Right to Know, Right to Refuse: Towards UI Perception-Based Automated Fine-Grained Permission Controls for Android Apps

Vikas K. Malviya
School of Computing and Information Systems, Singapore Management University
Singapore
vikasm@smu.edu.sg

Chee Wei Leow
School of Computing and Information Systems, Singapore Management University
Singapore
cwleow@smu.edu.sg

Ashok Kasthuri
School of Computing and Information Systems, Singapore Management University
Singapore
ashokk.2022@phdcs.smu.edu.sg

Yan Naing Tun
School of Computing and Information Systems, Singapore Management University
Singapore
yannaingtun@smu.edu.sg

Lwin Khin Shar
School of Computing and Information Systems, Singapore Management University
Singapore
lkshar@smu.edu.sg

Lingxiao Jiang
School of Computing and Information Systems, Singapore Management University
Singapore
lxjiang@smu.edu.sg

ABSTRACT

It is the basic right of a user to know how the permissions are used within the Android app's scope and to refuse the app if granted permissions are used for the activities other than specified use which can amount to malicious behavior.

This paper proposes an approach and a vision to automatically model the permissions necessary for Android apps from users' perspective and enable fine-grained permission controls by users, thus facilitating users in making more well-informed and flexible permission decisions for different app functionalities, which in turn improve the security and data privacy of the App and enforce apps to reduce permission misuses. Our proposed approach works in mainly two stages. First, it looks for discrepancies between the permission uses perceivable by users and the permissions actually used by apps via program analysis techniques. Second, it runs prediction algorithms using machine learning techniques to catch the discrepancies in permission usage and thereby alert the user for action about data violation. We have evaluated preliminary implementations of our approach and achieved promising fine-grained permission control accuracy. In addition to the benefits of users' privacy protection, we envision that wider adoption of the approach may also enforce better privacy-aware design by responsible bodies such as app developers, governments, and enterprises.

CCS CONCEPTS

• **Security and privacy → Software and application security**; **Privacy protections**; • **Software and its engineering**;

KEYWORDS

automated permission control, UI perception, Android application analysis, Android permissions, machine learning

ACM Reference Format:
Vikas K. Malviya, Chee Wei Leow, Ashok Kasthuri, Yan Naing Tun, Lwin Khin Shar, and Lingxiao Jiang. 2022. Right to Know, Right to Refuse: Towards UI Perception-Based Automated Fine-Grained Permission Controls for Android Apps. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3551349.3559556

ASE '22, October 10–14, 2022, Ann Arbor, Michigan, United States

ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3559556

1 INTRODUCTION

Mobile phones are indispensable in daily life. Besides being a strong communication medium, now they facilitate daily routine tasks via mobile apps. These apps use the personal information of users to personalize the functionality. But some apps misuse this function by secretly using the user's data for advertising or some other malicious work without the user's consent, because of which user's privacy is raising more and more concerns. "Do you allow the app to access your contacts, photos, media, files, messages, …" becomes a common question faced by users when they start to use an app, but such permission control mechanisms put too much burden on users (Hong 2017) [21]. Most users may not understand well the purposes of the accesses and the implications of granting permissions, and simply grant the permissions most of the time, leading to significant misuses of their privacy-sensitive data by apps.

It is unreliable to use the "reputation" of app developers to protect users' privacy for various reasons. For example, Facebook was reported to use "free" VPN apps and "research" apps that utilize a backdoor provided by Apple's Enterprise Developer Programme to collect participants' data without disclosure of its purposes (Constine 2019, Shealy 2019) [12, 37]. Google also used the backdoor to collect users' data, although it did better than Facebook in revealing itself and how the data collection works (Whittaker et al. 2019) [49].

It is too inflexible to grant permissions at the granularity level of apps. E.g., although it makes sense to allow a Messenger app to access microphone and camera during a video call, it opens the door

for the app to (mis)use the permissions for monitoring purposes. It would be better to grant permission for a functionality only when desired by the user following the principle of least privilege.

This paper aims to build up the capabilities that enable automated, finer-grained, and customizable permission controls that are aligned with users' perception of apps' functionalities according to what are perceivable from app user interfaces. This will promote a privacy ecosystem that keeps users aware, while reducing the burden on users, and pushes app developers to improve the privacy protection grades of their apps.

