UIPDroid: Unrooted Dynamic Monitor of Android App UIs for Fine-Grained Permission Control

Mulin Duan, Lingxiao Jiang, Lwin Khin Shar, Debin Gao
School of Computing and Information Systems
Singapore Management University
mlduan.2020@msc.smu.edu.sg, {lxjiang, lkshar, dbgao}@smu.edu.sg

ABSTRACT

Proper permission controls in Android systems are important for protecting users' private data when running applications installed on the devices. Currently Android systems require apps to obtain authorization from users at the first time when they try to access users' sensitive data, but every permission is only managed at the application level, allowing apps to (mis)use permissions granted by users at the beginning for different purposes subsequently without informing users. Based on privacy-by-design principles, this paper develops a new permission manager, named UIPDroid, that (1) enforces the users' basic right-to-know *through user interfaces* whenever an app uses permissions, and (2) provides a more *fine-grained UI widget-level* permission control that can allow, deny, or produce fake private data dynamically for each permission use in the app at the choice of users, even if the permissions may have been granted to the app at the application level. In addition, to make the tool easier for end users to use, unlike some other root-based solutions, our solution is *root-free*, developed as a module on top of a virtualization framework that can be installed onto users' device as a usual app. Our preliminary evaluation results show that UIPDroid works well for fine-grained, per-widget control of contact and location permissions implemented in the prototype tool, improving users' privacy awareness and their protection. The tool is available at https://github.com/pangdingzhang/Anti-Beholder; A demo video is at: https://youtu.be/dT-mq4oasNU

KEYWORDS

Android, Permission Management, Rootless, VirtualXposed

1 Introduction

While users enjoy the convenience brought by mobile applications (a.k.a. apps), they also have to bear threats like data leakage [1]. To protect users' privacy, Google uses a permission request mechanism in their Android systems that apps must declare permissions needed and send a request to users before they access the data controlled by the permissions for the first time [2]. For each permission request, users only have two choices, namely, *Allow* or *Deny*. Once users choose *Allow* for an app, the app gets access to users' data and the system will not check whether the subsequent permission uses are still allowed by the users or inform users about those uses. Once users choose *Deny*, the app often disables the functionality that may be wanted by the users. [1]

Such a permission control mechanism is far from enough in protecting users' privacy. Based on privacy-by-design and UI-design principles [3, 4, 5], an app should respect users' right to know whenever it needs to use private data and explain sufficiently to the users about its purpose of the uses (e.g., through a notification message on UI, short animation, flash lights /audio/vibration on device, etc.), and provide users the right to withdraw either partially for some parts of the apps or temporarily whenever they want to. Stock Android still lacks such features for user awareness and control of the uses of sensitive data after authorization [1]. In other words, apps can use permissions such as Contacts, Locations, etc. rampantly anytime for any purpose after the permissions are granted to them without informing users; they may simply fail to run if users do not grant permissions even though not all the functionalities of the apps need the permissions. In short, the Android permission manager is too coarse-grained and not flexible enough to protect user privacy at different granularity levels when users are using apps for different purposes.

To help users safely use third-party apps and reduce their concerns about privacy leaks, we need a fine-grained solution that can distinguish apps' different uses of a permission with respect to users' preferences and provide useful information and mechanisms for users to easily make suitable and adjustable choices [6, 7, 8]. Furthermore, users should still be able to use (or at least try out) app functionalities even if they do not grant permissions to an app. Lastly, the solution should be easy to install and use by end users without requiring rooting an Android phone.

Towards the above objectives, this paper proposes a tool, named UIPDroid, as a module developed on top of a virtualization framework called VirtualXposed [9] and can be installed like a usual app on *unrooted* Android phones of compatible versions. When an app tries to access a user's private data that requests for a permission, UIPDroid intercepts the requests and checks the UI context of this behavior, informs the user about the potential permission use through UI notification, and provides choices for the user to set their preferences. The notification and decision making can be done for each use per UI widget, depending on if the use is

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM 978-1-4503-9223-5/22/05...$15.00
https://doi.org/10.1145/3510454.3516844

[1] In recent Android versions, there are a few more choices for users: "Allow all the time," "Allow only while in use," and "Ask every time." But the essence of the discussion in this paper remains valid.

triggered by the user's actions on the UI or by the app's internal behavior (e.g., an operation done in a background service of the app). In addition to Allow and Deny, UIPDroid gives another option, *Fake*, to allow the permission use but return pseudo-random values to the requesting app to continue its functionality. Also, UIPDroid records all the apps' permission uses and can present the logs to users visually for postmortem inspection. Further, UIPDroid has an interface that can export/import predefined permission preferences for a user and automate the permission choices based on the UI context and users' preferences to help reduce the cognitive and decision burdens on users.

