the information you provide about your data processing needs to be easy to access and easy to understand*. Providing a privacy policy is an effective and necessary way to improve transparency [19]. Thus, we analyze the data processing transparency by detecting the completeness of privacy policy.

Data minimization: the GDPR requires that *personal data shall be adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed.* Thus, we analyze the consistency of data collection by detecting whether the app has accessed more data than what its privacy policy declares.

Confidentiality: the GDPR requires that *personal data shall be processed in a manner that ensures appropriate security of the personal data.* Thus, we check whether the data transmission of an app is adopted reasonable measures.

Note that there are many requirements defined in GDPR, and it is challenging to consider all the requirements. In this work, we only check three fundamental regulations based on the above three basic requirements for data protection. For other regulations, we leave them as our future work.

### B. Privacy Policy

Software that operates on personal data is often required to be accompanied by a *privacy policy*, which is a legal document and software artifact that describes consumer data practices [20]. For Android platform, the privacy policy is designed and uploaded to Google Play Store [21] by relevant developers for declaring what user data will be collected, why it will be collected, and how it will be used [3]. Fig. 1 presents the privacy policy of an app called *com.uevo.heartrate*. The privacy policy initially declares that the app will collect personal data from the users when they voluntarily choose to provide such data. Then, it describes that the app developers may share personal data with certain third parties without further notice to users. Finally, the developers provide their contact information. Note that the policies are generally ambiguous, resulting a challenge to directly understand the semantic meanings [22], [23].

### C. Research Questions

**RQ 1: Do the mHealth apps provide complete privacy policies?**

To improve the transparency of data processing, the mHealth app developers need to provide a privacy policy that at least:

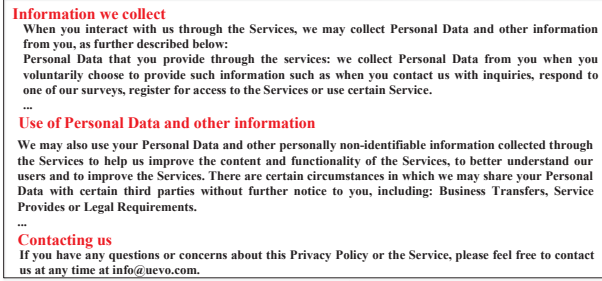- Indicates the precise categories of personal data that the app will process (**Data Collection**);

Fig. 1: A privacy policy of app called *com.uevo.heartrate*

- Describes the purpose of data processing, and how the data will be used and fitted in the products and services (**Data Usage**);
- Informs the user of their right to access and correct personal data, and to delete it (**User Right**);
- Informs the user that their use of the app is strictly voluntary, but requires their consent to permit the processing of personal data (**User Consent**);
- Informs that appropriate technical measures are adopted to protect personal data (**Data Security**);
- Provides contact information where the user can ask data protection related questions (**Contact Information**).

The first goal is to automatically detect whether a given privacy policy contains the six minimal notice specifications. We use $Notice_{pp}$ to denote the set of contained notice category labels of a privacy policy $pp$. Complete privacy policy means its $Notice_{pp}$ contains all the six labels, i.e., $|Notice_{pp}| = 6$. For example, by carefully scrutinizing the privacy policy illustrated in Fig. 1, we observe that it contains only three notice specifications and its $Notice_{pp} = \{$*Data Collection, Data Usage, Contact Information*$\}$. Therefore, the app does not provide a complete privacy policy. To achieve the goal, the main limitation is that the privacy policies are not written in a structured, commonly used, and machine-readable format, resulting that we cannot directly obtain $Notice_{pp}$.

**RQ 2: Do the mHealth apps declare all the collected PHI in their privacy policies?**

Every mHealth app must be designed to only collect and process PHI for its specific and legitimate purpose, which are clearly defined in the privacy policy. Here we define the PHI is collected by a mHealth app if it is input by users, and stored with different channels such as sending out through network or writing in local files. Collecting more PHI than what it declares is an illegal behavior to the app users. Furthermore, the data breach of such collected PHI could seriously affect the app users' profits and their confidence in the mHealth app.

The second goal is to automatically discover whether there is any PHI collected by a mHealth app but not declared in the privacy policy. To this end, we need to obtain two PHI sets. One is the set of declared collected PHI ($DCP$) extracted from the privacy policy. The other is the set of actually collected PHI ($ACP$) discovered from the app code. Thus, one inconsistent behavior is discovered if there is any item in $ACP$ that is not contained in $DCP$.

The $DCP$ can be constructed by analyzing the PHI items



Fig. 2: GUI of app called *com.sattva.sattvamanager*

declared in the privacy policy with NLP techniques. However, it is challenging to construct the $ACP$ because the PHI is generally input by users on app GUI [8], [9], [10]. For example, Fig. 2 presents the GUI of an app called *com.sattva.sattvamanager* that requires the users to provide the PHI such as symptoms, payment, and medicine. It is difficult for machines to automatically recognize the content of what user input due to the lack of fixed structures of the GUI.

**RQ 3: Do the mHealth apps implement reasonable measures to ensure the transmission security of their collected PHI?**

The GDPR requires the developers to provide appropriate technical safeguards to ensure the transmission security of collected PHI. Encryption is the most obvious determinant of security in mHealth apps communications [24]. However, even for the encrypted PHI, there might be encryption misuse that is not compliant with standard security rules.

Here, we make a list of the most common encryption misuses according to the existing studies [25], [26], [27], [17], [28]: ① Do not use electronic codebook (ECB) mode for encryption [25]; ECB mode uses a weak encryption algorithm that produces the same ciphertext from the same plaintext blocks, which would allow attackers to gain the sensitive data easily. ② Do not use MD5 or SHA-1 algorithms for encryption [26]; The modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords using massively-parallel computing. ③ Do not use a constant initialization vector (IV) [27] or constant keys [17] for encryption; The constant IV or keys result in a deterministic and stateless cipher, which would make the information insecure. ④ Do not use the static seed for *SecureRandom()* API call while generating random number [28]; Using the static seed is predictable and can result in the generation of random numbers with insufficient entropy.

Another common approach to protecting data during communication on the Android platform is to use the Secure Socket Layer (SSL) or Transport Layer Security (TLS) protocols. For brevity, we refer to both protocols as SSL. The inadequate use of SSL can be exploited to launch Man-in-the-Middle attacks [29], [30].

Note that the above encryption misuses are not specified in GDPR, but they are essential requirements that we need to satisfy to ensure transmission security, which is an important concept in GDPR.