Our key research question is as follows: How can we automatically determine if a permission request to use privacy-sensitive data by an app is legitimate at any point of use? We aim to address the question from multiple views that model both users' perceptions of permissions necessary for apps and actual permissions used by apps, following the right-to-know design principle that users should be able to know apps' functionality and private data needs based on their interactions with app UIs and have the right-to-refuse too. More specifically, we aim to,

(1) Build a model that can automatically infer the permission needs according to extracted elements and contexts of apps' UIs related to the functionality visible by a user.
(2) Build a model that can automatically determine the permissions actually used by apps according to its code and infer the functionality enabled by the permissions. Then, the discrepancies between (1) the user perceived functionality and permission needs and (2) the actual functionality and permissions used in the code enables detection of (il)legitimate permission uses.
(3) Develop a dynamic permission manager that can customize permission controls according to each user's perceptions and historical interactions with apps, monitor app features (UIs and code traces) at run time, detect permission discrepancies and warn the user about potential , and suggest to the user if to allow or deny the permission use, or to fake private data if needed.

We have implemented the components of our approach using combination of dynamic and static analysis techniques and deep learning-based UI widget/image classification techniques. Our evaluation on more than 20 real-world apps show promising results: Our approach correctly inferred perceivable functionalities and permissions needed for about 66% of screenshots of app UIs from users' perspective; It also correctly inferred the permissions necessary for more than 90% of the code functionalities in apps represented as code graphs; We also successfully utilize a dynamic permission manager to dynamically allow or deny permissions or fake data at run time according to the discrepancies between the inferred permission needs and the actual permissions used: if an app uses a permission for a functionality that is imperceptible by the user, the use is denied as illegitimate. [1]

The rest of the paper is organized as follows. Section 2 surveys related work and discusses the uniqueness of our approach. Section 3 overviews our approach. Sections 4,5,6,7 give more details about the major components and preliminary evaluation results. Section 8 concludes with future work.

[1]This paper considers a dangerous permission request by an app as a use of privacy-sensitive data; in general, a use of private data may not be directly guarded by a permission request, but can still be traced via taint analysis.

2 RELATED WORK

Understanding and Managing Privacy: Much work has exploited ways to empower users with knowledge and capabilities to better understand and manage privacy, such as Jason Hong, Jin et al. and Chitkara et al. [11, 21, 24], for the use of sensitive data and permissions in Android apps.

Detection of Abnormal Use of Private Data: Li et al. [27] proposed a functional programming model, for accessing private data. Gorla et al. [19] used app descriptions to check the abnormal uses of sensitive APIs. Avdiienko et al. [4] built classification models using data flow to differentiate normal and abnormal uses of private data. BOXMATE by Jamrozik et al. [23] infers sandbox rules to restrict private data uses. Avdiienko et al. and, Hotzkow [5, 22] used "GUI mining" to cluster similar GUI elements across apps and detect outliers that trigger different APIs in code.

User Perception vs App Behaviour: Chen et al. [9] used Permission Event Graphs to specify policies on the interactions between user actions and permissions. Chen and Zhu) [10] used app functionalities to justify if a data transmission is a leak. Yang et al. and Fu et al. [16, 45]) used data flow information during GUI change event to infer if a data transmission is intended by a user. Fernandes et al. [15] inferred semantic annotations for UI elements on-device to avoid sending sensitive on-screen data off the device. Yang et al. [44] used conditions and events surrounding private data uses to differentiate malicious uses from benign ones. Pandita et al. [34] and Qu et al. [36] checked if the descriptions of an app are consistent with permission uses. Yu et al. [47, 48] extended their work with additional information such as the app's privacy policy bytecode, code features and natural language processing techniques. Wijesekera et al. [43] used context likely visible to a user for classification models and personalizes the models; they may be the first to infer privacy decisions automatically on a case-by-case basis at runtime without active user involvement. Nguyen et al. [32] measured user perception towards Android app behaviour on the basis of GUI elements to detect access of sensitive resources. Zhang et al. [51] developed a language for formalizing GUI policies and proposed abstraction named *event-driven layout forest* to perform static analysis for finding violations of sensitive information usage. Cao et al. [8] conducted a study of user's behavior towards permissions granting and denial of Android apps.

