PABAU: Privacy Analysis of Biometric API Usage

Feiyang Tang

Norwegian Computing Center

Oslo, Norway

feiyang@nr.no

Abstract—Biometric data privacy is becoming a major concern for many organizations in the age of big data, particularly in the ICT sector, because it may be easily exploited in apps. Most apps utilize biometrics by accessing common application programming interfaces (APIs); hence, we aim to categorize their usage. The categorization based on behavior may be closely correlated with the sensitive processing of a user's biometric data, hence highlighting crucial biometric data privacy assessment concerns. We propose PABAU, Privacy Analysis of Biometric API Usage. PABAU learns semantic features of methods in biometric APIs and uses them to detect and categorize the usage of biometric API implementation in the software according to their privacy-related behaviors. This technique bridges the communication and background knowledge gap between technical and non-technical individuals in organizations by providing an automated method for both parties to acquire a rapid understanding of the essential behaviors of biometric API in apps, as well as future support to data protection officers (DPO) with legal documentation, such as conducting a Data Protection Impact Assessment (DPIA).

Index Terms—Data protection and privacy, GDPR, biometric privacy, program analysis

I. Introduction

Authentication solutions are shifting from password-based to passwordless solutions due to the documented risks of passwords because of the user-generated credentials, brute-force attacks, recycled passwords, and large-scale breaches. Taking advantage of biometrics, biometric authentication solutions are deployed for both professional usage, for example, company intranet authentication, and personal usage, for example, bank app authentication. The usage diversification of passwordless authentication solutions, based on biometrics, raises privacy concerns since biometric data is acquired and processed within a connected multi-program environment, such as a mobile device or a laptop instead of an offline single-program environment, such as a USB stick [1] that uses biometric as the authentication resource.

With the more common usage of biometric authentication, manipulating biometric data becomes controversial and makes those solutions run the risk of exposing this sensitive data. It is critical to guarantee excellent practice and effective solution implementation throughout the software development process. With more and more privacy-focused legal provisions being implemented, such as the General Data Protection Regulation (GDPR) in Europe and the California Consumer Privacy Act (CCPA), many companies are becoming aware of the importance of privacy in software development. They are beginning

to review their code for potential privacy issues in order to avoid the potential legal risks associated with these newly implemented provisions. GDPR classifies biometric data as "sensitive" [1], and privacy protection on biometric data has become one of the most discussed topics in privacy-related research [2], [3]. Biometric processing is now a significant and vital aspect of privacy protection. Given that more software employs biometric APIs to authenticate users for convenience, it is critical to understand how biometric APIs function and how biometric data is processed. As a result of their nature and sensitive use, biometric APIs must be used with a clear understanding by developers and data protection officers (DPOs) need relevant information from developers. A rigorous privacy assessment of the biometric authenticators is crucial to mitigate privacy risks. However, such an assessment is difficult to perform due to the subtle nature of privacy and the complex structure of programs implementing biometric authenticators.

In this paper, we would like to understand how biometric-relevant APIs are implemented in real-world applications. We propose PABAU, a technique to classify the usage of biometric APIs in the applications. Various usages (those are, methods from the biometric APIs) are classified into different labels according to their behavior, for example, biometric acquisition, user interaction, data transfer, data erasure, etc. We can use the results of the analysis to assist developers in better understanding current solutions that adopt biometric APIs in an action-based analysis and guide them to the section that requires more attention and provide a broad overview of how biometric data is handled and manipulated in the application to either project managers or data protection officers without examining the actual implementation code.

Our contributions are:

- An method for automatically categorizing biometric API methods (Section IV-A and Section V).
- A handcrafted ground-truth dataset built from the popular Android and FIDO2 biometric authentication APIs (Section IV-D).
- An automatic scheme to describe the privacy behavior of API usage in client code, for instance, authentication, cryptography, termination, and permission, that could aid developers and DPOs in analyzing biometric-related privacy compliance (Section IV-C).

We demonstrate the utility of our research by testing our classifier on eight popular Android applications (Section VI).

---

[1] https://fidoalliance.org/specifications/

II. RELATED WORK

Our study focuses on utilizing static analysis to discover software vulnerabilities and supporting non-technical experts with software privacy compliance checks. This section begins with a discussion of traditional software security vulnerability discovery using program analysis, followed by a discussion of software privacy assessment research.

Taint analysis is a practical approach to information flow analysis that is frequently used by researchers to analyze the transmission of private data. It comprises both dynamic and static taint analysis techniques. Using taint analysis to find privacy vulnerabilities in Android applications has been discussed in multiple research.