We envision that UIPDroid can be used by any end user who has basic knowledge of Android app installation and settings. Although it is a prototype, we believe that such a dynamic fine-grained permission notification and setting tool can help to raise users' awareness of privacy. We also believe that such fine-grained control mechanism could and should be integrated into future versions of Android for better privacy protection.

The rest of the paper is organized as follows. Sec. 2 illustrates a usage example of UIPDroid. Sec. 3 explains its design and implementation. Sec. 4 presents small-scale evaluation results and limitations for future improvements. Sec. 5 concludes this work.

2 Usage Example

To use UIPDroid, users should download and install the VirtualXposed app [9] and our module, which are as easy as installing a normal app. After that, users can add the apps whose permission uses need to be managed into VirtualXposed. The step-by-step instructions can be found in README in the tool's repository.

By default, UIPDroid generates fake values for all the permission requests of an app. When the apps or their UI widgets trigger a permission use for the first time, our tool pushes notifications to user and waits for user's decision. The user decisions are stored for future checks and can be changed in user's setting page.

2.1 Notification

For a permission request that UIPDroid cannot find in users' setting, it pushes a notification (Figure 1) to ask for their choices. Even if users have set up their preferences, UIPDroid can still send toast messages to make users aware of apps' behaviours.

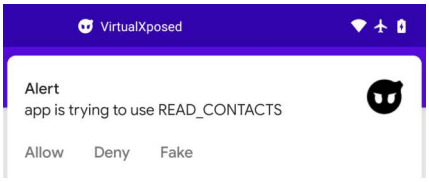

Figure 1: Notification

2.2 Setting

Users can manage their permission preferences for different apps or UI widgets. Basic Permission Management (Figure 2) lists all permissions requested by an app, which is similar to Android system's native permission manager. Users can switch to Widget-Level Permission Management (Figure 3) in which UIPDroid lists permissions triggered by each widget in a format of {`Permission_Name_WidgetID`}. One permission can appear multiple times under this widget-level setting as the same permission may be linked to different UI widgets, through which we enable more fine-grained control for user-aware permission uses (cf. Section 3.2.2).

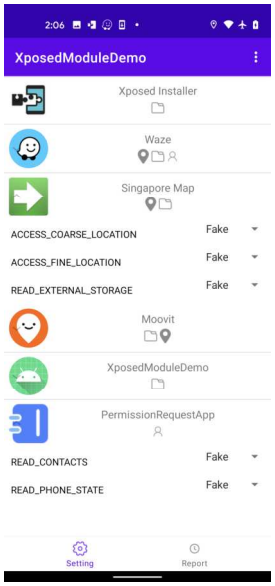

Figure 2: Basic Permission Management

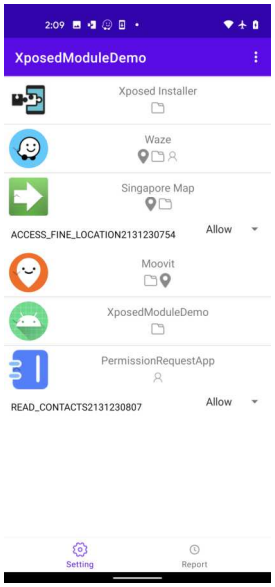

Figure 3: Widget-Level Permission Management

2.3 Report

Users can inspect the logs of apps' permission uses visually in two formats. One is Timeline Report (Figure 4) that displays all permission uses in a reverse chronological order. The other is Summary Report (Figure 5) that lists the total numbers of times each permission is used and the last access time. Users can filter the reports by a specific permission or an application.