Detection of Illegitimate Use of Permission: Zhang et al. [50] used dynamic analysis to recognize permission (mis)uses in apps. Fu et al. [17] used app code and foreground UIs to build classifiers that differentiate legitimate and illegitimate permission uses. SmarPer (Olejnik et al.) [33] used 32 types of context information (e.g., time, location, app name) to build classification models.

Privacy-by-Design: For the purpose of assisting smartphone app developers in comprehending and implementing the most relevant privacy and security concepts, Majid Hatamian [20] proposed a design guide. The stated legal concepts are mapped in this guide to workable privacy and security solutions. It supports developers in ensuring greater privacy that complies with current legal requirements. Sokolova and Lemercier [38] examined the existing privacy situation of mobile systems with reference to mobile operating systems and mobile apps, classifying privacy violation possibilities into three classes. They concluded by presenting a few patterns that

assist mobile application developers in implementing the Privacy by Design principle. A novel authorization model appropriate for mobile applications that respects Privacy by Design was introduced by Sokolova et al. [39]. The authors demonstrated how such a modified permission system might enhance not only the security but also the transparency and consent of mobile applications.

These related studies have the following limitations:

- Most of the studies (except for a few [43]) are coarse-grained, making decisions at app-level or library/package-level, and not flexible enough for different app usages;
- Most studies based on classification techniques need to assume some apps are "benign" beforehand, which is unreliable.
- The studies have not utilized links among user-understandable GUI features, app functionalities, and norms of private data uses across apps for legitimacy decisions.

Our approach and vision aim to develop novel techniques that have the following advantages:

- No need of "benign" apps, via cross-validation of legitimacy models across many apps to infer norms of minimal private data uses with minimal user labelling;
- More fine-grained classification of GUI features and app functionalities, via deep learning of user-perceivable UI elements and contexts, code patterns, and permission uses in each app;
- More accurate correlation among GUI features and private data uses, via new feature classification techniques that use both static and dynamic information of an app to relate UI elements and contexts and user perceptions to relevant code in the app;
- More understandable permission legitimacy decisions, via UI perception-based matching and comparison across UI, code, and private data uses in apps and individual users' preferences.

3 OVERVIEW OF SOLUTION APPROACH

An overview of our approach is shown in Figure 1. It has two main stages: building permission legitimacy (control) models, and applying the models. The first stage infers the norms of dangerous permission uses (and thus privacy-sensitive data uses) across a set of apps. It is to be done on servers that have full-control of the devices to analyse and test Android apps' GUIs and bytecodes. UI elements and contexts and data usage patterns in code are extracted and related to each other to construct legitimacy/control models of private data uses. We implement it from three aspects.

- The first aspect is dynamic analysis of apps. We extract screenshots (as images and UI element layouts) and code execution traces (e.g., API call sequences) from an app. Such data is used to train deep learning models in a later step to identify UI features and permission uses perceivable by users.
- The second aspect is to utilize user perceptions towards permissions and UI features (e.g., icons, widgets, their functionalities and intents). Such perceptions can be curated via user studies of real-world apps and/or via automated inference from trained machine learning models (Section 5).
- The third aspect is static analysis of apps. It produces various kinds of code graphs (e.g., control-flow graphs, call graphs, dependence slices, window-transition graphs) linked to UI elements (e.g., triggered by a user action via the app UI). Such graphs can be used to train deep learning models to classify code functionalities and permission needs and to check if they are consistent with users' perception via UIs. At the same time, static analysis can identify all permissions that may be potentially used in the app code, although with more false alarms.

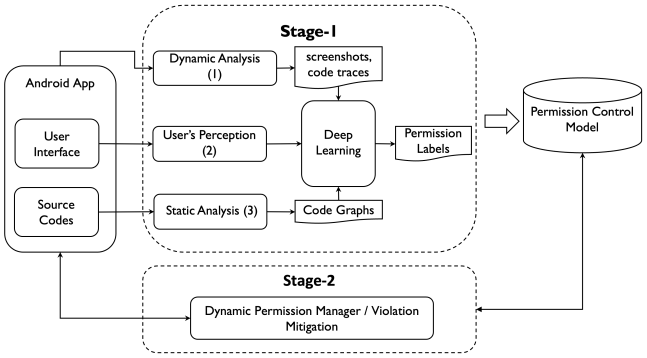

Figure 1: Overview of Approach

The three aspects can thus produce different but related views of the functionalities of an app and its permission needs. Following the principles of right-to-know and least privilege, we use the view perceivable by users via UIs as the "legitimate" functionalities and permissions; the discrepancies between this UI view and the other views indicate potential violations in app code, which are used as the decision guideline for forming our Permission Control Model.