By studying data relationships between program variables without running the program, static taint analysis may determine if data can propagate from a taint source to a taint aggregation point [4]. Many Java and Android-based research are based on the tool Soot. Soot [5][2] is a Java bytecode analysis tool developed by the Sable research group of McGill University in 1996. It provides a variety of bytecode analysis and transformation functions, through which it can perform intra- and inter-process analysis and optimization, and program flow analysis. FlowDroid [6] mimics a component's lifecycle by building virtual main functions to find sensitive data transfer channels for privacy leaks with an 86% accuracy. MudFlow [7] used the static taint analysis tool FlowDroid to test 2,866 benign software and produced a total of 338,610 taint analysis results. The tool can only give out whether a piece of software as a whole has malicious behavior (and the tool will also have false positives) and cannot tell whether a taint analysis path (source to sink) is the result of direct privacy leakage. Moving from Android to general Java applications, in 2018, Sas et al. proposed SSCM [8] to identify data sources and sinks from arbitrary Java libraries. SSCM uses Soot Framework to extract information from the Java Bytecode and uses its novel rules and WEKA Data Mining Framework [9] to classify different types of security sources and sinks. SWAN [10][3] is similar to SSCM. Both SSCM and SWAN use similar features for the classification model's learning process. SSCM proposed 9 when SWAN contains 25 basic features. These features focus mostly on the name of *Resource Class* and *Resource Method* and *call to the corresponding methods*. Meanwhile, parameters and some specific patterns are also important. Almost all the features used in SSCM have been included in SWAN, and SWAN contains a few more features on return types and some specific patterns in methods. Meanwhile, parameters and some specific patterns are also important.

Privacy-by-design (PbD) has inspired a significant amount of research that provides approaches and models to preserve software privacy before implementation begins, as well as to forecast or manage developer privacy compliance throughout implementation [11], [12]. According to a survey [13] written

by Dias Canedo et al., technical employees frequently lack legal expertise in terms of privacy protection, which encouraged us to design a new easy-to-use technique for developers to check the privacy compliance of their implementation on a regular basis without diving into hundreds of lines of code.

With requirements like the data protection impact assessment (DPIA), researchers have been attempting to make it easier for non-technical professionals, such as attorneys, to confirm GDPR compliance. To aid the examination of privacy consequences in ubiquitous computing systems, Fernández et al. [14] suggested a software-assisted process methodology. Similarly, the BPR4GDPR [15] initiative has the same objective, while Zibuschka [16] analyzed the automation potential of the DPIA process. Using privacy flow-graphs, Tang et al. [17] presented an approach for analyzing privacy in the context of GDPR. This motivates us to bridge the gap between the technical and non-technical experts with an approach that might assist both parties in terms of assessing privacy in biometric API usage.

III. MOTIVATION

Conducting a privacy analysis necessitates a vast number of software information, many of which might be highly specific. Privacy lawyers may request that developers offer precise answers with particular processing in software, which could be challenging when the software is not built and maintained by a single team from the start. Furthermore, manually reviewing hundreds of lines of code demands a significant amount of effort on the part of the development team. It is difficult for developers and lawyers to have a mutual understanding and benefit from each other's expertise and analyzes [11], [18], [19]. Our goal is to see if we can transpose GDPR principles and requirements with actual code locations or patterns in software. In this section, we look at fundamental GDPR obligations from the standpoint of a developer and then consider how we may change our solution to serve both parties in terms of privacy protection in biometric API usage.

A. Legal perspectives

GDPR states several obligations from the data controller which should be monitored by the DPO [20]:

- by default and by design, to have a record of processing activities (Article 30);
- to ensure the security of the processing (Article 32);
- to notify personal data breaches to the supervisory authorities (Article 33);
- to communicate personal breaches to the data subject (Article 34);
- to conduct DPIA (Article 35);
- to conduct prior consultation with supervisory authorities (Article 36).

The DPO's mission is to monitor whether the data controller fulfilled all of his or her commitments, which includes performing a high-quality DPIA when required. This makes the assignment of DPIA creation a duty for both data controllers and DPOs. DPIA is a technique that assists developers and

---

[2]https://github.com/soot-oss/soot

[3]https://github.com/secure-software-engineering/swan/tree/master/swan_core

organizations in systematically analyzing, identifying, and mitigating data protection risks in a project or plan. The deliverable document created by the approach in this study is referred to as a DPIA.

We intend to offer tailored solutions for DPIA under diverse applications, similar to earlier research [21], [22]. The aim of our technique is to provide information that can directly benefit developers and DPOs by providing comprehensive guidance on producing a quality DPIA. Developers and DPOs may identify the components of the software that process biometric data from users within applicable legal viewpoints with a list of biometric API use described under different kinds of behavioral labels. Because various API methods can capture different forms of biometrics processing, we can adopt this into our study.

IV. APPROACH

To tackle the barrier between developers and DPOs in terms of biometric privacy protection compliance checks, we utilize SWAN as a foundation to create a technique PABAU that can automatically categorize methods in biometric APIs based on their actions.