The third goal is to identify whether there is unprotected PHI, and misuse of the technical safeguards. The main challenge is the mapping of abstract cryptographic concepts
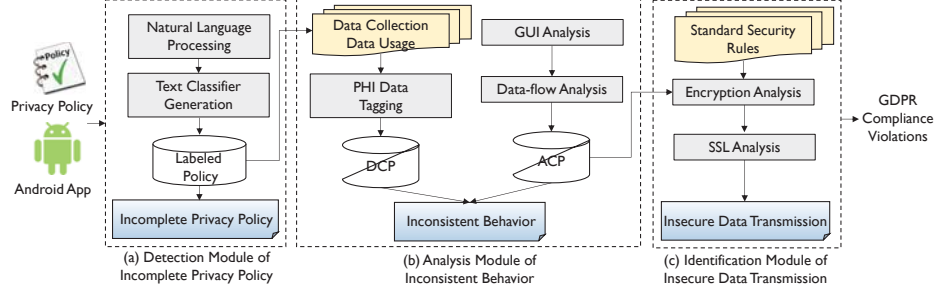
**Fig. 3:** Architecture of HPDROID.

introduced above to concrete program properties. Even existing approaches [15], [16], [17] have conducted related studies; they cannot be directly adopted here since they only check violations of security rules regardless of what data is protected.

## III. METHODOLOGY

The overview architecture of HPDROID is illustrated in Fig. 3, which consists of three main modules.

### A. Detecting Incomplete Privacy Policy

**Natural Language Processing.** Given a privacy policy crawled from the websites in HTML format, we use JSOUP [31], a Java HTML parser, to extract the content from the HTML file, and remove all non-ASCII symbols. Then we split the extracted content into a set of sentences using STANFORD TYPED DEPENDENCY PARSER [32]. After that, each sentence is formalized using the bag-of-words model [33] with the word stemming and removing of stop words. Finally, a sentence $st$ is represented as a feature vector, where each dimension corresponds to the occurrence of a separate word, and the number of dimensions denotes the total number of unique words extracted from privacy policy corpus. If a word occurs in the sentence, its value in the vector is 1; otherwise, the value is 0.

**Machine Learning Classification.** After the prepossessing of the privacy policy, we leverage the machine learning algorithms to predict what notices do the extracted sentences belong to. We use the term $Notice_{st}$ to denote the set of notice labels for sentence $st$. To generate the classifiers used for notice prediction, we first need to construct a ground truth dataset manually. In detail, we initially select 100 privacy policies of mHealth apps that are crawled from the Google Play Store and extract all corresponding sentences. Then, two co-authors go through these sentences and understand the semantics of each sentence, and manually construct their corresponding $Notice_{st}$. Note that, for each sentence, if the constructed $Notice_{st}$ by the two co-authors are not same, then a third co-author will check the result by having a discussion with them to guarantee the labeling correctness. In this way, we are able to obtain a ground truth dataset of sentences that are attached with notice labels. The description of the dataset is listed in Table I. Each notice category contains at least 100 sentences, and there are 1,284 labeled sentences in total. Note that the construction of labeled ground truth with manual

**TABLE I:** Descriptions of the constructed ground truth

| Notice Category | #Sentences | Notice Category | #Sentences |
|---|---|---|---|
| *Data Collection* | 385 | *User Consent* | 121 |
| *Data Usage* | 334 | *Data Security* | 176 |
| *User Right* | 164 | *Contact Information* | 104 |

intervention is once for all when training the classifiers. For the other modules, we do not need additional manual intervention.

Next, we construct a classifier for each notice prediction based on the dataset. For example, to construct the classifier used for *Data Collection* notice prediction, the 385 sentences that contain the *Data Collection* label are regarded as the positive instances. In addition, equal size of negative instances are randomly selected from other labeled sentences. After that, with the state-of-the-art machine learning algorithms such as Random Forest [34], the classifier used to predict the *Data Collection* notice can be generated.

**Incomplete Policy Detection.** After the generation of six classifiers, the feature vector of a new sentence will be put into the six classifiers in sequence. If the prediction result of a notice classifier is 1, then its corresponding notice label will be put into the $Notice_{st}$ of the given sentence. Finally, the $Notice_{pp}$ is obtained by merging all the generated $Notice_{st}$ to detect whether the privacy policy is complete.

### B. Analyzing Inconsistent Behavior

Before the construction of $DCP$ and $ACP$, it is essential to construct a set of PHI. To this end, we carefully read the GDPR recital 54 [35] and the National Health Data Dictionary provided by the Australian Institute of Health and Welfare [36], and then add the concrete PHI items into a set. We use $PS = \{ps_i | 1 \leq i \leq m\}$ to denote the full set of PHI, where $ps_i$ denotes the unique name of a PHI item in $PS$, and $m$ denotes the number of PHI items; $m = 227$ in this work.

*1) DCP Construction:* To construct $DCP$, we focus on the sentences with notice labels of *Data Collection* and *Data Usage* since we observe that all the data related operations are declared in them. We first leverage the STANFORD TYPED DEPENDENCY PARSER [32] to extract all the noun phrases. However, it is not effective to directly map the noun phrases with the items in the $PS$ due to the diversity of natural language. For example, the PHI called "doctor name" might be written as "name of doctor" in the privacy policy.

**Phrase Similarity Calculation.** To measure the similarities between the semantics of PHI with the noun phrases extracted from the privacy policy, we rely on the tool called

256

**Fig. 4:** Two examples of user input widgets and their associated metadata.

WORD2VEC [37] to transform the PHI items and the noun phrases into a calculable form. As a result, the similarities can be obtained based on the cosine similarity. If the similarity is higher than the threshold $\epsilon$, which is set as 0.85, then $ps_i$ is added into the $DCP$. Note that the parameter 0.85 is preset based on our experience analysis on a set of similar phrases.

*2) ACP Construction:* The construction of $ACP$ relies on two kinds of techniques: the GUI analysis technique, which is used to identify the user input PHI; the data-flow analysis technique, which is used to filter out the non-collected PHI.

**GUI Analysis.** To analyze the user input PHI from the GUI of Android app, we need to render all the GUI layouts and identify the semantics of the user inputs. Note that our technique is extended based on UiRef [38]. First, an APK file is disassembled using APKTOOL [39]. A file called *public.xml* is generated to store all the resource identifiers of layout widgets. Subsequently, a customized activity is injected into the APK, and the manifest file of the app is rewritten to register the injected activity as an entry point. After that, the APK is reassembled and installed on a live device. When the injected activity is launched, the layouts of the app are rendered iteratively by invoking the *setContentView()* API call. However, the dynamically generated text (e.g., label text set by the *setText()* API call) cannot be extracted in this way using UiRef [38]. To solve the problem, we also launch the activities declared in the *AndroidManifest.xml* file to render the corresponding layouts. Once a current layout is loaded, its view hierarchy is dumped by UIAUTOMATOR [40]. Then, the associated metadata of UI widgets are extracted from the view hierarchy, and each widget is represented as a four tuples form $\langle class, id, text, bound \rangle$, where

- $class$ denotes the widget type such as *EditText*.
- $id$ denotes the widget id stored in the *public.xml* file.
- $text$ denotes the plain text presented in the widget.
- $bound$ denotes the coordinate of the displayed widget.