The second stage is intended for developing a permission manager that, based on the Permission Control Model, helps users to have fine-grained controls of permission requests at runtime. The second stage also utilizes dynamic monitoring and analysis and consists of a violation detector and mitigator. It monitors the app features (UI elements, code API call traces, and permission requests) and utilizes the models from the first stage and compares them on the fly. In case of discrepancies, it warns the user about that and provides mitigation options (e.g., to deny the permission request, or to fake the private data guarded by the permission) which can be saved and tailored for future uses.

Our approach can be used by an app store to infer and apply the permission legitimacy/control models to detect if an app contains illegitimate permission uses. It can also be used as a system app (with privileges) or a standalone app on (unrooted) devices to apply the inferred models on the fly to analyse if each use by an app on-device is illegitimate (i.e., not perceivable or not controllable by the user at run time) and customize mitigation options. The components of the stages are explained further in the next sections.

4 DYNAMIC ANALYSIS OF ANDROID APPS

Dynamic Analysis of Android apps is done to obtain screenshots and code execution traces at run time. These will be utilized in training the deep learning models during the first stage and also in developing a permission manager during the second stage.

We adapt DroidMate-2 [6] for our purpose during the first stage as it can achieve relatively high coverage in comparison with related work. Based on the screenshot images and UI elements extracted, we trained a deep learning model based on image and text embedding and multi-class classification using the Keras library [35], together with data from user perceptions as labels (Section 5). The model

can take images and UI elements of an app as input and produce as output the functionality descriptions and permissions for the app that is likely user-perceivable via UIs. We evaluated 89 applications having 76 benign and 13 malicious apps for uses of 9 sensitive permissions. We achieved on average 66% precision and 56% recall. We are now conducting larger-scale evaluations with more apps and more user labels to enhance the results.

5 USER PERCEPTION CURATION

We collect users' perceptions for GUI elements, their features and the permission(s) needed for them. We conducted a user study through an online survey with 102 participants for screenshots taken from about 20 apps commonly used in our city [46]. There are 144 Yes-No Questions, 30 Multi-Selection questions, and 17 UI Design Preference questions. The questions in the user study were formatted with inspiration from the previous work [29]. We also cross-validated the Yes-No MCQ answers from the 14 participants with only Android experience, 14 participants with only iPhone experience, and 74 participants who are experienced with both.

Through the answers to the questions, we want to find out how well each UI element expresses its intent to use certain permissions for users to understand. As a summary, participants on average identified only 55.6% of the permissions used in the apps based on the provided UI images. This shows that, the UI does not accurately express the intent of permission uses. If we set the threshold for "Good" UI at 70% agreement among participants (i.e., more than 70% of the participants indicate that the UI expresses the need of a permission), we got 100 of such "Good" UI images for training, and the remaining 44 UI images and also screenshots from malware apps were considered as "Poor" examples.

Besides this study, we expand our study with recent related work [32]. They measured user perception towards Android app behaviour on the basis of GUI elements via user studies. Then they developed a tool named GUIBAT to detect undesired access of sensitive resources. We are also using their dataset of UI elements for expanding our study and training deep learning models.

6 STATIC ANALYSIS WITH GRAPH REPRESENTATION LEARNING

6.1 Static Program Analysis

Static analysis is a fundamental element in program analysis and is used to debug, examine the source code before the program execution or run. We used Soot [40] and other tools derived from it, such as FlowDroid [3] and Gator [26], to load Android apps and to generate the graphs such as call graph, CFG, PDG, SDG, Window-Transition Graphs. These graphs act as basic building blocks for our deep learning models of code in our work. We also perform slicing [41] on the graphs to reduce code that is irrelevant to permission uses or user inputs via UI event handlers, so that the sliced graphs can be more accurate in representing permission or UI-relevant code functionalities for training deep learning models.