A. General architecture

The architecture of our technique is illustrated in Fig. 1. The compiled class files from Java or Android applications are our source of analysis. We can learn the features of these APIs and use them to categorize their usage in real-world applications by training a model based on the most common biometric API methods, such as the Android official biometric API and FIDO2 implementations. The biometric types included in this study cover the two most common biometrics: face and fingerprint.

PABAU operates on different classification sets, one for each kind of biometric API method. Most action-based sets are not disjoint (for example, `Authenticate()` may be labeled as both **CRYPTO** (cryptography involved) and **AUTHENTI-CATE**), but certain sets like the biometric strength levels (for example, **BSC1** and **BSC2**) are disjoint (that is, each method can only get labeled as one of them). The classification for each label operates independently.

B. Feature types and feature instances

We adopted the same system as SWAN with feature types and corresponding feature instances. The features are extracted from the semantics of methods in the program. Feature types are general features such as `methodNameContains` and feature instances represent concrete instances of feature types such as `methodNameContainsUser`. We considered the following criteria when constructing feature types.

- Names of the methods and the declaring classes:
  - Start with a keyword;
  - Contain a keyword;
  - End with a keyword.
- Return types of the methods:
  - Primitive types (for example, `int`, `string`);

- Other objects (for example, `PromptInfo`).
- Parameters:
  - Number of parameters;
  - Types of parameters: primitive or other object types.
- Invocations: whether the current method calls any other method:
  - The name of callee;
  - The return type of callee;
  - The parameter of the callee.
- Particular pattern derived from above:
  - Parameter flows into the return statement;
  - Parameter flows into a local field;
  - A local field flows into the return statement.

C. Labels

Some particular actions of biometric APIs might lead to potential privacy risks, which motivates us to create labels representing different actions. The following labels in Tab. I are what we used to classify different behaviors in biometric API usage.

TABLE I
LABELS AND THE DESCRIPTION

Label	Description
BSC1	Biometric strength level 'convenience'. This level does not use cryptography nor the `BiometricPrompt` API.
BSC2	Biometric strength level 'weak'. This level uses only the `BiometricPrompt` API but no cryptography.
BSC3	Biometric strength level 'strong'. This level uses both the `BiometricPrompt` API and cryptography.
SOURCE	Where the potentially sensitive data come from.
SINK	The places where the sensitive data might end up in.
CHECKER	Checking occurs, for example, check for hardware prerequisites.
PERMISSION	Getting or verifying the user's permission for the biometric process.
AUTHENTICATE	Authentication-related process.
CRYPTO	Where encryption is involved in the process.
TERMINATION	Where termination of specific service(s) happens.
INTERACTION	The system is making interactions with users, for example, giving users a few options.
TRANSFER	Biometric-related data or decisions derived from it are being transferred.
ACQUISITION	The acquisition of biometric-related data.
DELETION	The deletion of biometric-related data.
STORAGE	Where biometric-related data is being stored and kept.
DATABASE	Where a database is involved.

D. Ground-truth

In order to train PABAU, we need to build a ground-truth dataset specific to biometric authentication and annotate the methods. Because Android apps are the most popular domain for biometric API usage, we chose official Android Biometric APIs as our domain of study. To also cover web applications that use FIDO2, we also find a representative implementation to enrich the ground truth.

Initially, we parse the existing methods from Android biometrics APIs[4] and the FIDO2 server implementation from LINE[5]. To enrich the training set, we also manually annotated over 150 methods from several popular sample implementations of biometric APIs[678]. We manually decompress the