Next, we need to understand the semantics of the inputs. In general, there are two methods for developers to present the semantics of the inputs to help users understand what they are required to provide. The first method is presenting the semantics based on the hint text of the user input widget. The second method is leveraging a label widget to present the semantics. We call the two methods as the hint-based method and the label-based method.

Two examples are illustrated in Fig. 4. The first example presents an *EditText* that uses the hint text "Add symptoms here" belongs to the hint-based method. The second example presents an *EditText* that shows semantics with a combined *TextView* belongs to the label-based method.

For the hint-based method, we analyze the $text$ value of the user input widget. The string value of the $text$ is preprocessed

---

**Algorithm 1:** Mapping of Labels and Input Widgets

**Input:** $LB$: the set of labels displayed in an UI; $UIW$: the set of user input widgets displayed in an UI.

**Output:** $M$: the set of output label and input widget mapping pairs.

1 **foreach** *input* in $UIW$ **do**
2      $LeftSet \leftarrow$ ConstructLeftLabelSet($input$, $LB$)
3      **if** $LeftSet.size()>0$ **then**
4          $mapLabel \leftarrow argmin_{label \in LB}$ dis($label$, $input$)
5          $M$.put($input$, $mapLabel$)
6          $LB$.remove($mapLabel$) and $UIW$.remove($input$)
7      **else**
8          $UpSet \leftarrow$ ConstructUpLabelSet($input$, $Label$)
9          **if** $UpSet.size()>0$ **then**
10             $mapLabel \leftarrow argmin_{label \in LB}$ dis($label$, $input$)
11             $M$.put($input$, $mapLabel$)
12             $LB$.remove($mapLabel$) and $UIW$.remove($input$)

13 **return** $M$

---

by the NLP technique. Then it is split into a set of words. With such words, we construct a set of unigram phrases and a set of bigram phrases. After that, each phrase in the two sets is matched with the PHI items in $PS$ based on the introduced phrase similarity calculation method. If $ps \in PS$ is matched with a phrase, $ps$ is regarded as one user input PHI. If no $ps$ is matched, the given widget might use the label-based method.

For the label-based method, the main challenge is to map the labels with their combined user input widgets. In our work, we propose algorithm 1 to map the labels with user input widgets. Algorithm 1 requires two inputs, $LB$ and $UIW$. For each input widget in the $UIW$, it is checked whether there are any labels in $LB$ that are closely placed on its left or above by using the function ConstructLeftLabelSet() and ConstructUpLabelSet(). The positional relations between the widgets are calculated based on their extracted $bound$ values. Finally, for each pair in the output $M$, the $text$ value of the label will be matched with the PHI items in the same way as solving the hint-based way to identify the user input PHI.

**Data-flow Analysis.** Note that the user input PHI cannot be directly added into the $ACP$ since the app might not store them. Thus, in our approach, the data-flow analysis technique is used to recognize the point of getting specified data and to further check the destination of fetched data. We conduct the data flow analysis based on static analysis tools such as FLOWDROID [41] and VULHUNTER [42], which are implemented based on the Soot framework [43]. Notably, to enhance the static analysis accuracy, ICCTA [44] is employed to identify the source and the target of intents, and EDGEMINER [45] is utilized to determine the implicit callbacks (e.g., from *setOnClickListener()* to *onClick()*). The data-flow analysis includes three main parts, listing as below:

- **Data Sources.** The data sources are the points that we obtain the PHI which will be tracked. We focus on the user input PHI and the API call *findViewById()* is selected as the data source.
- **Data Propagation.** To track the data propagation in the app code, we leverage the taint propagation techniques [41]. In detail, the data sources are initially assigned with

257

a unique taint tag. Then, the taint tag will be propagated based on the direct data flow propagations according to the intermediate representation extracted by Soot.

- **Data Sinks.** The data sinks are the data use points of the tainted variables. There are six different kinds of data storage methods, including writing data into a log (e.g., *Log.d()*) or a file (e.g., *FileoutputStream.write()*), or sending data out through network (e.g., *HttpClient.execute()*) or short messages (e.g., *SmsManager.sendTextMessage()*), or inserting the data into a database (e.g., *SQLite-Database.update()*) or the content provider (e.g., *ContentResolver.insert()*).

Note that, we also need to link the data flows with their corresponding widget objects. We resolve the argument value of the *findViewById()* API call, and the argument value demonts the *id* of the specific wedget object.

In summary, if an app collects one PHI and stores the data with the above six methods, the PHI is added into $ACP$. Finally, one inconsistent behavior is discovered if there exists an item in $ACP$ that is not contained in $DCP$.

### C. Identifying Insecure Data Transmission

*1) Encryption Analysis:* For the identification of the use of system-provided encryption algorithms, we observe that the symmetric encryption (e.g., AES algorithm) and asymmetric encryption (e.g., RSA algorithm) schemes are generally accessible to an app through the *Cipher* object by using the *doFinal()* API call. In addition, the one-way encryption schemes (e.g., MD5 and SHA-1 algorithms) are generally accessible to an app through the *MessageDigest* object by using the *digest()* API call. Therefore, we use such encryption API calls as data sinks and apply data-flow analysis techniques to detect whether there exists complete data flow from the data sources (i.e., *findViewById()*) to any encryption API calls. If no complete data flow is found, we regard that such PHI is not protected with system-provided encryption algorithms.

Although there are security pieces of advice in the official documents and an extensive body of security-related research about exploits and vulnerabilities, using encryption algorithm correctly is still not easy for inexperienced or distracted developers [46]. We assess the four security rules (introduced in Section II-C) on the mHealth apps by checking their corresponding program properties.

To evaluate the rule ①, we resolve the *Cipher.getInstance()* API call to find what transformation string is specified by the developers to be used as arguments of the API call. If the string "ASE" is used as the argument, the encryption mode is automatically chosen as ECB mode. To improve security, another encryption mode with padding such as "AES/CBC/PKCS5Padding" should be used.

To evaluate the rule ②, we initially check whether the *digest()* API call is used as the data sink. If so, we then resolve the *MessageDigest.getInstance()* API call to find whether its argument is specified with string "MD5" or "SHA-1".

To evaluate the rule ③, we compute the backward slices for the arguments of *IvParameterSpec()* and *SecreKeySpec()* API

calls, and then determine whether the used arguments consist of constant values. If the slices only depend on constant values, the IV or the keys are constant too.

To evaluate the rule ④, we construct a backward slice from each call site to the *SecureRandom()* API to check whether the developers specify the seed value.

*2) SSL Analysis:* SSL is another common approach to protect data during transmission on the Android platform, in which the *java.net*, *android.net* and *org.apache.http* packages can be used to create sockets or HTTP(S) connection. However, as introduced in [47], a large number of apps implement SSL with inadequate validation such as containing code that allows all hostnames or accepts all certificates. Insecure SSL transmission is dangerous since they would generally carry critical sensitive data such as the collected PHI. To detect the usage of SSL analysis in mHealth apps, we focus on the PHI that is sent out through the network with sink API calls in the three network-related packages. Then we identify the SSL misuse by using a static analysis tool called MALLODROID [16], which can automatically check SSL security risks in apps.