6.2 Whole Graph Embedding based Prediction

Graph embedding techniques [7] are to turn graphs into numerical encoding vectors based on the graph structures and their implicit features, so that the graph vectors can be easily used as features to train machine learning models to detect a desired property of the graphs. We ran whole graph embedding on the inter-procedural control-flow graphs for Android apps to obtain the feature vectors using the NetworkX [31] library and GL2vec [18] in the karateclub machine learning library [25].

We applied the graph vectors on various classification models to classify the code functionalities of the graphs and their permission mappings, evaluated with precision and recall metrics. For the same 20 apps used in Section 5 and also 20 malicious apps obtained from DeepIntent [13], we assumed all code functionalities and permission uses in the malwares are illegitimate, but ran a malware scanning tool Androwarn [2] on the "benign" apps to detect any malicious behaviours; only code graphs that do not contain any reported malicious behaviors are considered legitimate and used for our training. Thus, a graph is considered as legitimate if there is no malicious behaviour detected (with a label 0), and is considered as illegitimate if there are any malicious behaviours found (with a label 1). We intend to further engage annotators and utilize user studies to enrich the labels for both UI screenshots and code graphs, to minimize training biases and improve the quality of the graph labels and thus deep learning models.

Table 1 shows our preliminary results. We used sklearn ensemble library [14] for Ensemble Graph Classification. A VotingClassifier was created based on majority voting from seven commonly used machine learning classifiers. The Ensemble model resulted in a promising precision of up to 92% for detecting code graphs that may contain illegitimate (i.e., malicious or imperceptible by users via UIs) permission uses.

Table 1: Ensemble Classification Report

	Precision	Recall	F1 Score
Label 0 (legitimate)	0.89	0.93	0.91
Label 1 (illegitimate)	0.92	0.88	0.95
Weighted Avg	0.91	0.91	0.90

To reduce the need of manually labelled user perceptions, we also envision the use of unsupervised learning models according to similarity and difference among code graphs and UIs [1, 28] for detecting abnormal permission uses that may violate the right-to-know and least privilege principles.

7 DYNAMIC PERMISSION MANAGER

Dynamic permission managers, such as UIPDroid [30], can be employed in the second stage for detecting discrepancies in permission usage. To enable dynamic monitoring and permission management on unrooted devices, we can utilize app virtualization frameworks (e.g., VirtualXposed [42] used by UIPDroid) in an architecture like Figure 2. Android API methods can be hooked using VirtualXposed. Whenever any app wants to access private data of user, it has to call platform APIs which are secured by the Android permission mechanism. API hooks to these method calls can check whether the permission should be granted based on the Permission Control Model from the first stage and/or users' customized preferences. Users' decisions are recorded and stored for the future incidents.

Permission Configuration UIs can be used to change the preferences. Permission Configuration UIs can also import decision from other analysis tools or from other experienced persons.

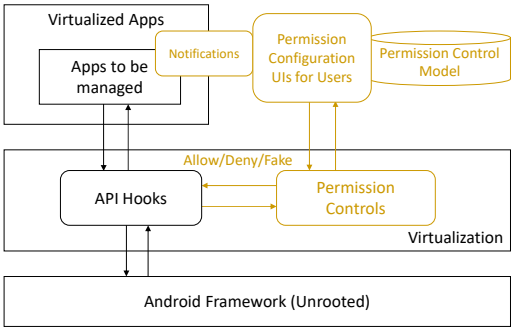

Figure 2: Virtualization-based Permission Control

We have evaluated our Permission Control Model for the same Android apps used in Section 5 using UIPDroid on a Google Pixel 4A phone with Android 10. As UIPDroid currently hooks only the APIs that use contact and location permissions, we only consider the parts of the control model that involve contact and location permissions. Out of about 20 apps, UIPDroid failed to hook 11 apps due to limitations of the VirtualXposed frameworks and anti-analysis techniques used in the apps. For rest of the apps UIPDroid successfully enabled permission control according to the decisions given by our Permission Control Model, its execution overhead was also minimal as it did not affect user experience. This preliminary result shows the usefulness of our Permission Control Model together with a dynamic permission manager like UIPDroid for automated fine-grained permission management.