---

[4]https://android.googlesource.com/platform/frameworks

[5]https://github.com/line/line-fido2-server

[6]Soter: https://github.com/Tencent/soter/

[7]Android-Goldfinger : https://github.com/infinum/Android-Goldfinger

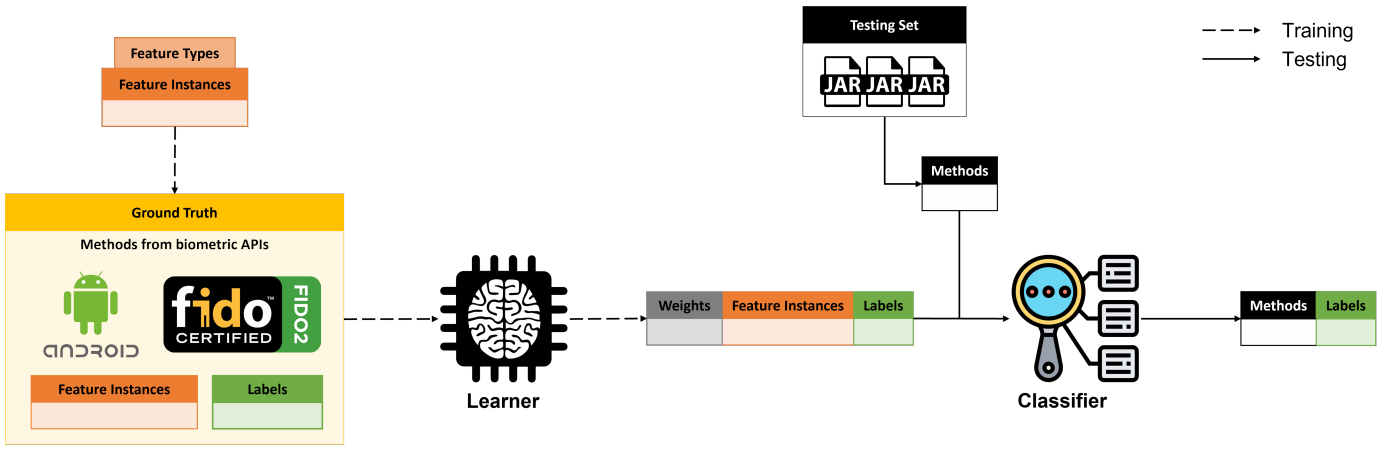[8]https://github.com/sergeykomlach/AdvancedBiometricPromptCompat

Fig. 1. General architecture of PABAU

library files, which were originally in JAR format, and label the methods in the class files in accordance with their corresponding behaviors. A general training set should cover all of the common biometric API methods, which is why we include the native Android biometric API methods and some third-party ones. Each of these methods is annotated with privacy behavior labels. Then, in a structured manner, for each method we collect the following properties:

- **name.** The method's fully qualified name (for example, `package.class.method`). This indicates which class the method belongs to.
- **return.** The method's return type (`void`, primitive types (for example, `int`), or objects of other classes). This helps determine whether there is biometric data flow out from a local method.
- **parametersTypes.** The method's parameters types (empty, primitive types, for example, `int`), or reference types).
- **calleeNames.** The full name of the methods invoked inside that method. This helps determine relationships between methods, to locate data flows of biometric data better.

Each method in this dataset is manually associated with the label(s), the detailed explanation of labels is discussed in Section IV-C. The example of how the training process is illustrated in Fig. 2 along with an annotation example in Fig. 3. Details on which type of feature and feature instance are summarized in Section IV-B.

V. TRAINING AND CLASSIFICATION

Similarly to SWAN [10]'s technique, we develop a collection of binary features that assess certain qualities of the methods to assist the machine learning algorithm in labeling the methods based on their actions.

We evaluated six popular classifiers from WEKA [9], a commonly used machine learning API written in Java: Bayes Net, Naive Bayes, Logistic Regression, C4.5, and Decision Stump and SVM. For each classifier in the training set, ten 10-fold cross-validations were conducted to find the best classifier

Fig. 2. Example of a method from the Android Biometric API: `authenticate` is a method from the class `android.hardware.biometrics.BiometricPrompt` and the corresponding annotated method (in the green text) data point. The numbered circles (from lines 23 to 28) correspond to the labels (Section IV-C). They were assigned to this data point since `authenticate` has the properties in lines 1, 4, 7, 10, 16-20 in the text.

for PABAU. The median precision and recall statistics for each classifier are presented in Table. II. In terms of average accuracy and recall, we can observe that SVM performed the best with an average precision of 0.9725 and recall of 0.98. Except for Stump, which performed far less well than the other five, the majority of classifiers functioned well.