## IV. EVALUATION

### A. Data Collection

To evaluate HPDROID we initially crawl 1,200 popular real mHealth apps from *Medical* and *Health* categories on the *Google Play* [21] according to their download counts. Then we remove the apps that do not contain a privacy policy written in English language. Finally, there are 796 mHealth apps remained. After that, to answer **RQ 1**, we use the constructed ground truth dataset (i.e., 1,284 labeled sentences in total) introduced in Section III-A to generate notice classifiers, and then use the classifiers to detect whether the unlabeled policies are complete or not. To answer **RQ 2**, we focus on the apps that require users to input PHI on the GUI. After the removing of the apps that are only used to introduce health-related knowledge based on the GUI analysis step, 253 remaining mHealth apps (31.78% of 796 apps) are analyzed in the analysis module of inconsistent behavior. To answer **RQ 3**, the 59 apps that collect PHI are put in the identification module of insecure data transmission to check whether they have implemented reasonable PHI data protection measures.

In addition to the mHealth apps, we also need to construct the set of sink API calls. Based on the API list provided by SUSI [48], we manually remove the API calls that do not belong to our six defined data storage methods. In the end, 78 sink API calls are used.

The apps and their privacy policies, as well as the generated data flow information, can be found on our website[1].

### B. RQ 1: Do the mHealth apps provide complete privacy policies?

*1) Performance on Labeled Dataset:* A ground truth dataset that consists of 1,284 labeled sentences is used to construct

[1]https://drive.google.com/drive/folders/18qaSTuHcm_
2LLBsMM70Y9VwZrfvi686t?usp=sharing

**TABLE II:** Classification results for the notice categories in labeled dataset.

| Notice Category | Classifier | TPR | FPR | Precision | Recall | F-1 |
|---|---|---|---|---|---|---|
| Data Collection | **RF** | **0.937** | **0.132** | **0.877** | **0.937** | **0.906** |
| | NB | 0.882 | 0.153 | 0.852 | 0.882 | 0.867 |
| | DT | 0.789 | 0.192 | 0.804 | 0.789 | 0.797 |
| Data Usage | **RF** | **0.884** | **0.154** | **0.852** | **0.884** | **0.867** |
| | NB | 0.865 | 0.164 | 0.841 | 0.865 | 0.853 |
| | DT | 0.830 | 0.214 | 0.795 | 0.830 | 0.812 |
| User Right | **RF** | **0.950** | 0.157 | 0.858 | **0.950** | **0.901** |
| | NB | 0.862 | **0.113** | **0.884** | 0.862 | 0.873 |
| | DT | 0.836 | 0.176 | 0.826 | 0.836 | 0.831 |
| User Consent | RF | 0.878 | 0.104 | 0.902 | 0.878 | 0.890 |
| | NB | 0.887 | 0.130 | 0.872 | 0.887 | 0.879 |
| | **DT** | **0.922** | **0.035** | **0.964** | **0.922** | **0.942** |
| Data Security | **RF** | **0.914** | **0.018** | **0.980** | **0.914** | **0.946** |
| | NB | 0.908 | 0.043 | 0.955 | 0.908 | 0.931 |
| | DT | 0.896 | 0.055 | 0.842 | 0.896 | 0.918 |
| Contact Information | RF | 0.980 | 0.020 | 0.980 | 0.980 | 0.980 |
| | NB | 0.980 | **0.010** | **0.990** | 0.980 | 0.985 |
| | **DT** | **0.990** | **0.010** | **0.990** | **0.990** | **0.990** |
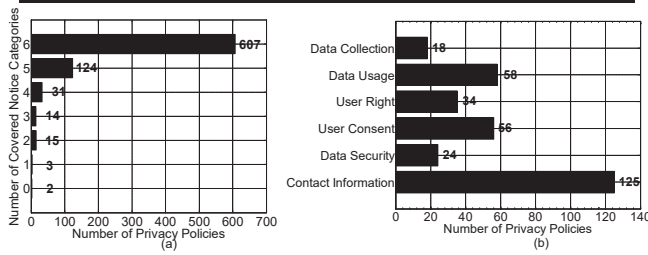


**Fig. 5:** Completeness detection results for 796 unlabeled privacy policies: (a) the number of privacy policies that cover different notice categories; (b) the number of privacy policies that lack of different notice categories.

six notice classifiers. To select the proper classifier with best detection performance, we apply three widely-used classification algorithms [10] (i.e., Decision Tree (DT) [49], Random Forest (RF) [34] and Naive Bayes (NB) [50]) on each notice category with 10-fold cross-validation. Table II lists the results. Using F-1 value to measure the classifier performance, Random Forest performs best in four notice categories while Decision Tree performs best in the remaining two notice categories. Consequently, the Decision Tree classifier is used to detect the *User Consent* and *Contact Information* categories, while the Random Forest classifier is used to detect the others.

*2) Detection Results on Unlabeled Dataset:* We apply the constructed classifiers on the 796 unlabeled privacy policies to detect whether they are complete. As illustrated in Fig. 5 (a), 607 (76.3%) privacy policies cover all the notice categories and the other 189 (23.7%) privacy policies are incomplete, of which 34 policies cover no more than three notice categories. Notably, there are two privacy policies (i.e., *com.bytewaremobile.oasi* and *com.app.tctnews*) that do not cover any notice category, indicating that they do not provide any useful information for app users. We manually inspect the privacy policy links provided by the two apps, and find that they redirect to other websites that do not contain any privacy policy related content. Fig. 5(b) presents the number of privacy policies that lack different notice categories. For example, 58 privacy policies that do not cover *Data Usage* notice. More interesting, we find that 48 of the 58 policies have the *Data Collection* notice, indicating that such privacy policies collect personal information but do not illustrate



**Fig. 6:** An example of the incomplete privacy policy of app *com.bodyyouwant* what purpose the information is used for, which violates the transparency requirement of GDPR. In addition, 125 (15.7%) privacy notices do not cover the *Contact Information* notice category, which indicates the users could not contact with the app developers if they have any problems about the app usages. The main reason might be that the developers have provided emails on the app downloading page; thus, they think it is not necessary to provide again in the privacy policy.

*3) False Positives and False Negatives:* We further investigate the false positives and false negatives of the constructed classifiers. To this end, we randomly select 100 sentences from the unlabeled dataset. Then we carefully read the sentences and check whether their attached labels are correct. We find that 7 sentences are incorrectly classified due to the features with high information gain (e.g., "we", "please", and "collect") occur in more than two categories. For example, the sentence *"By using our website, you agree that we can place cookies on your computer/device."* is attached to labels *User Consent* and *User Right*. The label *User Right* is incorrectly attached since the sentence contains the keywords "you" and "can" similar to the structures of training sentences in *User Right* category.