8 CONCLUSION AND FUTURE WORK

An approach and a vision for fine-grained permission controls in Android apps is presented in this work, guided by the right-to-know, right-to-refuse, and least-privilege principles. Illustrated with preliminary implementations and evaluations, this approach, together with its supporting models, aims to automatically determine the legitimacy of each use of private data by analysing app UI and code functionality linked to it. The components of this approach can be customised as per users' needs. We are aware of various limitations in the components of our two-stage approach but have confidence that they can be improved with better analysis and learning techniques for UI and code as a part of future work.

In more general, the approach can be useful for various scenarios:

- As a gatekeeper for application stores to build the norms of private data uses and analyse if the features of an app in the store comply with the norms.
- As a system app (with system privileges) shipped with mobile systems to analyse the features of an app on-device against the norms and manage private data uses on the fly.
- As a standalone app on an (unrooted) user's device to analyse the features of an app on the device against the norms and report and/or block anomalies. The user can choose to customize the norms used and enable/disable the analysis for each app.

The above usage scenarios can help to shift the burden of privacy protection from end-users to app developers, to app stores, to system developers, to OEM vendors, etc. [21] by identifying features in their apps that unnecessarily use private data and improving the privacy protection ecosystem for mobile systems and apps.

ACKNOWLEDGMENTS

This work is supported by the National Research Foundation Singapore, under the National Satellite of Excellence in Mobile System Security and Cloud Security (NRF2018NCR-NSOE004- 0001).

REFERENCES

[1] Sondus Almrayat, Rana Yousef, and Ahmad Sharieh. 2019. Evaluating the Impact of GUI Similarity between Android Applications to Measure their Functional Similarity. *International Journal of Computer Applications* 178 (06 2019), 31–38. https://doi.org/10.5120/ijca2019919075

[2] Androwarn 2019. Androwarn: Yet another static code analyzer for malicious Android applications. https://github.com/maaaaz/androwarn.

[3] Secure Software Engineering Group at Paderborn University and Fraunhofer IEM. 2022. *FlowDroid Data Flow Analysis Tool.* Retrieved May 19, 2022 from https://github.com/secure-software-engineering/FlowDroid

[4] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2015. Mining Apps for Abnormal Usage of Sensitive Data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 426–436. https://doi.org/10.1109/ICSE.2015.61

[5] Vitalii Avdiienko, Konstantin Kuznetsov, Isabelle Rommelfanger, Andreas Rau, Alessandra Gorla, and Andreas Zeller. 2017. Detecting Behavior Anomalies in Graphical User Interfaces. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 201–203. https://doi.org/10.1109/ICSE-C.2017.130

[6] Nataniel P. Borges Jr., Jenny Hotzkow, and Andreas Zeller. 2018. *DroidMate-2: A Platform for Android Test Generation.* Association for Computing Machinery, New York, NY, USA, 916–919. https://doi.org/10.1145/3238147.3240479

[7] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[8] Weicheng Cao, Chunqiu Xia, Sai Teja Peddinti, David Lie, Nina Taft, and Lisa M. Austin. 2021. A Large Scale Study of User Behavior, Expectations and Engagement with Android Permissions. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 803–820. https://www.usenix.org/conference/usenixsecurity21/presentation/cao-weicheng

[9] Kevin Zhijie Chen, Noah M. Johnson, Vijay Victor D'Silva, Shuaifu Dai, Kyle MacNamara, Thomas R. Magrino, Edward XueJun Wu, Martin C. Rinard, and Dawn Xiaodong Song. 2013. Contextual Policy Enforcement in Android Applications with Permission Event Graphs. In *NDSS*.

[10] Xin Chen and Sencun Zhu. 2015. DroidJust: automated functionality-aware privacy leakage analysis for Android applications. *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks* (2015).

[11] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I. Hong, and Yuvraj Agarwal. 2017. Does This App Really Need My Location?: Context-Aware Privacy Management for Smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1 (09 2017), 1–22. https://doi.org/10.1145/3132029

[12] Josh Constine. 2019. *Facebook pays teens to install VPN that spies on them.* Retrieved May 19, 2022 from https://techcrunch.com/2019/01/29/facebook-project-atlas/

[13] DeepIntent 2021. DeepIntent: Deep Icon-Behavior Learning for Detecting Intention-Behavior Discrepancy in Mobile Apps. https://github.com/deepintent-ccs/DeepIntent.

[14] Ensemble 2021. sklearn Ensemble Module. https://scikit-learn.org/stable/modules/ensemble.html.