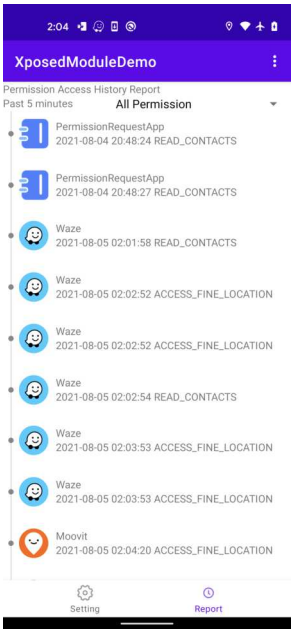

Figure 4: Timeline Report

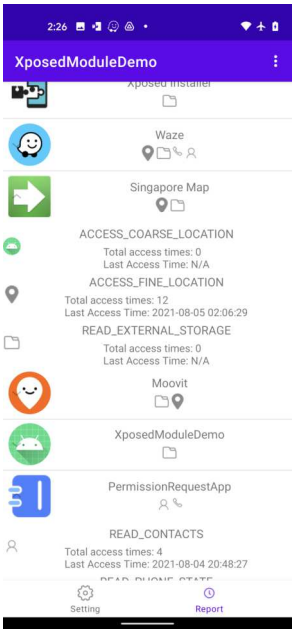

Figure 5: Summary Report

3 Design & Implementation

This section introduces the architecture of UIPDroid and describes major technical details in its design and implementation.

3.1 Architecture

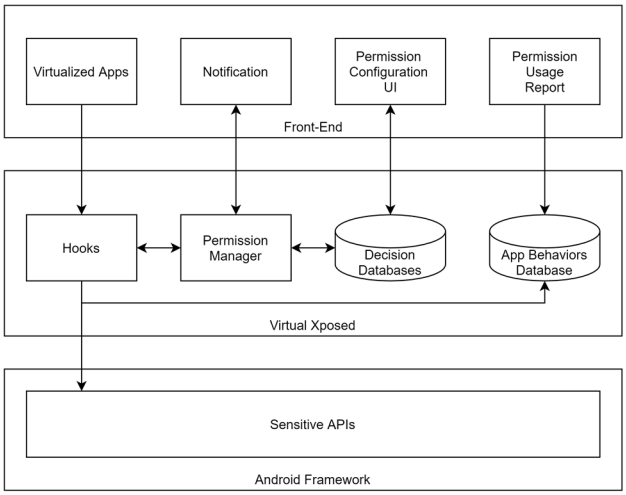

Figure 6: Overview of Architecture

Figure 6 illustrates the architecture of UIPDroid. This tool is built on the VirtualXposed [9] providing the ability to hook any Java method invoked by apps that are executed within the VirtualXposed sandbox. When an app tries to access users' private data in Android, it needs to call corresponding APIs of the Android system. For example, apps often call the method, `android.location.LocationManager.getLastKnownLocation`, to get users' locations, and call the method, `android.content.ContentResolver.query`, to retrieve contacts. UIPDroid implements the hooks to intercept such method calls and uses custom-built *Permission Manager* to check whether each request should be granted with permissions. If Permission Manager does not find existing users' preferences, it will push a notification to ask users for their decisions and record them in the Decision Databases. Users can also change their preferences any time via the Permission Configuration UI or by importing decisions suggested by other analysis tools or experts. Meanwhile, all the permission requests and decision results are recoded in the App Behaviours Database and can be displayed via Permission Usage Report for users' inspection.

3.2 Permission Management

From the Android system's perspective, VirtualXposed (VX) runs like a normal app, and all permission requests from the apps running in the VX sandbox are like permission requests from the VX app itself. Thus, the VX app should be given all the permissions that may be needed by the apps that users want to manage; then, UIPDroid's Permission Manager implements more fine-grained permission controls for the apps. If VX is not given a permission, UIPDroid also denies the permission for the managed apps.

3.2.1 Basic permission management. For each request for Contact or Location permissions from an app, UIPDroid implements three decision choices: Allow, Fake, and Deny. UIPDroid intercepts calls to sensitive APIs such as `getLastKnownLocation` and `query` as illustrated in Section 3.1 to check users' preferences. For Allow, the calls proceed per usual; for Deny, the calls are aborted with a SecurityException; for Fake, the calls still proceed but UIPDroid intercepts their return values and replace them with semi-random values with the same data types and formats.

In addition, some apps follow Android's privacy best practices [2]: they check whether they have the permission (e.g., by calling `ContextCompat.checkSelfPermission`) before calling the sensitive APIs; if permission is not given, they may abort gracefully by themselves, or issue a request for the permission at runtime (e.g., by calling `ActivityCompat.requestPermissions` and waiting for users' response via the `onRequestPermissionsResult` callback). Thus, we also hook `checkSelfPermission` and `onRequestPermissionsResult` and replace their decision results according to our Decision Databases and users' preferences so that the apps can still abort gracefully on their own when the permission is denied.