Moreover, four sentences are found as false negatives because their verbs do not occur in our labeled training dataset. For example, the verb "forward" in the sentence *"The data have never been and will not be forwarded to third parties."* is not found in the training dataset. To reduce this threat, replacing the uncommon verbs with the common ones through sentence semantic analysis is a promising way.

*4) Case Study of the Incomplete Privacy Policy:* As shown in Fig. 6, the privacy policy of app *com.bodyyouwant* is attached with three labels, i.e., *Data Collection*, *Data Usage*, and *Contact Information*. It does not declare that it will ask for the users' consent before collecting user information, indicating that the collection process is non-transparent to the users. Furthermore, there is no user right described in the policy, indicating that the user does not know what they can do with the collected data. Such incomplete privacy policy is not clear to the app users and violates the GDPR. However, HPDROID is significant to guide app developers to provide complete policy and help app users quickly understand the semantics of important notices in the expatiatory policy.

> **Answer to RQ1**: For the 796 mHealth apps, 189 (23.7%) of them do not provide complete privacy policies. The incomplete privacy policy violates the GDPR and poses privacy issues for app users in the real world. It is imperative for app developers to complete existing incomplete privacy policies.

*C. RQ 2: Do the mHealth apps declare all the collected PHI in their privacy policies?*

To answer this research question, we only focus on the 253 mHealth apps that require users to input PHI.

**TABLE III:** Distribution of the collected PHI for 59 mHealth apps that actually collect PHI. The numbers of the non-declared PHI are listed in the brackets

| PHI | Log | File | Network | SMS | Database | Content | Total | PHI | Log | File | Network | SMS | Database | Content | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | 23 | 7 | 7 | 0 | 3 | 1 | 41(17) | symptom | 1 | 0 | 0 | 0 | 0 | 0 | 1(1) |
| email | 17 | 1 | 9 | 0 | 0 | 0 | 27(7) | amount | 1 | 0 | 0 | 0 | 0 | 0 | 1(0) |
| password | 13 | 4 | 4 | 0 | 0 | 0 | 21(16) | patient | 0 | 1 | 0 | 0 | 0 | 0 | 1(1) |
| phone | 6 | 1 | 5 | 1 | 0 | 0 | 13(6) | pregnancy | 0 | 0 | 0 | 0 | 0 | 1 | 1(1) |
| weight | 9 | 1 | 0 | 0 | 0 | 0 | 10(7) | gender | 1 | 0 | 0 | 0 | 0 | 0 | 1(0) |
| location | 1 | 3 | 1 | 0 | 3 | 1 | 9(5) | reason | 1 | 0 | 0 | 0 | 0 | 0 | 1(1) |
| height | 3 | 1 | 0 | 0 | 1 | 0 | 5(4) | activity | 1 | 0 | 0 | 0 | 0 | 0 | 1(1) |
| duration | 1 | 1 | 2 | 0 | 1 | 0 | 5(5) | glucose | 0 | 1 | 0 | 0 | 0 | 0 | 1(1) |
| description | 1 | 1 | 0 | 0 | 2 | 1 | 5(5) | fat | 1 | 0 | 0 | 0 | 0 | 0 | 1(1) |
| note | 1 | 0 | 0 | 0 | 3 | 0 | 4(4) | account | 0 | 0 | 0 | 0 | 0 | 1 | 1(1) |
| date | 2 | 0 | 2 | 0 | 0 | 0 | 4(4) | doctor | 1 | 0 | 0 | 0 | 0 | 0 | 1(0) |
| age | 2 | 0 | 0 | 0 | 0 | 0 | 2(2) | registration | 0 | 0 | 1 | 0 | 0 | 0 | 1(1) |
| medication | 2 | 0 | 0 | 0 | 0 | 0 | 2(1) | **Sum** | 88 | 22 | 31 | 1 | 13 | 5 | 160(92) |

*1) Result of Inconsistent PHI Collection:* After the data-flow analysis, 59 of the 253 apps collect the user input PHI by storing them with six different channels. To investigate the reasons for the other 194 apps that require the PHI input but have no PHI collection, we randomly select 20 apps from the 194 apps and manually analyze the app code. We find that there are two main reasons. First, most of the input PHI is only used to perform calculations such as health state, and the mHealth apps will not store the PHI with the six channels. Second, PHI collections might be missing due to the limitations of the existing data-flow analysis techniques. For example, the Android annotation technique [51] makes HPDROID fail to link the *findViewById()* with the widget objects. We will discuss this limitation in Section V.

We use the terms *Log*, *File*, *Network*, *SMS*, *Database* and *Content* to denote each data storage method, respectively. The frequency distribution of the collected PHI is listed in Table III. There are 160 PHI collection behaviors for the 59 apps, in which name, email, and password are the most common ones. Among the six data storage methods, *Log* is the most common method since 55% of the collected PHI is written into logs, *SMS* is the least-used method because there is only one phone number collection behavior via sending messages.

After the construction of $DCP$ by tagging the declared PHI in the *Data Collection* and *Data Usage* sentences, we match them with the collected PHI in $ACP$. The results show that 46 apps have collected more PHI than what they declared in their privacy policies. The numbers of the non-declared PHI are listed in the brackets of the **Total** column in Table III. For example, 21 apps collect the password of the users, but 16 of them do not declare the password collection in their privacy policies. In total, 92 inconsistent behaviors are discovered for the 46 apps. We observe that the app developers prefer to declare the common data, such as name, email, password, and phone number. However, for the uncommon but important data, such as duration, description, and date, they do not declare such data collection behaviors in the privacy policy. The inconsistent data collections violate the GDPR and mislead the app users. Therefore, we want to alert the app developers to provide consistent privacy policy by conducting the inconsistent behavior analysis.

*2) False Positives and False Negatives:* We further investigate the false positives and false negatives for inconsistent behavior analysis. For the false positive analysis, we initially
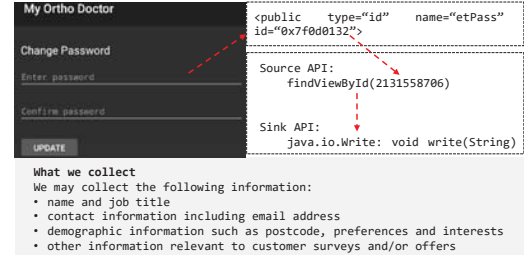


**Fig. 7:** An inconsistent behavior of app *com.app.app889ee1ec94b3*

read all privacy policies of the 46 mHealth apps that contain inconsistent behaviors. Then we check whether each inconsistent behavior is correct. The results show that among the 46 apps, no false positive is found. For the false negative analysis, since there is no ground truth for inconsistent behavior discovering, we manually analyze the 13 mHealth apps with no inconsistent behaviors to determine whether our approach results in false negatives. The results show that all the collected PHI has been declared in their privacy policies. Therefore, there is no false negative.

