Sensitive and Personal Data: What Exactly Are You Talking About?

Maria Kober*, Jordan Samhi†, Steven Arzt‡, Tegawendé F. Bissyandé† and Jacques Klein†
* Email: mariakober.research@gmx.com
†SnT, University of Luxembourg, Luxembourg - Email: firstname.lastname@uni.lu
‡Fraunhofer Insititute for Secure Information Technology, Darmstadt, Germany - Email: steven.arzt@sit.fraunhofer.de

***Abstract*—Mobile devices are pervasively used for a variety of tasks, including the processing of sensitive data in mobile apps. While in most cases access to this data is legitimate, malware often targets sensitive data and even benign apps collect more data than necessary for their task. Therefore, researchers have proposed several frameworks to detect and track the use of sensitive data in apps, so as to disclose and prevent unauthorized access and data leakage. Unfortunately, a review of the literature reveals a lack of consensus on what sensitive data is in the context of technical frameworks like Android. Authors either provide an intuitive definition or an ad-hoc definition, derive their definition from the Android permission model, or rely on previous research papers which do or do not give a definition of sensitive data. In this paper, we provide an overview of existing definitions of sensitive data in literature and legal frameworks. We further provide a sound definition of sensitive data derived from the definition of personal data of several legal frameworks. To help the scientific community further advance in this field, we publicly provide a list of sensitive sources from the Android framework, thus starting a community project leading to a complete list of sensitive API methods across different frameworks and programming languages.**

I. Introduction

The Android mobile operating system dominates the current mobile market [1]. Millions of Android applications (apps) are available in both official markets like Google Play and alternative markets like AppChina. These apps can gain access to various types of data such as user personal data (name, phone number, address, email address, etc.), sensor readings (accelerometer, gyroscope, light, gravity, temperature, etc.), connection and device settings (cellular network, Wi-Fi, serial number, NFC data, etc.), and many more.

Parts of the data accessible on a phone, e.g., location information or the list of phone calls, are highly *sensitive* for the security and privacy of the users. The Android permission system [2] is not a full remedy for unauthorized data access as many apps ask for more permissions than necessary for their use case [3]. Users often fail to understand the permission requests of apps and grant permissions overly broadly [4]. Furthermore, malware apps try to gain access to sensitive data, e.g., from eavesdropping on private conversations [5].

Hence, techniques and frameworks have been proposed that try to detect and sometimes vet Android apps for leaks and misuse of sensitive data [6]–[11]. Regardless of the concrete approach, all of these works require a definition of which data needs protection and which properties should be checked.

Most current works base their definition of sensitive data on either ① implicit intuition, ② a custom and ad-hoc definition, ③ a predefined list of methods that are considered to return sensitive data, or ④ the Android permission system.

The first two approaches, intuition and ad-hoc definitions, render it impossible to compare results between papers. Further, such definitions are imprecise, leaving different researchers with different judgments on which data to include as sensitive. Additionally, example-guided approaches provide no evidence, let alone proof of completeness. For the third approach, several studies [12]–[14] have shown that these predefined lists are both incomplete and over-approximated.

Finally, relying on the Android permission model is not an alternative. Not all API methods protected by a permission return sensitive data. For example, the `setNetworkSelectionModeManual` method of the `TelephonyManager` class requires the `MODIFY_PHONE_STATE` permission but does not return sensitive data. Instead, it returns a Boolean value that specifies whether the requested change of settings was successful or not. Conversely, some API methods not protected by permissions can be used to retrieve sensitive data. The `getText` method of the `EditText` class, for example, can be used to retrieve data from a text input field in the user interface. Depending on the context, such data might be sensitive, e.g., a password, a social security number, or banking information. Such accesses, since they happen inside one app, are not protected by a permission. Still, a tool that detects leaks of sensitive data should track the data read from these fields. Further, the Android permission model is platform-specific. While iOS offers similar sensitive data, it relies on entirely different protection mechanisms. Defining sensitive data using references to platform-specific concepts may lead to inconsistencies, e.g., when comparing how privacy-friendly apps are for different platforms.

In total, whether intuition-based, ad-hoc, based on predefined lists, or derived from platform-specific permissions, existing approaches do not provide a clear, concise and thorough definition on which data shall be considered sensitive. Consequently, they cannot offer code-level definitions of which data to track or check against privacy rule sets.