For example, the feature instance `hasCalleeNameStartsCheck` is likely to indicate that the current method calls a checker function to see
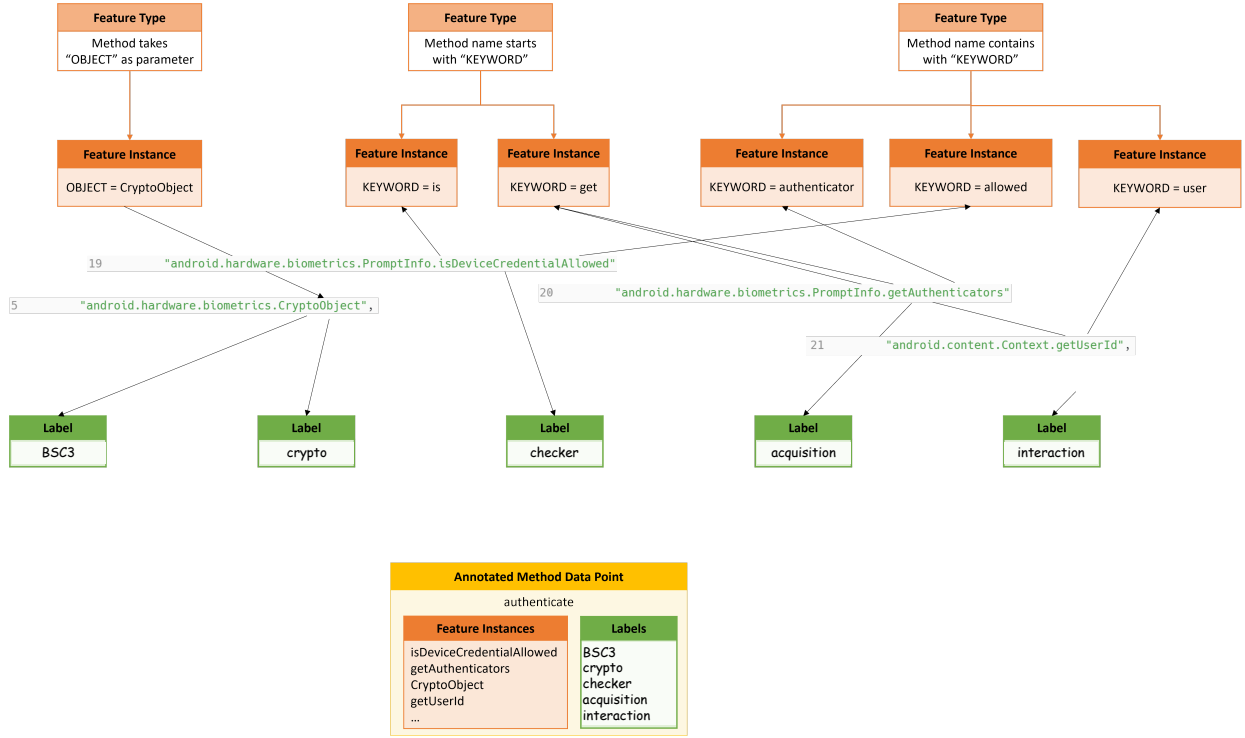
Fig. 3. Annotation diagram of the sample method

whether specific conditions have been satisfied. When the learning process starts, PABAU computes a yes or no response for each feature instance for each method in the training set. Based on the feedback, we are able to learn which combination of features is the most associated with the labels. These combinations will be then used to label methods in testing sets.

VI. EXPERIMENT

The main research question here is how our technique performs in terms of assigning biometric API methods with the corresponding behavioral labels. We aim for high precision because calculating recall is unrealistically tricky. All our experiments were performed on a Windows machine with an Intel i7 1.90GHz CPU and 16 GB memory.

A. Description of the datasets

Besides the ground-truth dataset we built using Android biometric APIs and FIDO2 server implementation, we also manually annotated biometric API usages from real-world Android mobile applications. Given that biometric authentication is only used in a subset of Android applications, we chose eight popular apps from the Google Play Store[9] in four categories: banking & finance (Klarna, Sparebank1, Revolut, Paypal), private storage (Private Photo Vault, NordLocker), privacy notebook (Notability), and password management (1Password).

[9]https://play.google.com/store/apps/

The description of the main functionality and biometric usage of the apps are listed in Table III.

We split the entire handcrafted dataset into two parts, 70% of the data points were used to train the classifier and the rest 30% were used for testing.

B. Example

Fig. 4 gives an example of what kind of result PABAU generates by analyzing the JAR files. The method `onCreateView` was extracted from the class `androidx.biometrics.BiometricFragment`.

We could see that the method itself gets classified by multiple labels, for example, it gets classified as **BSC3** and **Crypto** from the invocation to the data type `CryptoObject`. The app's implementation has a fairly small number of biometric API methods, however, this does not diminish their importance. Actually, their significance might be underestimated, and the underlying correlations between one method and other subtle behaviors reveal a sensitivity to privacy. This is also why we need a technique such as PABAU to classify each method rigorously with as many privacy behavior labels as feasible.

C. Runtime and memory performance

Table IV demonstrates the stability of PABAU, and it does not require huge computational costs to analyze common mobile apps. The cost of time and memory increases as the size of a program grows, which also reflects a greater number of methods inside the application. We also observed that the quantity of biometric-related approaches influences the time and memory requirements. Even though Notability is almost