3.2.2 Widget-level Permission management. Following the privacy-by-design principles, private data uses should have relevant UI widgets that are linked to user actions and inform users via UI status changes. Thus, UIPDroid uniquely relates permission controls to UI widgets. i.e., it allows users to set permissions based on visible UI widgets (e.g., clickable buttons and scrollable list view). We hook each of the event handlers of the UI widgets (e.g., `onClick` and `onScroll`) that would trigger calls to sensitive APIs and inject our permission control per Section 3.2.1, so that we enable widget-level permission control and allow users to allow/deny/fake the permission use *before* each sensitive API call. Permission requests that do not relate to any UI widget are defaulted to Fake if the permission is given to VirtualXposed for management (handled as Section 3.2.1) or Deny otherwise, as such permission requests without user awareness should not be allowed to use private data.

An important issue here is to correctly relate a potential permission request to a UI widget that interacts with users so that the hooks are placed in the right event handlers for users' control. In our current implementation, we focus on only clickable buttons by manually running the subject apps separately and analyzing their execution profiles. In particular, we identify the unique button ID of an app and the permission triggered by clicking it to add the button-level permissions (cf. Figure 3) for users. In principle, such links between UI widgets and permissions triggered can be automatically discovered by static and dynamic analysis of the apps. Then, such settings for different apps can be imported to UIPDroid to extend the scope of UI widgets that it can manage.

4 Empirical Evaluation

4.1 Experimental Setup

We experimented with 19 apps that use Contact and Location permissions in our preliminary evaluation. These apps are popular in various categories such as Map, Instant Messaging, Social Networking, Food, Shopping, Transportation. All applications are the latest version on Google Play as of July-23-2021. We ran the apps manually on a Google Pixel 4A phone with Android 10 to

trigger their Contact or Location uses and enable UIPDroid to control.

4.2 Experimental Results

The experimental results are summarized in Table 1. Some applications detect whether current mobile phone has installed Xposed or other similar frameworks, and detect whether their process are hooked. If so, they refuse to be executed for security reasons. Due to this reason, 9 out of 19 apps cannot be run in the sandbox and thus cannot be managed by our tool (cf. "VX loaded" column). For other apps, UIPDroid successfully enabled Allow/Deny/Fake permission control except for Telegram and Instagram Lite as they use some mobile development techniques, such as React Native, that are not hooked by VX. The execution performance impact is negligible from users' perspective because all codes are still executed on the native Android operating system and VirtualXposed only intercepts some UI-triggered method calls when inner applications interact with the system. For some large applications, the installation process may take longer time than native system because VirtualXposed disables JIT.

Table 1: Experimental Results. "VX loaded" indicates if an app can be run within VirtualXposed. "Allow", "Deny", "Fake" indicate if each of the permission settings can pass our manual test cases or not; "N/A" is for apps that cannot be run due to VX.

App Name	Permission	VX loaded	Allow	Deny	Fake
Red	Contact	Y	Y	Y	Y
WeChat	Contact	Y	Y	Y	Y
Telegram	Contact	Y	Y	N	Y
Instagram Lite	Contact	Y	Y	N	N
Facebook Lite	Contact	N	N/A	N/A	N/A
Lazada	Contact	N	N/A	N/A	N/A
Shopee	Contact	N	N/A	N/A	N/A
WhatsApp	Contact	N	N/A	N/A	N/A
FairPrice	Location	Y	Y	Y	Y
Moovit	Location	Y	Y	Y	Y
OneMap	Location	Y	Y	Y	Y
PizzaHut	Location	Y	Y	Y	Y
Singapore Map	Location	Y	Y	Y	Y
Waze	Location	Y	Y	Y	Y
Deliveroo	Location	N	N/A	N/A	N/A
FoodPanda	Location	N	N/A	N/A	N/A
McDonald	Location	N	N/A	N/A	N/A
myEnv	Location	N	N/A	N/A	N/A
MyTransport	Location	N	N/A	N/A	N/A

4.3 Discussions, Related Work & Future Work

Recently, Apple introduces App Privacy Report to let users know which and when permissions have been accessed by which apps [10]. Google also introduces a new privacy dashboard to help increase permissions transparency [11]. However, in both of IOS 15 and Android 12, users can only see the usage reports but cannot have fine-grained controls. Other manufacturers like Xiaomi and Huawei provide ROM-based solutions, using customized permission managers [12, 13]. However, they do not have widget-level control either. Root-based solutions are available, but are not convenient for end users to root their phones [14, 15, 16]. In comparison, UIPDroid has the advantages that it is root-free, provides more fine-grained widget-level control for better user awareness. On the other hand, its implementation has the following limitations.