*3) Case Study of the Inconsistent Behavior:* Furthermore, we conduct a case study of inconsistent behavior, which is illustrated in Fig. 7. The app provides the functionality of changing the password for app users. By analyzing the attributes of *EditText* widget, we obtain its hint value "Enter password" and *id* "0x7f0d0132". Note that the hexadecimal number "0x7f0d0132" is equal to the decimal number "2131558706". After that, based on the data-flow analysis, the input password is written in a file through the sink API *java.io.Write: write()*. The writing of input password to file poses great threats to app users. Even worse, the app developers do not declare that they collect password in their provided privacy policy, which violates the data minimization requirement of GDPR. Note that although they declare that other information relevant to the users is collected, this kind of vague description is also not transparent to app users and it should be specified clearly.

**Answer to RQ2**: Among the analyzed 253 apps, 59 apps collect PHI via different methods. However, 46 (77.9%) of them contain at least one inconsistent collection behavior. The app developers should not use vague descriptions about data collection, which might cause the inconsistent data collection behavior that poses great threats to the app users and seriously violates the GDPR.

## D. RQ 3: Do the mHealth apps implement reasonable measures to ensure the transmission security of collected PHI?

To identify whether the collected PHI for the 59 apps is protected with reasonable measures such as data encryption or SSL, we analyze the correctness of encryption usage for the 49 apps that collect information via five methods (i.e., *Log*, *File*, *SMS*, *Database*, and *Content*), and check the correctness of SSL usage for the other 10 apps that collect information through *Network*.

*1) Result of Encryption and SSL Analysis:* After the detection of encryption function, we observe that only 2 apps (i.e., *com.ysp.l30band* and *kalcare.dsc*) have adopted system-provided encryption API calls to protect collected data. For exmaple, *com.ysp.l30band* encrypts the input password and email with *MessageDigest.digest()* API call. By tracking the argument of the *MessageDigest.getInstance()* API call we observe that both the password and the email are encrypted with MD5 algorithm, which breaks the security rule "Do not use MD5 or SHA-1 algorithms for encryption." For the incorrectness encryption usage of app *kalcare.dsc*, we leave it as a case study later.

Then, by applying MALLODROID [16] on the 10 apps storing collected PHI through networks, we observe that 4 apps do not use SSL. For the other 6 apps that adopt SSL, each of them contains at least one SSL misuse such as trusting all certificates or allowing all hostnames. For example, the app called *com.pumapumatrac* makes a blank implementation of the *TrustManager* interface so that it will trust all the server certificates (regardless of who signed it, what is the CN etc.). Furthermore, it even requires the use of *SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER*. As a result, hostname verification should take place when establishing an SSL connection is disabled.

*2) False Positives and False Negatives:* We also investigate the false positives and false negatives for insecure data transmission identification. Since our results demonstrate that the data transmission of all the 59 mHealth apps is insecure, there is no false negative. To evaluate whether there exist any false positives, we manually check all the data flow information of the collected PHI for the apps that do not use encryption function or SSL. We observe that all the PHI is stored in plaintext. Furthermore, we manually check the 6 apps that use SSL and do not find any false positives.

*3) Case Study of the Insecure Data Transmission:* Finally, we take the encryption detail of app *kalcare.dsc* as a case study. *kalcare.dsc* encrypts four kinds of PHI, including name, phone, email, and password. The collected name and phone are encrypted with *MessageDigest.digest()* API call and the collected email and password are encrypted with *Cipher.doFinal()* API call. By tracking the argument of *MessageDigest.digest()* we observe that SHA-256 algorithm is used. For the use of *Cipher.doFinal()* API call, we further check its compliance with the security rules. The code snippets of encrypting email and password with the *doFinal()* API call are presented in Fig. 8. The argument of *Cipher.getInstance()* is "DESede/CBC/PKCS5Padding," which indicates that the

```
1   public class CipherUtil{
2       private static byte[] sharekey;
3       private static byte[] sharedvector;
4       static{
5           CipherUtil.sharedkey=new byte[]{1,2,3,5,…};
6           CipherUtil.sharedvector=new byte[]{1,2,3,5,…};
7       }
8       public static String encrypt(String arg6){
9           ……
10          Cipher v0=Cipher.getInstance("DeSede/CBC/PKCS5Padding");
11          v0.init(1, new SecretKeySpec(CipherUtil.sharedkey, "DESede"),
12                      new IvParameterSpec(CipherUtil.sharedvector));
13          v2=String.copyValueOf(Base64Coder.encode(v0.doFinal
14                      (arg6.getBytes("UTF-8"))));
15      }
16      return v2;
17  }
```

**Fig. 8:** Code snippets of encrypting data with the *doFinal()* API call in app *kalcare.dsc*.

adopted encryption algorithm is DESede and the encryption scheme is CBC mode. Thus, this app obeys the rule "Do not use ECB mode for encryption" but does not follow the rule "Do not use a constant IV or constant keys for encryption" as both the arguments of *SecretKeySpec()* and *IvParameterSpec()* are static constants listed in line 5 and line 6, which would cause data more subject to attacks.

> **Answer to RQ3**: For the 59 mHealth apps that collect user input PHI, only 8 of them try to ensure the PHI transmission security with reasonable measures, i.e., 2 apps protect data with encryption algorithms and 6 apps adopt SSL protocol. However, all of the 8 apps contain at least one kind of encryption or SSL misuse. The security of the collected PHI data has not been heeded enough, which would cause serious data breaches.

## V. DISCUSSIONS AND THREATS TO VALIDITY

### A. Lessons Learned from Results

Providing transparent and accessible privacy policy about the personal data is the essential requirement for organizations that fall under the scope of GDPR. However, according to our evaluation result, 189 (23.7%) do not provide complete privacy policies. The main reason is that GDPR introduces several new privacy requirements compared with old regulations. For example, one new requirement is that apps must acquire the user's active and informed consent before any personal data is collected (i.e., *User Consent* notice in this work). However, up to now, many apps would assume that a user's decision to proceed with app registration and use is equivalent to having the user's consent to collect data. Lacking any notice introduced in our work would not meet the transparency requirement of GDPR. By using our tool, app developers can discover the missing notices and complement them. For app users, they can quickly understand the semantics for the most important data processing related content, so well as their rights when using the app.

Data minimization is another essential requirement of GDPR. Data processing should only use as much data as is required to successfully accomplish a given task. However, in our work, we find that 46 apps contain at least one inconsistent data collection behavior. We manually check the policies and find there are two reasons: First, 59 inconsistent behaviors occurred in 26 apps are caused by the vague description such as "other information"; Second, the other 33 inconsistent behaviors occurred in the other 20 apps are mainly caused

by the app developers' intentional ignorance (e.g., they think such data is not important, or they want to collect such data without informing app users). To mitigate the occurrence of inconsistent data collection behaviors, by leveraging our tool, the app developers can first list all their collected data. Then they can specify such collected data in their privacy policy clearly. For app users, they can have a clear understanding of which personal data are collected by app developers and how they are processed in the app.