	Source		Sink		Auth		Crypto		Average	
	P	R	P	R	P	R	P	R	P	R
BayesNet	0.92	0.97	0.95	0.95	1.00	1.00	1.00	1.00	0.9675	0.9800
NaiveBayes	0.93	0.95	0.94	0.96	0.88	1.00	1.00	1.00	0.9375	0.9775
Logistic Reg	0.94	0.95	0.95	0.92	0.88	1.00	1.00	1.00	0.9425	0.9675
C4.5	0.93	0.96	0.93	0.95	0.88	1.00	1.00	0.75	0.9350	0.9775
Stump	0.79	0.84	0.88	0.92	0.75	1.00	1.00	0.5	0.8550	0.8150
SVM	0.94	0.96	0.95	0.96	1.00	1.00	1.00	1.00	**0.9725**	**0.9800**

```
1  public View onCreateView(@NonNull LayoutInflater var1, @Nullable ViewGroup var2, @Nullable Bundle
   var3) {                        Interaction
2      if (!this.mShowing) {
3          Bundle var5 = this.mBundle;
4          if (var5 != null) {          BSC2 or BSC3
5              this.mBiometricPrompt = var9.build();          Termination
6              CancellationSignal var10 = new CancellationSignal();
7              this.mCancellationSignal = var10;
8              CryptoObject var15 = this.mCryptoObject;
   Crypto & BSC3
9              if (var15 == null) {          Authenticate
10                 this.mBiometricPrompt.authenticate(var10, this.mExecutor,
   this.mAuthenticationCallback);
11             } else {
12                 this.mBiometricPrompt.authenticate(wrapCryptoObject(var15), this.mCancellationSignal,
   this.mExecutor, this.mAuthenticationCallback);
13             }          Interaction
14         }
15     }
16     this.mShowing = true;
17     return super.onCreateView(var1, var2, var3);          Flow to return
18 }
```

Fig. 4. Sample classification result snippet from Sparebank1

TABLE III
DESCRIPTION OF THE DATASETS

App name	Description
Klarna	A financial app that provides online payment and direct payments along with post-purchase payments. Users can use biometrics to unlock the app and authorise certain payments.
Sparebank1	A online banking app for Sparebank1. Users can use biometrics to unlock the app, authorize certain payments and approve e-invoice.
Revolut	An online banking app for Revolut Bank. Users can use biometrics to unlock the app, add a new card, authorize payments, and update profiles.
Paypal	A financial app that provides the online payment between users or merchants. Users can use biometrics to unlock the app, authorize payments, and update card/account details.
Private Photo Vault	A photo storage app that provides access control. Users can use biometrics to unlock the app, add photos to the vault and add extra control to certain folders.
NordLocker	A storage app that provides access control. Users can use biometrics to unlock the app, add files into the locker and apply extra encryption to certain files.
Notability	An online notebook. Users can use biometrics to lock and unlock certain notes.
1Password	A password manager. Users can use biometrics to unlock the app, add/update credentials, and sync information with the cloud.

three times the size of Sparebank1, it does not cost three times as much in terms of time and memory since Notability uses

biometric APIs for fewer features than Sparebank1.

TABLE IV
AVERAGE RUNTIME AND MEMORY USAGE FOR CLASSIFYING DIFFERENT APPS

	#Total methods	Average cost	
		Time (s)	Memory (MB)
Klarna	105,185	38.51 ± 1.68	641.48 ± 2.15
Sparebank1	89,142	32.79 ± 0.72	584.37 ± 1.93
Revolut	86,457	30.25 ± 0.78	518.56 ± 1.84
Paypal	94,618	35.96 ± 1.23	604.62 ± 1.89
Private Photo Vault	175,841	43.19 ± 1.48	659.07 ± 2.24
NordLocker	115,294	30.57 ± 1.55	556.29 ± 2.05
Notability	284,585	65.48 ± 2.14	844.73 ± 3.21
1Password	134,806	31.33 ± 1.67	581.46 ± 2.23

D. Evaluating the precision

To examine the classification precision, we manually go through the methods that got classified by each behavioral label. The methods were extracted from the decompiled `apk` files and read by our technique. Table V displays the total number of methods analyzed in each application as well as the classification results for various behaviors (categories). It is worth noting that all finance apps use the cryptography-related API, and none of the apps have access to actual

biometric data storage, database, or acquisition (the only acquisition that happens is the cryptographic keys involved in the authentication process, no biometric data involved). This is aligned with GDPR requirements since transporting and storing biometric data is very sensitive and requires further consent from data owners.

According to Table VI, PABAU has an average precision of 0.84 across all the labels. It is more precise for detecting **Source** and **Sink** (0.96), as well as some categories such as **BSC1/2/3** (0.97), than for other categories such as **Termination** (0.87). Some of the behavior labels appear infrequently in the dataset as well.

PABAU can be improved by using a larger and more diverse training set that includes real-world API usage annotations and domain-specific information.

TABLE V
TOTAL NUMBER OF METHODS ANALYZED (#M) AND NUMBERS OF METHODS DETECTED BY PABAU PER BEHAVIORAL LABEL.