Limited Permission Hooks. Many other permissions for various sensitive data, such as body sensors, accounts, call logs, etc. need fine-grained control too. More types of UI widgets can interact with users and stipulate permission uses. UIPDroid can hook more such places automatically in the future, although it may need more customization efforts for faking different types of data.

VirtualXposed Capabilities. VirtualXposed does not need root privileges, but thus cannot provide system-level operations when sandboxing apps, which may be the main reason for load failures in Table 1. Better app virtualization techniques are interesting future work for broader application of UIPDroid.

Permission Decision Databases. For a hooked point, UIPDroid allows permission decisions according to either users' choices or imported settings. This opens the possibility for UIPDroid to utilize crowdsourced shared permission settings [8] to reduce users' decision burdens and improve the privacy protection ecosystem.

5 Conclusion

We design and implement a fine-grained widget-level permission controller for managing permissions used in Android apps. It is built on the VirtualXposed framework without root privileges, making it easy for any end user with basic Android app knowledge to use our tool. Its current implementation monitors apps' every use of Contact and Location permissions and links the uses to UI widgets with respect to basic privacy-by-design and UI principles; then, it informs users of the permission uses and denies or fakes the data if chosen to. Furthermore, it provides interfaces for users to manage permission settings and review permission usage logs to enhance their awareness of the private data uses by apps. Finally, we evaluate the effectiveness of our tool based on a set of popular apps and highlight possibilities for future work.

6 Acknowledgment

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] M. V. Kleek, I. Liccardi, R. Binns, J. Zhao, D. J. Weitzner and N. Shadbolt, "Better the Devil You Know: Exposing the Data Sharing Practices of Smartphone Apps," in *CHI Conference on Human Factors in Computing Systems*, 2017.

[2] Google, "Request app permissions," [Online]. Available: https://developer.android.com/training/permissions/requesting.

[3] A. F. Westin, "Social and Political Dimensions of Privacy," *Journal of Social Issues,* vol. 59, no. 2, pp. 431-453, 2003.

[4] A. Cavoukian, "Privacy by design," *Identity in the Information Society,* vol. 3, no. 2, pp. 247-251, 2010.

[5] B. Shneiderman, C. Plaisant, M. S. Cohen, S. Jacobs, N. Elmqvist and N. Diakopoulos, Designing the User Interface: Strategies for Effective Human-Computer Interaction, Pearson, 2016.

[6] J. I. Hong and J. A. Landay, "An architecture for privacy-sensitive ubiquitous computing," in *2nd international conference on Mobile systems, applications, and services (MobiSys)*, 2004.

[7] J. Colnago, Y. Feng, T. Palanivel, S. Pearman, M. Ung, A. Acquisti, L. F. Cranor and N. Sadeh, "Informing the design of a personalized privacy assistant for the internet of things," in *CHI Conference on Human Factors in Computing Systems*, 2020.

[8] Y. Agarwal and M. Hall, "ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing," in *11th annual international conference on Mobile systems, application, and services (MobiSys)*, 2013.

[9] weishu, "VirtualXposed," [Online]. Available: https://github.com/android-hacker/VirtualXposed.

[10] Apple, "Apple advances its privacy leadership with iOS 15," [Online]. Available: https://www.apple.com/sg/newsroom/2021/06/apple-advances-its-privacy-leadership-with-ios-15-ipados-15-macos-monterey-and-watchos-8/.

[11] S. N-Marandi, "What's new in Android Privacy," 18 May 2021. [Online]. Available: https://android-developers.googleblog.com/2021/05/android-security-and-privacy-recap.html.

[12] Xiaomi, "Privacy at Xiaomi," [Online]. Available: https://privacy.miui.com/en/#/.

[13] Huawei, "Permission Management," [Online]. Available: https://consumer.huawei.com/en/support/content/en-us06515308/.

[14] S. Chitkara, N. Gothoskar, S. Harish, J. I. Hong and Y. Agarwal, "Does this App Really Need My Location? Context-Aware Privacy Management for Smartphones," *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies,* vol. 1, no. 3, pp. 1-22, 2017.

[15] Z. Alkindi, M. Sarrab and N. Alzidi, "CUPA: A Configurable User Privacy Approach For Android Mobile Application," in *7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2020.

[16] M. Bokhorst, "XPrivacyLua," [Online]. Available: https://github.com/M66B/XPrivacyLua.