Confidentiality is the only principle that deals explicitly with security in GDPR. Meanwhile, it is the most concerned one by users since there are more and more data breach events in recent years. Based on our evaluation results, although most apps declare that they would implement reasonable measures to keep the data secure, only 8 apps try to adopt security measures, and all of them contain at least one kind of misuse. The evaluation results indicate that the app users' data might be leaked in high probability, which is amazing to us.

### B. Threats to Validity

**Data-flow Analysis.** Even we use the popular static analysis tools, including FLOWDROID, ICCTA, and EDGEMINER, to tract the data flow, there still exist false negatives, which might cause the missing of PHI collection. The existing of false negatives further unveils that the status quo of GDPR compliance violations in Android apps is worse than what we demonstrate. Combining with dynamic analysis [52], [53] is a promising way to solve this problem.

**Self-implemented Encryption Detection.** For the encryption analysis, our approach would fail if the self-implemented encryption function is applied since we only rely on the study of system-provided encryption algorithms. A promising way to address this limitation is to compare the data entropy before and after the invocation of possible encryption function while the app is running [54]. If the data entropy is much higher after the function invocation, then the data might be encrypted in the corresponding function.

**Ground Truth Dataset:** We use a triple module redundancy approach when preparing ground truth dataset. However, if three authors fail to achieve a consistency, we would not add the sample into our dataset. The lacking of such sentences in training set would affect the classifier performance when dealing the similar sentences.

## VI. RELATED WORK

**Privacy Policy Analysis.** Several studies focus on the privacy policy analysis in recent years [55], [56], [57], [10]. Slavin *et al.* [9] proposed a semi-automated framework for detecting the violations based on a privacy-policy-phrase ontology and a collection of mappings from API calls to policy phrases. Yu *et al.* [8] proposed PPCHECKER that focuses on system-managed data and identify three kinds of problems in the privacy policy. The most related work is proposed by Wang *et al.* [58], who automatically detect privacy leaks of user-entered data for a given Android app and determines whether such leakage may violate the app's privacy policy claims. There are three main

differences between our work and the above studies: 1) We combine the analysis of mHealth apps with the GDPR while [9], [10], [8], [58] do not consider. 2) We focus on the PHI input by the users on GUI rather than the system-managed data analyzed by [9], [10], [8]. 3) We further investigate the transmission security of collected data while [58] does not.

**GUI Analysis.** A few studies are focusing on the analysis of GUI [59], [60], [38], [61], [62], [63], [64]. The most related works are UIPICKER [62], SUPOR [61], UIREF [38] and GUILEAK [58], the goals of which are identifying the sensitive user input information on the GUI. UIPICKER [62] and GUILEAK [58] use sibling relationships in layouts to find the associated labels and input widgets. However, in practice, sibling relationships do not accurately gauge proximity. Both SUPOR [61] and UIREF [38] select the optimal label by calculating the distances between the labels and the input widgets based on the positions displayed on the screen.

**GDPR Compliance Checking.** Several recent works focus on the GDPR compliance checking [65], [66], [67], [68], [69]. However, their methodologies are quite different from ours. Torre *et. al* [65] proposed a model-based GDPR compliance analysis solution using unified modeling language (UML) and object constraint language (OCL). Torre *et. al* [66] provided an automated support for checking whether the content of a given privacy policy is complete according to the provisions stipulated by GDPR. Due to a different research goal, our paper focuses on a specific subset of GDPR privacy policy requirements, and so we consider 6 out of the 55 categories that are presented in [66]. Palmirani and Governatori [67] presented a proof-of-concept applied to the GDPR domain, with the aim to detect infringements of privacy compulsory norms or to prevent possible violations using BPMN and Regorous engine. These existing approaches conduct compliance checking from the perspective of modeling. We go beyond the above approaches by transforming the GDPR requirements into specific regulations and combine with program analysis techniques so that we can conduct a fine-grained empirical evaluation on real mHealth apps.

## VII. CONCLUSION

We develop a system called HPDROID based on existing techniques and conduct the first systematic investigation on automatically detecting the compliance violations between the GDPR and mHealth apps. The experimental results on 796 real mHealth apps reveal that most of the apps are not compliant with the GDPR, which would raise the awareness of the privacy protection for the mHealth app users and developers.

REFERENCES

[1] "Fines and penalties," https://www.gdpreu.org/compliance/fines-and-penalties/, 2019.

[2] "National health data dictionary: version 16.2," https://www.aihw.gov.au/reports/australias-health/national-health-data-dictionary-version-16-2/contents/table-of-contents, 2015.

[3] "Privacy policy guidance," https://developers.google.com/actions/policies/privacy-policy-guide, 2017.

[4] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie, "Automated extraction of security policies from natural-language software documents," in *Proc. FSE*, 2012, p. 12.

[5] E. Costante, J. den Hartog, and M. Petković, "What websites know about you," in *Proc. DPM*, 2013, pp. 146–159.

[6] C. A. Brodie, C.-M. Karat, and J. Karat, "An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench," in *Proc. SOUPS*, 2006, pp. 8–19.

[7] M. Fan, X. Luo, J. Liu, C. Nong, Q. Zheng, and T. Liu, "Ctdroid: leveraging a corpus of technical blogs for android malware analysis," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 124–138, 2020.

[8] L. Yu, X. Luo, X. Liu, and T. Zhang, "Can we trust the privacy policies of android apps?" Proc. DSN, 2016, pp. 538–549.

[9] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, and J. Niu, "Toward a framework for detecting privacy policy violations in android application code." Proc. ICSE, 2016, pp. 25–36.

[10] 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." Proc. NDSS, 2017.

[11] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu, "Dapasa: detecting android piggybacked apps through sensitive subgraph analysis," *IEEE TIFS*, vol. 12, no. 8, pp. 1772–1785, 2017.

[12] M. Fan, J. Liu, X. Luo, K. Chen, T. Chen, Z. Tian, X. Zhang, Q. Zheng, and T. Liu, "Frequent subgraph based familial classification of android malware," in *Proc. ISSRE*, 2016, pp. 24–35.

[13] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE TIFS*, vol. 13, no. 8, pp. 1890–1905, 2018.

[14] M. Fan, X. Luo, J. Liu, M. Wang, C. Nong, Q. Zheng, and T. Liu, "Graph embedding based familial analysis of android malware using unsupervised learning," in *Proc. ICSE*, 2016, pp. 771–782.

[15] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications." Proc. CCS, 2013, pp. 73–84.

[16] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl (in) security." Proc. CCS, 2012, pp. 50–61.

[17] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl, "A stitch in time: Supporting android developers in writing secure code." Proc. CCS, 2017, pp. 1065–1077.

[18] "General data protection regulation (gdpr)," https://gdpr-info.eu/, 2019.

[19] "We help with the legal requirements, so you can focus on the business," https://www.iubenda.com/en/, 2019.

[20] J. Bhatia, T. D. Breaux, and F. Schaub, "Mining privacy goals from privacy policies using hybridized task recomposition," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 25, no. 3, p. 22, 2016.

[21] "Google play," https://play.google.com/store/apps, 2019.