[15] Earlence Fernandes, Oriana Riva, and Suman Nath. 2016. Appstract: On-The-Fly App Content Semantics With Better Privacy. In *ACM Annual International Conference on Mobile Computing and Networking (MobiCom'16)* (acm annual international conference on mobile computing and networking (mobicom'16) ed.). ACM. https://www.microsoft.com/en-us/research/publication/appstract-on-the-fly-app-content-semantics-with-better-privacy/

[16] Hao Fu, Zizhan Zheng, Aveek K. Das, Parth H. Pathak, Pengfei Hu, and Prasant Mohapatra. 2016. FlowIntent: Detecting Privacy Leakage from User Intention to Network Traffic Mapping. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 1–9. https://doi.org/10.1109/SAHCN.2016.7732993

[17] Hao Fu, Zizhan Zheng, Sencun Zhu, and Prasant Mohapatra. 2019. Keeping Context In Mind: Automating Mobile App Access Control with User Interface Inspection. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2089–2097. https://doi.org/10.1109/INFOCOM.2019.8737510

[18] GL2Vec 2019. GL2vec: Graph Embedding Enriched by Line Graphs with Edge Features. https://link.springer.com/chapter/10.1007/978-3-030-36718-3_1.

[19] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking App Behavior Against App Descriptions. In *ICSE '14: Proceedings of the 2014 International Conference on Software Engineering* (Hyderabad, India). ACM Press, 292–302.

[20] Majid Hatamian. 2020. Engineering Privacy in Smartphone Apps: A Technical Guideline Catalog for App Developers. *IEEE Access* 8 (2020), 35429–35445. https://doi.org/10.1109/ACCESS.2020.2974911

[21] Jason Hong. 2017. The Privacy Landscape of Pervasive Computing. *IEEE Pervasive Computing* 16, 3 (2017), 40–48. https://doi.org/10.1109/MPRV.2017.2940957

[22] Jenny Hotzkow. 2017. Automatically inferring and enforcing user expectations. *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (2017).

[23] Konrad Jamrozik, Philipp von Styp-Rekowsky, and Andreas Zeller. 2016. Mining Sandboxes. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 37–48. https://doi.org/10.1145/2884781.2884782

[24] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Kumar Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. 2018. Why Are They Collecting My Data?: Inferring the Purposes of Network Traffic in Mobile Apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2 (12 2018), 1–27. https://doi.org/10.1145/3287051

[25] Karateclub 2021. Karateclub Whole Graph Embedding. https://karateclub.readthedocs.io/en/latest/modules/root.html.

[26] Gyutak Kim. 2022. *GATOR: Program Analysis Toolkit For Android.* Retrieved May 19, 2022 from http://web.cse.ohio-state.edu/presto/software/gator

[27] Yuanchun Li, Fanglin Chen, Toby Jia-Jun Li, Yao Guo, Gang Huang, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. 2017. PrivacyStreams: Enabling Transparency in Personal Data Processing for Mobile Apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1 (09 2017), 76:1—-76:26. https://doi.org/10.1145/3130941

[28] Jian Mao, Jingdong Bian, Hanjun Ma, Yaoqi Jia, Zhenkai Liang, and Xuxian Jiang. 2018. Robust Detection of Android UI Similarity. In *2018 IEEE International Conference on Communications (ICC)*. 1–6. https://doi.org/10.1109/ICC.2018.8422189

[29] Kristopher Micinski, Daniel Votipka, Rock Stevens, Nikolaos Kofinas, Michelle Mazurek, and Jeffrey Foster. 2017. User Interactions and Permission Use on Android. 362–373. https://doi.org/10.1145/3025453.3025706

[30] Duan Mulin, Jiang Lingxiao, Shar Lwin, Khin, and Gao Debin. 2022. UIPDroid: Unrooted Dynamic Monitor of Android App UIs for Fine-Grained Permission Control. In *2022 International Conference on Software Engineering*.

[31] NetworkX 2021. NetworkX Network Analysis. https://networkx.org/documentation/stable/tutorial.html.

[32] Trung Tin Nguyen, Duc Cuong Nguyen, Michael Schilling, Gang Wang, and Michael Backes. 2021. Measuring User Perception for Detecting Unexpected Access to Sensitive Resource in Mobile Apps. *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security* (2021).