This shortcoming is especially severe, because preventing privacy leaks is a key legal requirement for software in many jurisdictions. Legal frameworks such as the European GDPR [15] threaten app developers and cloud operators with

significant fines in case of data loss due to neglect. At the same time, high-level legal frameworks do not easily map to code-level elements such as API calls that are used for retrieving data, making it hard for developers and security analysts to ensure compliance.

We identify the need for a clear, well-accepted, and universal definition of what *sensitive data* is in the context of technical frameworks like the Android operating system. With such a definition, researchers can compare their approaches centered around sensitive data. The definition can also help penetration testers, software developers, and developers of code scanners to better identify improper handling of sensitive data and, thus, to comply with privacy regulations.

While we focus on privacy and personal data, we acknowledge that other data may also be sensitive. For example, companies have commercial secrets such as the blueprints for their products or their financial details. However, such non-privacy-related data is highly domain-specific and usually not accessible via specific, purpose-built APIs provided by platforms such as Android. Further, it is usually not regulated using generic legal frameworks such as the GDPR. We therefore exclude such data and focus on privacy-sensitive data.

This paper makes the following original contributions:

- We provide a multi-layered definition of *sensitive data*.
- We introduce a definition language to tag methods of programming APIs, e.g., Java methods, as sensitive data.
- We provide a list of manually assembled Android API methods that return sensitive data based on our definition. This list is meant to evolve. We see this work as start of a community project aiming to build a public and complete list of sensitive APIs. The list is available at:

  https://github.com/JordanSamhi/SensitiveData

This paper is structured as follows. Section II introduces existing legal and technical definitions of personal and sensitive data. Section III proposes a sound, multi-layered definition of sensitive data. Section IV provides a definition language for tagging API methods according to our definition. In section V, we apply both our definition and definition language to methods of the Android API. In section VI, we introduce the community project. Section VII concludes this paper.

II. Existing Definitions and Problems

Android apps can access and manipulate several types of data from many different sources, given that the required permissions have been granted. *Personal data* can be used to identify a person or disclose information about a person. Another set of data can be considered as *sensitive*; in the following, sensitive data is always related to a person (cf. Section II-A). In this section, we show that: ① existing definitions in legal framework cannot be used as is to define what sensitive data is from a technical point of view; and ② existing works do not agree on how to define sensitive data.

A. Legal Frameworks

In this section, we describe legal frameworks that define what *sensitive* and *personal data* is. These legal definitions shall serve as the ground truth for lists of sensitive APIs on a technical level.

The General Data Protection Regulation (GDPR) [15] is a European Union legal framework on data and privacy protection that became enforceable in 2018. Article 4 of the GDPR gives the following definition of personal data:

> *'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person*

Sensitive data is seen as "special categor[y] of personal data" [15, p. 2] that is subject to additional limitations and that requires additional precautions. The European Commission provides a condensed overview of which data can be considered as sensitive in the framework of the GDPR[1]:

> *personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs; trade-union membership; genetic data, biometric data processed solely to identify a human being; health-related data; data concerning a person's sex life or sexual orientation.*

Other juridical areas provide similar legal frameworks for restricting the access to and use of personal data. Their definitions of sensitive and personal data are largely similar to the GDPR, e.g., in the *Colorado Privacy Act* (CPA)[2] and the *Virginia Consumer Data Protection Act* (CDPA)[3].

Some regulations provide additional requirements beyond the provisions of the GDPR. The South African *Protection of Personal Information Act* (POPI Act)[4] explicitly includes "views or opinions of another individual about the person" as personal data. The *California Consumer Privacy Act* (CCPA)[5] explicitly protects information that can identify a household or a device, even if such data is not immediately associated with a single person. The *Brazilian General Data Protection Law* (LGPD)[6] explicitly considers behavioral profiling information as personal, if it is connected to an identified person. The *Japanese Act on the Protection of Personal Information* (APPI)[7] considers information as personal when it can be collated with other information to identify a person.

Data Protection in Turkey (KVKK)[8] relies on an implicit definition of personal data. The legal text in itself does not provide guiding examples of the data to which it refers.