	Klarna	Sparebank1	Revolut	Paypal	PPV	Nordlocker	Notability	1Password
#M	105,185	89,142	86,457	94,618	175,841	115,294	284,585	134,806
Source	127	94	174	140	65	98	39	55
Sink	96	70	143	109	41	64	17	32
BSC1	15	12	21	16	15	20	11	13
BSC2	17	24	29	14	7	9	0	5
BSC3	2	2	3	2	0	0	0	0
Checker	9	7	13	8	3	4	1	3
Permission	7	5	9	8	1	1	1	3
Auth	8	6	9	11	2	3	2	4
Crypto	5	5	6	9	0	0	0	0
Termin	9	7	10	9	3	1	1	5
Interact	4	2	5	7	1	2	1	4
Transfer	0	0	0	0	0	0	0	0
Acquist	1	1	1	1	0	0	0	0
Delete	2	1	2	3	0	0	0	0
Storage	0	0	0	0	0	0	0	0

TABLE VI
NUMBER OF BIOMETRIC-RELATED METHODS DETECTED (#BM), AND PRECISION OF PABAU FOR EACH LABEL ON EIGHT ANDROID APPLICATIONS. '/' MARKS LABELS FOR WHICH PABAU DETECTED NO METHODS.

	Klarna	Sparebank1	Revolut	Paypal	PPV	Nordlocker	Notability	1Password
#BM	326	201	427	372	170	233	89	143
Source	0.95	0.94	0.92	0.9	0.99	0.99	0.97	0.99
Sink	0.97	0.95	0.97	0.94	0.98	0.99	0.99	0.96
BSC1	0.93	1	1	0.89	1	0.95	1	0.92
BSC2	1	0.96	0.93	1	1	0.89	/	1
BSC3	1	1	1	1	/	/	/	/
Checker	0.67	0.57	0.77	0.5	0.33	0.5	1	0.67
Permission	1	0.8	0.67	0.89	1	1	1	1
Auth	1	1	1	1	1	1	1	1
Crypto	1	1	1	1	/	/	/	/
Termin	0.56	0.86	0.8	1	1	1	1	0.8
Interact	1	1	1	0.86	1	1	1	1
Transfer	/	/	/	/	/	/	/	/
Acquist	1	1	1	0	/	/	/	/
Delete	0.5	1	1	0.3	/	/	/	/
Storage	/	/	/	/	/	/	/	/

VII. THREATS TO VALIDITY

To check for GDPR compliance in the biometric authentication system and find relevant behavior, here we picked one of the most often used sample templates for generating a DPIA from the Commission nationale de l'informatique et des libertés (CNIL) [23].

Under "*Section 2 Assessment of controls protecting data subjects' rights*", there are key questions that can be answered by the classification results from PABAU. We pick the following examples which can be answered by our classification result:

- *Determination and description of the controls for obtaining consent.*
  Typically, the operating system, such as Apple or Google, obtains biometric permission first from users when they enroll their biometrics into the system. Permission is required the first time an application asks for access to the biometric API. This refers to the methods we classified with the **Permission** label, these methods indicate the code location and description of biometric permissions sought by the application. From the preceding experiment result, it is evident that all apps that implemented biometric authentication requested permission. By analyzing the discovered methods, further information can be uncovered. For instance, the use of **Permission** methods may be used to examine privacy policies to determine whether there are locations that utilize biometric data without user consent.
- *Information on the secure data storage method, particularly in the event of sourcing.*
  Various biometric security class levels signify varying degrees of security. DPOs need to understand how user data is being securely stored. Labels **Crypto** and **BS3** offer the greatest degree of data protection and inform DPOs that the biometric authentication method utilized sophisticated encryption.
- *Detailed presentation of the data processing purposes (specified objectives, data matching where applicable, etc.).*
  Biometric APIs are utilized for more than just app unlocking. The label **Iteract** classifies the interaction between the user and the application in terms of biometric API usage, as user interaction must be precisely identified. By analyzing these approaches, DPOs can determine which activity motivates the use of biometric APIs. Similarly, labels such as **Auth** explain the authentication procedure in a clear manner by pinpointing the locations of authentication processes and associating the context in the code.
- *Possibility of retrieving, in an easily reusable format, personal data provided by the user, so as to transfer them to another service*
  Transferring biometric data or the authentication session's decisions is a highly sensitive process. In spite of the fact that our experiment did not reveal any methods with the **Transfer** label, it is still worthwhile to discover them in the implementations. Considering the Android biometric data is stored in the Trusty TEE (Trusted Execution Environment) [24], which is isolated from the rest of the system by both hardware and software. By analyzing the Android implementation, we should not find any data transportation process since Trusty and Android run parallel to each other.
- *Indication of the personal data that will nevertheless be stored (technical requirements, legal obligations, etc.).*
  The **Storage** and **Delete** labels enable DPOs to identify and comprehend methods that may breach GDPR data retention requirements.

VIII. Conclusion and future work

It has always been a barrier between technical and non-technical people in terms of privacy protection for biometric data. DPOs rely on technical specifications from developers to make legal decisions in order to continue monitoring privacy protection compliance in software implementations.

In this paper, we propose a technique for labeling biometric API usage with behavioral labels. This approach overcomes this barrier by providing an automated method for both parties to gain a quick overview of the important fundamental behaviors of biometric API in applications, as well as future assistance to DPOs with legal paperwork, such as performing a DPIA.