[22] J. Bhatia, T. D. Breaux, J. R. Reidenberg, and T. B. Norton, "A theory of vagueness and privacy risk perception," in *Proc. RE*, 2016, pp. 26–35.

[23] M. C. Evans, J. Bhatia, S. Wadkar, and T. D. Breaux, "An evaluation of constituency-based hyponymy extraction from privacy policies," in *Proc. RE*, 2017, pp. 312–321.

[24] "Cwe-311: Missing encryption of sensitive data," https://cwe.mitre.org/data/definitions/311.html, 2019.

[25] "Vulnerability details : Cve-2002-1697," http://www.cvedetails.com/cve/CVE-2002-1697/, 2002.

[26] "Cwe-916: Use of password hash with insufficient computational effort," https://cwe.mitre.org/data/definitions/916.html, 2019.

[27] "Cwe-329: Not using a random iv with cbc mode," https://cwe.mitre.org/data/definitions/329.html, 2019.

[28] "Msc63-j. ensure that securerandom is properly seeded," https://wiki.sei.cmu.edu/confluence/display/java/MSC63-J.+Ensure+that+SecureRandom+is+properly+seeded, 2019.

[29] F. Jackson and A. Barth, "Protecting high-security web sites from network attacks." Proc. WWW, 2008, pp. 525–534.

[30] Y. Song, C. Yang, and G. Gu, "Who is peeping at your passwords at starbucks? to catch an evil twin access point." Proc. DSN, 2010, pp. 323–332.

[31] "jsoup: Java html parser," https://jsoup.org/, 2019.

[32] "The stanford parser: A statistical parser," https://nlp.stanford.edu/software/lex-parser.shtml, 2019.

[33] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, 2010.

[34] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[35] "Recital 54 gdpr," https://www.convert.com/eu-gdpr/recital-54-gdpr/, 2018.

[36] "National health data dictionary," https://www.aihw.gov.au/reports/health-care-quality-performance/national-health-data-dictionary-version-16-2/contents/table-of-contents, 2019.

[37] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. NIPS*, 2013, pp. 3111–3119.

[38] B. Andow, A. Acharya, D. Li, W. Enck, K. Singh, and T. Xie, "Uiref: analysis of sensitive user inputs in android applications." Proc. WiSec, 2017, pp. 23–34.

[39] "Apktool: A tool for reverse engineering android apk files," https://ibotpeaches.github.io/Apktool/, 2019.

[40] "Ui automator," https://developer.android.com/training/testing/ui-automator, 2019.

[41] 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." Proc. PLDI, 2014, pp. 259–269.

[42] C. Qian, X. Luo, Y. Le, and G. Gu, "Vulhunter: toward discovering vulnerabilities in android applications," *IEEE Micro*, vol. 35, no. 1, pp. 44–53, 2015.

[43] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot-a java bytecode optimization framework." Proc. CASCON, 1999, pp. 214–224.

[44] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, "Iccta: Detecting inter-component privacy leaks in android apps." Proc. ICSE, 2015, pp. 280–291.

[45] Y. Cao, Y. Fratantonio, A. Bianchi, M. Egele, C. Kruegel, G. Vigna, and Y. Chen, "Edgeminer: Automatically detecting implicit control flow transitions through the android framework." Proc. NDSS, 2015.

[46] S. Rahaman, Y. Xiao, S. Afrose, F. Shaon, K. Tian, M. Frantz, M. Kantarcioglu *et al.*, "Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects," *arXiv preprint arXiv:1806.06881*, 2018.

[47] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[48] S. Rasthofer, S. Arzt, and E. Bodden, "A machine-learning approach for classifying and categorizing android sources and sinks." Proc. NDSS, 2014.

[49] J. R. Quinlan, "C4.5: programs for machine learning," *Machine learning*, vol. 16, no. 3, pp. 235–240, 1994.

[50] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers." Proc. UAI, 1995, pp. 338–345.

[51] "Android annotations," https://github.com/androidannotations/androidannotations/wiki, 2016.

[52] L. Xue, Y. Zhou, T. Chen, X. Luo, and G. Gu, "Malton: Towards on-device non-invasive mobile malware analysis for art." Proc. SECURITY, 2017, pp. 289–306.

[53] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors." Proc. NDSS, 2015.

[54] D. Wood, N. Apthorpe, and N. Feamster, "Cleartext data transmissions in consumer iot medical devices." Proc. IoTS&P, 2017, pp. 7–12.

[55] 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 *Proc. USENIX SEC*, 2018, pp. 531–548.

[56] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie, "Policylint: investigating internal privacy policy contradictions on google play," in *Proc. USENIX SEC*, 2019, pp. 585–602.

[57] Ö. Kafali, J. Jones, M. Petruso, L. Williams, and M. P. Singh, "How good is a security policy against real breaches?: a hipaa case study," in *Proc. ICSE*, 2017, pp. 530–540.

[58] X. Wang, X. Qin, M. B. Hosseini, R. Slavin, T. D. Breaux, and J. Niu, "Guileak: tracing privacy policy claims on user input data for android applications," in *Proc. ICSE*, 2018, pp. 37–47.

[59] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction," in *Proc. ICSE*, 2014, pp. 1036–1046.

[60] C. Mulliner, W. Robertson, and E. Kirda, "Hidden gems: automated discovery of access control vulnerabilities in graphical user interfaces," in *Proc. S&P*, 2014, pp. 149–162.

[61] J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, and G. Jiang, "Supor: Precise and scalable sensitive user input detection for android apps." Proc. SECURITY, 2015, pp. 977–992.

[62] Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, and X. Wang, "Uipicker: User-input privacy identification in mobile applications." Proc. SECURITY, 2015, pp. 993–1008.

[63] S. Yang, H. Zhang, H. Wu, Y. Wang, D. Yan, and A. Rountev, "Static window transition graphs for android (t)." Proc. ASE, 2015, pp. 658–668.

[64] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of android apps." Proc. OOPSLA, 2013, pp. 641–660.

[65] D. Torre, G. Soltana, M. Sabetzadeh, L. C. Briand, Y. Auffinger, and P. Goes, "Using models to enable compliance checking against the gdpr: an experience report." Proc. MODELS, 2019, pp. 1–11.

[66] D. Torre, S. Abualhaija, M. Sabetzadeh, L. Briand, K. Baetens, P. Goes, and S. Forastier, "An ai-assisted approach for checking the completeness of privacy policies against gdpr," in *Proc. RE*, 2020.

[67] M. Palmirani and G. Governatori, "Modelling legal knowledge for gdpr compliance checking." in *Proc. JURIX*, 2018, pp. 101–110.

[68] V. Ayala-Rivera and L. Pasquale, "The grace period has ended: An approach to operationalize gdpr requirements." Proc. RE, 2018, pp. 136–146.

[69] J. Tom, E. Sing, and R. Matulevičius, "Conceptual representation of the gdpr: model and application directions." Proc. BIR, 2018, pp. 18–28.