In total, we find that many provisions and definitions are similar. Yet, there are also key differences as to which data is personal or sensitive and which is not. When translating

[1] https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/legal-grounds-processing-data/sensitive-data/what-personal-data-considered-sensitive_en

[2] https://leg.colorado.gov/bills/sb21-190

[3] https://lis.virginia.gov/cgi-bin/legp604.exe?211+sum+SB1392

[4] https://popia.co.za/

[5] https://oag.ca.gov/privacy/ccpa

[6] https://iapp.org/media/pdf/resource_center/Brazilian_General_Data_Protection_Law.pdf

[7] https://www.ppc.go.jp/files/pdf/APPI_english.pdf

[8] https://www.kvkk.gov.tr/Icerik/5389/Data-Protection-in-Turkey

legal definitions to the code level, i.e., concrete APIs, analysts must be able to distinguish them based on the target audience of the application. Lists of sensitive methods must therefore be tagged with the legal framework under which the data returned by the respective API is considered sensitive. Additionally, when data is sensitive or personal and when it can be considered sufficiently anonymized is still actively being debated [16]. Therefore, lists of sensitive API methods need to be kept up-to-date, which is possible, for example, in an continuous community effort.

Further, we stress that the semantic gap between legal frameworks and technical frameworks on API level (e.g., the Android ecosystem) has not been solved yet. While legal definitions provide guidelines for human data protection officers, they cannot directly be mapped to API calls and data fields without ambiguity. While some pieces of data can directly be used to identify a person alone (i.e., name), other data ① needs to be combined to identify someone (e.g., postal code) and, thus, is not sensitive alone; ② can be used to identify a person but only in a particular context (e.g., a picture can be sensitive or not, depending on the context); ③ depends on concrete values (e.g., clipboard data); or ④ gives access to other sensitive information (e.g., passwords as part of authentication).

B. Existing Definitions

In Section I, we introduced four basic approaches: intuition, custom definition, permission-based, and pre-defined lists. In the following, we explain these approaches in more detail.

a) Permission-based: Several approaches consider data that is returned by a method to be sensitive if the method is protected by an Android permission [3], [11], [17]–[20]. In most cases, they assume that all data obtained from such methods is sensitive, and that all sensitive data can only be obtained with a permission. In section I, we have shown that this is not the case in practice.

b) Pre-defined list of methods: Another category of approaches re-uses lists of methods that return sensitive data from earlier works [21]–[25]. Some of these lists are permission-based [3], [26], [27]. Others, such as SuSi [28], were generated using Machine Learning. Other approaches rely on DidFail's [29] list which is example-based and lacks a proper definition of sensitive data.

Such lists are mostly a byproduct of other works (e.g., a data flow tracker). Therefore, they can neither be expected to be complete, nor kept up-to-date. Further, not all methods in these lists necessarily return sensitive data. The original (implicit) definition of a sensitive method may not align with the use case for which the list is later applied, e.g., when new legal frameworks apply. Further, these lists contain false positives [12]–[14], [30].

c) Intuition: Some approaches rely on intuition and examples for identifying sensitive data [31], [32]. Other works do not define what sensitive data is [33], [34]. These approaches implicitly accept an incomplete list. UiRef [35], e.g., provides the examples passwords, credit card numbers, social security numbers, passport numbers, and healthcare information. Besides being incomplete on a conceptual level, the mapping between a concept like "healthcare information" and concrete APIs is also ad-hoc, incomplete, and easily outdated.

d) Ad-hoc definitions: Another category of approaches uses ad-hoc definitions of what sensitive data is. Examples of these approaches are: ① Difuzer [36], a hybrid approach for detecting suspicious hidden sensitive operations in Android apps. For their data flow analysis, the authors systematically collected a list of source methods. However, they restricted themselves to system inputs and environment variables in the Android framework that they consider to return sensitive data; ② TaintDroid [9] is a dynamic analysis approach to track information flow in Android apps to reveal potential sensitive data leaks. The authors define sensitive data as a piece of data returned by a taint source. Thereafter, they define a taint source as a privacy-sensitive source. The authors give the list of sources they considered in their study: LocationManager data, SensorManager data, data buffers and files to track microphone and camera information, database files, the phone number, ICC-ID, IMEI number, and the native socket library.