Our work provides an early promising result in categorizing the behaviors of biometric API methods in eight popular apps. However, we did not have a large number of samples for the training set, the diversity can also be improved. Meanwhile, experienced developers' involvement might benefit the learning process by providing valuable relevant elements for training, such as varied weights for features. We hope that this method will help both technical and non-technical workers gain a better grasp of privacy protection scenarios during the software development process.

Acknowledgment

References

[1] Council of European Union, "General data protection regulation (GDPR) Art. 9: processing of special categories of personal data," 2018, https://gdpr-info.eu/art-9-gdpr/.

[2] N. Gruschka, V. Mavroeidis, K. Vishi, and M. Jensen, "Privacy issues and data protection in big data: a case study analysis under GDPR," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 5027–5033.

[3] C. Jasserand, "Legal nature of biometric data: From generic personal data to sensitive data," *Eur. Data Prot. L. Rev.*, vol. 2, p. 297, 2016.

[4] L. Wang, F. LI, L. LI *et al.*, "Principle and practice of taint analysis," *Journal of Software*, vol. 28, no. 4, pp. 860–882, 2017.

[5] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot: A Java bytecode optimization framework," in *CASCON First Decade High Impact Papers*, 2010, pp. 214–224.

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

[7] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining apps for abnormal usage of sensitive data," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 426–436.

[8] D. Sas, M. Bessi, and F. A. Fontana, "Automatic detection of sources and sinks in arbitrary java libraries," in *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2018, pp. 103–112.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[10] G. Piskachev, L. N. Q. Do, and E. Bodden, "Codebase-adaptive detection of security-relevant methods," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2019. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3293882.3330556

[11] I. Hadar, T. Hasson, O. Ayalon, E. Toch, M. Birnhack, S. Sherman, and A. Balissa, "Privacy by designers: software developers' privacy mindset," *Empirical Software Engineering*, vol. 23, no. 1, pp. 259–289, 2018.

[12] J.-H. Hoepman, "Privacy design strategies," in *ICT Systems Security and Privacy Protection*, N. Cuppens-Boulahia, F. Cuppens, S. Jajodia, A. Abou El Kalam, and T. Sans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 446–459.

[13] E. Dias Canedo, A. Toffano Seidel Calazans, E. Toffano Seidel Masson, P. H. Teixeira Costa, and F. Lima, "Perceptions of ICT practitioners regarding software privacy," *Entropy*, vol. 22, no. 4, p. 429, 2020.

[14] A. P. Fernández and G. Sindre, "Software Assisted Privacy Impact Assessment in interactive ubiquitous computing systems," in *Conference on e-Business, e-Services and e-Society*. Springer, 2019, pp. 60–71.

[15] G. V. Lioudakis, M. N. Koukovini, E. I. Papagiannakopoulou, N. Dellas, K. Kalaboukas, R. M. d. Carvalho, M. Hassani, L. Bracciale, G. Bianchi, A. Juan-Verdejo *et al.*, "Facilitating GDPR compliance: the H2020 BPR4GDPR approach," in *Conference on e-Business, e-Services and e-Society*. Springer, 2019, pp. 72–78.

[16] J. Zibuschka, "Analysis of automation potentials in privacy impact assessment processes," in *Computer Security*. Springer, 2019, pp. 279–286.

[17] F. Tang and B. M. Østvold, "Assessing software privacy using the privacy flow-graph," in *Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security*, ser. MSR4P&S'22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3549035.3561185

[18] M. Hansen, "Top 10 mistakes in system design from a privacy perspective and privacy protection goals," in *IFIP primelife international summer school on privacy and identity management for life*. Springer, 2011, pp. 14–31.

[19] C. Bettini and D. Riboni, "Privacy protection in pervasive systems: State of the art and technical challenges," *Pervasive and Mobile Computing*, vol. 17, pp. 159–174, 2015.

[20] Council of European Union, "General data protection regulation (GDPR)," 2018, https://gdpr-info.eu/.

[21] M. Horák, V. Stupka, and M. Husák, "Gdpr compliance in cybersecurity software: A case study of dpia in information sharing platform," in *Proceedings of the 14th international conference on availability, reliability and security*, 2019, pp. 1–8.

[22] J. Henriksen-Bulmer, S. Faily, and S. Jeary, "Dpia in context: Applying DPIA to assess privacy risks of cyber physical systems," *Future Internet*, vol. 12, no. 5, p. 93, 2020.

[23] "Privacy Impact Assessment (PIA) by CNIL," https://www.cnil.fr/en/privacy-impact-assessment-pia, February 2018, (Accessed on 08/19/2022).

[24] Wikipedia contributors, "Trusted execution environment — Wikipedia, the free encyclopedia," 2022, [Online; accessed 15-October-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Trusted_execution_environment&oldid=1105563332