[33] Katarzyna Olejnik, Italo Dacosta, Joana Soares Machado, Kévin Huguenin, Mohammad Emtiyaz Khan, and Jean-Pierre Hubaux. 2017. SmarPer: Context-Aware and Automatic Runtime-Permissions for Mobile Devices. In *2017 IEEE Symposium on Security and Privacy (SP)*. 1058–1076. https://doi.org/10.1109/SP.2017.25

[34] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. 2013. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, Washington, D.C., 527–542. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/pandita

[35] Konstantin Pogorelov, Kristin Ranheim Randel, Carsten Griwodz, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Concetto Spampinato, Duc-Tien Dang-Nguyen, Mathias Lux, Peter Thelin Schmidt, et al. 2017. Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. 164–169.

[36] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhongchao Chen. 2014. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014).

[37] Matt Shealy. 2019. *What Facebook's VPN scandal tells us about its approach to data privacy.* Retrieved May 19, 2022 from https://channels.theinnovationenterprise.com/articles/facebook-pays-teens-to-download-a-vpn-that-spy-s-on-their-activity

[38] Karina Sokolova and Marc Lemercier. 2013. Privacy by Design in Mobile Devices. In *4ème Atelier sur la Protection de la Vie Privée (APVP)*. Les Loges-en-Josas, France. https://hal-utt.archives-ouvertes.fr/hal-02274620

[39] Karina Sokolova, Marc Lemercier, and Jean-Baptiste Boisseau. 2014. Respecting user privacy in mobiles: privacy by design permission system for mobile applications. *International Journal On Advances in Security* 7, 3-4 (2014), 110–120. https://hal-utt.archives-ouvertes.fr/hal-02274681

[40] Soot 2021. Soot Program Analysis Framework. http://soot-oss.github.io/soot/.

[41] Mark Weiser. 1984. Program Slicing. *IEEE Transactions on Software Engineering* SE-10, 4 (1984), 352–357. https://doi.org/10.1109/TSE.1984.5010248

[42] Weishu. 2022. *VirtualXposed.* Retrieved May 17, 2022 from https://github.com/android-hacker/VirtualXposed

[43] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David Wagner, and Konstantin Beznosov. 2017. The Feasibility of Dynamically Granted Permissions: Aligning Mobile Privacy with User Preferences. In *2017 IEEE Symposium on Security and Privacy (SP)*. 1077–1093. https://doi.org/10.1109/SP.2017.51

[44] Wei Yang, Xusheng Xiao, Benjamin Andow, Sihan Li, Tao Xie, and William Enck. 2015. AppContext: Differentiating Malicious and Benign Mobile App Behaviors Using Context. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 303–313. https://doi.org/10.1109/ICSE.2015.50

[45] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and Xiaoyang Sean Wang. 2013. AppIntent: analyzing sensitive data transmission in android for privacy leakage detection. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013).

[46] Apps For You. 2022. *Singapore Smart Nation Apps.* Retrieved July 18, 2022 from https://www.smartnation.gov.sg/community/apps-for-you

[47] L. Yu, X. Luo, C. Qian, S. Wang, and H. N. Leung. 2018. Enhancing the Description-to-Behavior Fidelity in Android Apps with Privacy Policy. *IEEE Transactions on Software Engineering* 44, 09 (sep 2018), 834–854. https://doi.org/10.1109/TSE.2017.2730198

[48] Le Yu, Tao Zhang, Xiapu Luo, and Lei Xue. 2015. AutoPPG: Towards Automatic Generation of Privacy Policy for Android Applications. *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices* (2015).

[49] Whittaker Zack, Constine Josh, and Lunden Ingrid. 2019. *Google will stop peddling a data collector through Apple's back door.* Retrieved May 19, 2022 from https://techcrunch.com/2019/01/30/googles-also-peddling-a-data-collector-through-apples-back-door/

[50] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, Xiaoyang Sean Wang, and Binyu Zang. 2013. Vetting undesirable behaviors in android apps with permission use analysis. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013).

[51] Zhen Zhang, Yu Feng, Michael D. Ernst, Sebastian Porst, and Isil Dillig. 2021. Checking conformance of applications against GUI policies. In *ESEC/FSE 2021: The ACM 29th joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. Athens, Greece.