In total, we conclude that several approaches do not provide a definition, rely on a (present or not present) definition in previous work, or provide a definition that is not comparable with other approaches in literature (e.g., based on intuition or an ad-hoc definition), thus being without guarantees on the underlying list for either completeness or comparability.

III. A Definition of Sensitive Data

The definition of *personal data* in legal frameworks is usually quite broad; *sensitive data* can oftentimes be summarized as personal data requiring special protection. For any definition targeting technical frameworks, this means that almost every data obtained through an API call can be personal data and possibly sensitive data – an image, a file, a phone number, a location. Yet we note that not all data (e.g., images) are always sensitive or personal data, e.g., the sensitiveness depends on the context of the data. Therefore, we propose a multi-layered definition of *sensitive data*, derived from the definitions of *personal data*:

(a) **Always-sensitive data:** Data that can categorically identify a person without any other information and/or is always directly related to a person. For example, an email address, username, name, or unique device identifier.

(c) **Combination-sensitive data:** Data that is only sensitive when combined with other data. For example, a location is neither sensitive nor personal data, but becomes sensitive once combined with a persons' name and the date the person visited the location.

(t) **Context-sensitive data:** Sensitive depending on the content and context. A photo, for example, is likely sensitive in a healthcare app (photo of, e.g., some wounded skin), but not in a review app for hotels, where the photo is intended for publication in any case.

(g) **General-purpose data:** Sensitive depending on the concrete value, e.g., clipboard data. Regardless of the cur-

rent app, the system-wide shared clipboard may contain arbitrary data, including a password or phone number.

(p) **Access data:** Data non-sensitive by itself, but – possibly in combination with other data – gives access to sensitive data. An example is a password, which is usually combined with a username or email address.

In general, we assume a device to be linked with a person. This is in accordance with, e.g., the GDPR which states that "[n]atural persons may be associated with online identifiers provided by their devices, applications, [...] or other identifiers [...]" [15, p. 6]. We also assume that users use an app as intended. It is not the developers' responsibility to protect the contents of, e.g., a chat app the same way as medical records, although users may abuse the app to send files to doctors.

IV. Definition Language

In this section, we propose a formal definition language for tagging sensitive API methods, i.e., methods that may provide access to sensitive data, and corresponding parameters. This language is based on our definition of sensitive data in section III. The language itself is tool- and platform-agnostic. Concrete method lists, however, are specific to a particular framework, such as Android or iOS.

A sensitive method is a tuple $\Sigma := \langle \sigma, \kappa, \zeta, \rho, \tau \rangle$, where σ is the signature of the method in Soot's notation [37], which is based on the Jimple [38] method signature. While originally designed for Java, this notation – consisting of class name, method name, and parameter list – can also be applied to other object-oriented languages such as ObjectiveC for iOS.

Sensitive data can be obtained in two ways. Firstly, the developer can call a getter-style API method, e.g., `getLastKnownLocation` for obtaining the last GPS position. Secondly, many platforms provide callbacks that notify apps when data becomes available, e.g. `onLocationChanged` for when the GPS position changes. In our model, κ is the parameter index in a callback, or -1 to denote the return value of a getter-style call.

$\zeta \in a, c, t, g, p$ specifies whether the data provided by a method is always sensitive (a), combination-sensitive (c), context-sensitive (t), general-purpose (g), or access (p) data.

$\rho \in \mathcal{P}(\mathbb{R})$ denotes the set of regulations to which this method applies, e.g., "GDPR-sensitive" for Art. 9 data from the European GDPR. This annotation is necessary because the definition from section III captures a superset over different legal frameworks. Still, a given developer may only be interested in API methods relevant for a particular legislation based on their respective jurisdiction.

$\tau \in \mathcal{P}(\mathbb{T})$ denotes a set of tags for categories (e.g., "photo", "email"), indicating which type of sensitive data the method provides. Analysts and developers can then choose whether, in their context, photos are considered sensitive. We envision that data protection officers may provide a list of such relevant categories. This would allow the data protection officer to reason about the app on a domain-specific, asset-driven level, while still allowing for tool support based on our method list. We consider the universe of tags $\mathcal{P}(\mathbb{T})$ as semi-extensible, i.e., subject to a curated process to ensure a consistent taxonomy.

V. API Examples

As an example, we discuss the `getImsi()` method of the `TelephonyManager` class. The method returns data that is always sensitive, because the IMSI can uniquely identify the user. In this case σ = android.telephony.TelephonyManager: java.lang.String getImsi() which is the Jimple [38] method signature; $\kappa = -1$ since the returned value is considered as sensitive for this method call; $\zeta = a$ since the data is considered as being always sensitive; $\rho = \{$GDPR, CCPA, ...$\}$; and the tag-category $\tau = \{$unique_device_identifier$\}$.

As another example, we discuss the `getBitmap()` method of the `MediaStore.Images.Media` class in `android.provider`. This method returns a `Bitmap` object representing, e.g., a picture. The picture alone is not considered sensitive in itself since it is context-dependent. Indeed, if the image comes from a healthcare app, it is highly likely that it represents sensitive data (e.g., an MRI picture). However, if the app is about displaying random cat pictures, it is highly likely that the picture does not represent sensitive data. In this case σ = android.provider.MediaStore$Images$Media: android.graphics.Bitmap getBitmap(android.content.-ContentResolver, android.net.Uri); $\kappa = -1$ since the returned value is considered as sensitive for this method call; $\zeta = t$ since the data is considered as being sensitive depending on the context; $\rho = \{$GDPR, ...$\}$; $\tau = \{$photo$\}$.

VI. Perspectives and Community Project

With this paper, we start a community project to manually assemble lists of sensitive API methods. While we provide an initial list, this list is far from complete. The Android framework alone has over $40\,000$ public API methods, which is an arduous number of methods to go through manually as single person or single research group. Therefore, we set up a repository on GitHub with a collection of Android API methods categorized according to the definition of sensitive data and the definition language introduced in this paper. The lists that we provide serve as a starting point. We invite other researchers and practitioners to join our effort and complete this list – for Android, for other Java frameworks, as well as samples of sensitive API methods from other platforms.

VII. Conclusion

In this paper, we have presented a definition of sensitive data along with a definition language for tagging API methods that provide such data to client code, e.g., Android applications. We provide an initial categorized and annotated tool-agnostic list of methods to the research community and call for a collaborative effort to extend and maintain this list.

References

[1] IDC. (2022) Smartphone market share, https://www.idc.com/promo/smartphone-market-share/os. Accessed May 2022.

[2] Google. (2022) Permissions on android, https://developer.android.com/guide/topics/permissions/overview. Accessed May 2022.

[3] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: Analyzing the android permission specification," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 217–228.

[4] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, ser. SOUPS '12. New York, NY, USA: Association for Computing Machinery, 2012.

[5] Er1c_C. (2022) Complete dissection of an apk with a suspicious c2 server, https://lab52.io/blog/complete-dissection-of-an-apk-with-a-suspicious-c2-server/. Accessed May 2022.

[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," *SIGPLAN Not.*, vol. 49, no. 6, p. 259–269, Jun. 2014.

[7] 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," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 280–291.

[8] J. Samhi, A. Bartel, T. F. Bissyandé, and J. Klein, "Raicc: Revealing atypical inter-component communication in android apps," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1398–1409.

[9] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, jun 2014.

[10] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard, "Information flow analysis of android applications in droidsafe." in *NDSS*, vol. 15, no. 201, 2015, p. 110.

[11] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," *ACM Trans. Priv. Secur.*, vol. 21, no. 3, apr 2018.

[12] W. Wang, J. Wei, S. Zhang, and X. Luo, "Lscdroid: Malware detection based on local sensitive api invocation sequences," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 174–187, 2020.

[13] M. Junaid, D. Liu, and D. Kung, "Dexteroid: Detecting malicious behaviors in android apps using reverse-engineered life cycle models," *Computers & Security*, vol. 59, pp. 92–117, 2016.

[14] L. Luo, E. Bodden, and J. Späth, "A qualitative analysis of android taint-analysis results," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 102–114.

[15] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)," *Official Journal of the European Union*, vol. 119, pp. 1–88, 2016.

[16] S. Stummer, "Issues of verifying anonymity: An overview," in *INFORMATIK 2022*, D. Demmler, D. Krupka, and H. Federrath, Eds. Gesellschaft für Informatik, Bonn, 2022, pp. 179–194.

[17] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale," in *Trust and Trustworthy Computing*, S. Katzenbeisser, E. Weippl, L. J. Camp, M. Volkamer, M. Reiter, and X. Zhang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 291–307.

[18] X. Cui, D. Yu, P. Chan, L. C. K. Hui, S. M. Yiu, and S. Qing, "Cochecker: Detecting capability and sensitive data leaks from component chains in android," in *Information Security and Privacy*, W. Susilo and Y. Mu, Eds. Cham: Springer International Publishing, 2014, pp. 446–453.

[19] S. Y. Y. W. Y. Yao and H. W. Y. F. Y. X. Xiao, "Describectx: Context-aware description synthesis for sensitive behaviors in mobile apps," in *International Conference on Software Engineering (ICSE'22)*, 2022.

[20] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 1025–1035.

[21] L. Luo, F. Pauck, G. Piskachev, M. Benz, I. Pashchenko, M. Mory, E. Bodden, B. Hermann, and F. Massacci, "Taintbench: Automatic real-world malware benchmarking of android taint analyses," *Empirical Software Engineering*, vol. 27, no. 1, p. 16, Oct 2021.

[22] D. Zhu, H. Jin, Y. Yang, D. Wu, and W. Chen, "Deepflow: Deep learning-based malware detection by mining android application for abnormal usage of sensitive data," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, 2017, pp. 438–443.

[23] S. Lou, S. Cheng, J. Huang, and F. Jiang, "Tfdroid: Android malware detection by topics and sensitive data flows using machine learning techniques," in *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*, 2019, pp. 30–36.

[24] Z. Meng, Y. Xiong, W. Huang, L. Qin, X. Jin, and H. Yan, "Appscalpel: Combining static analysis and outlier detection to identify and prune undesirable usage of sensitive data in android applications," *Neurocomputing*, vol. 341, pp. 10–25, 2019.

[25] G. Russello, B. Crispo, E. Fernandes, and Y. Zhauniarovich, "Yaase: Yet another android security extension," in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, 2011, pp. 1033–1040.

[26] M. Backes, S. Bugiel, E. Derr, S. Weisgerber, P. McDaniel, and D. Octeau, "Poster: On demystifying the android application framework: Re-visiting android permission specification analysis."

[27] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 627–638.

[28] S. Arzt, S. Rasthofer, and E. Bodden, "Susi: A tool for the fully automated classification and categorization of android sources and sinks," *University of Darmstadt, Tech. Rep. TUDCS-2013-0114*, 2013.

[29] L. H. Tuan, N. T. Cam, and V.-H. Pham, "Enhancing the accuracy of static analysis for detecting sensitive data leakage in android by using dynamic analysis," *Cluster Computing*, vol. 22, no. 1, pp. 1079–1085, 2019.

[30] 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, 2015, pp. 426–436.

[31] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1043–1054.

[32] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, ser. SOAP '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1–6.

[33] B. Soewito and A. Suwandaru, "Android sensitive data leakage prevention with rooting detection using java function hooking," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 5, pp. 1950–1957, 2022.

[34] X. Pan, X. Wang, Y. Duan, X. Wang, and H. Yin, "Dark hazard: Learning-based, large-scale discovery of hidden sensitive operations in android apps." in *NDSS*, 2017.

[35] B. Andow, A. Acharya, D. Li, W. Enck, K. Singh, and T. Xie, "Uiref: Analysis of sensitive user inputs in android applications," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 23–34.

[36] J. Samhi, L. Li, T. F. Bissyande, and J. Klein, "Difuzer: Uncovering suspicious hidden sensitive operations in android apps," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. Association for Computing Machinery, may 2022.

[37] P. Lam, E. Bodden, O. Lhoták, and L. Hendren, "The Soot framework for Java program analysis: a retrospective," in *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*, vol. 15, 2011.

[38] R. Vallee-Rai and L. J. Hendren, "Jimple: Simplifying java bytecode for analyses and transformations," 1998.