Application Transiency: Towards a Fair Trade of Personal Information for Application Services

Raquel Alvarez(✉), Jake Levenson, Ryan Sheatsley, and Patrick McDaniel

Pennsylvania State University, State College, PA 16802, USA
{rva5120,jml6407,rms5643}@psu.edu, mcdaniel@cse.psu.edu,
http://siis.cse.psu.edu/

Abstract. Smartphone users are offered a plethora of applications providing services, such as games and entertainment. In 2018, 94% of applications on Google Play were advertised as "free". However, many of these applications obtain undefined amounts of personal information from unaware users. In this paper, we introduce *transiency*: a privacy-enhancing feature that prevents applications from running unless explicitly opened by the user. Transient applications can only collect sensitive user information while they are being used, and remain disabled otherwise. We show that a transient app would not be able to detect a sensitive user activity, such as a daily commute to work, unless it was used during the activity. We define characteristics of transient applications and find that, of the top 100 free apps on Google Play, 88 could be made transient. By allowing the user to decide when to allow an app to collect their data, we move towards a fair trade of personal information for application services.

Keywords: Mobile privacy · Android

1 Introduction

In 2018, over 2.6 million apps were available on the Google Play market, of which 94% were advertised as "free" [42]. Users can request a ride from apps like Uber or Lyft, share pictures on Facebook, or send money to a friend through Venmo. While these applications are advertised as free, they present a hidden cost: privacy. The effects of smartphones on user privacy have been widely studied since their commercialization in 2007 [5,7,8,12,28–30,32–37]. Recently, the private preferences and habits of millions of Facebook users were misused for political purposes [39]. Many studies also show that users *do* care about their privacy [14,15,20]. A study by Oates et al. found that users have mental models of what privacy means to them [23]. However, platforms can fail to provide them with intuitive options to control application behaviors [14,16,22]. Many users think of

Supported by NSF.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2019
Published by Springer Nature Switzerland AG 2019. All Rights Reserved
S. Chen et al. (Eds.): SecureComm 2019, LNICST 305, pp. 47–66, 2019.
https://doi.org/10.1007/978-3-030-37231-6_3

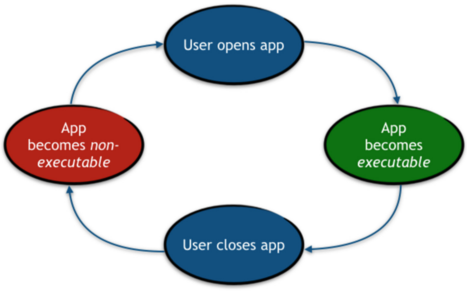


Fig. 1. Application transiency gives users control over when an application has access to sensitive information. A more fair trade is achieved since the user gets to decide whether their personal information is worth the privacy cost for the application service. We propose a new model in which the intuition of applications no longer running after being closed is realized.

closing physical barriers, such as doors and curtains, as an analogy to enabling privacy. By allowing all applications to constantly run in the background even after they are closed, smartphone application models don't adhere to common user privacy expectations.

Android controls user privacy through permissions that protect sensitive device resources. While permissions prevent applications from accessing arbitrary resources, studies have shown that permission models do not match user expectations on when and why sensitive data is being accessed [4,25]. Here, the authors showed that the context in which a resource is accessed matters, and that users prefer to deny access to resources that do not contribute to the functionality of the app. This led to recent work on helping users make educated decisions when granting permissions [10,11,13,24]. Recently, with Android 9.0, apps are no longer allowed background access to the camera and microphone. However, other sensitive resources (e.g., text messages) are still available to applications at any time, given that they were granted the permission once. Some studies have addressed this by sandboxing applications [8,9]. For example, Narain et al. fed crafted fake location data to protect the real location of users. In this paper, we show that smartphones can be designed to provide a fair trade of personal information in exchange for application services. We introduce *transiency*: a privacy-enhancing feature that prevents applications from running unless explicitly opened by the user. Transiency ensures that applications can only collect sensitive information when expected, as shown in Fig. 1. Therefore, users can now decide if application services are worth revealing their personal information for. We make the following contributions:

- We define transiency and efficiently integrate it into Android. This enforces a fair trade of personal information for application services.
- We define criteria for transient applications, and find that 88 of the top 100 free apps on Google Play (total of 105 ranked apps) should be treated as transient.

– We provide a case study to show the impact of treating applications as transient in terms of data collection. We find that we can prevent apps from detecting activity patterns by treating them as transient.

2 Background

In this section, we define technical details of Android relevant to the implementation of transiency.

2.1 Android OS Overview

The Android operating system is built on top of the Linux kernel, which is used to interact with the hardware functionality of smartphone devices, such as cameras and microphones. The main executable programs on Android devices are *applications*, which are developed in Java and built using the Application Framework. For more information about the architecture details of applications, refer to [1,44,46–48]. Below we describe in more detail some of the relevant architecture components to enforce application transiency.

Application Execution Flow. Applications exist in the system as Android Packages (APKs). APKs contain the necessary information to be executed by the Android Runtime (ART) [46]. In order for an application to be executed correctly, Android expects APKs to be available and readable by the system.

Background Processes. Android applications are designed to off-load large tasks that may potentially block the user interface (UI) to be run in the background. Logging the current location of the user to a server is an example of a task that would be executed in the background. In 2018, Votipka et al. performed an extensive user study in which they gained insights about what users think of background processes [14]. The results showed that users tend to understand the need for background processes while they interact with the UI. However, users were less comfortable with background processes not tied to foreground activities.

Permissions. Android uses permissions to protect applications from accessing device resources. Permissions are divided into two categories: *normal* and *dangerous*. Normal permissions are granted by default, but dangerous permissions must be granted by the user. Dangerous permissions protect access to sensitive resources, such as camera, microphone, text messages, call logs, calendar, etc.

Opening and Closing Applications. In order to open an application, Android defines a specific type of application that displays the available options to the user: *launcher*. The launcher app can be opened by pressing the "Home" button on the mobile device. Once open, a user can close an application by removing it from the "Recents Screen", as suggested by Google Support [51] (see Fig. 2). However, it is worth noting that closing an application does not prevent the application from running in the background and exercising the previously authorized access to sensitive resources.

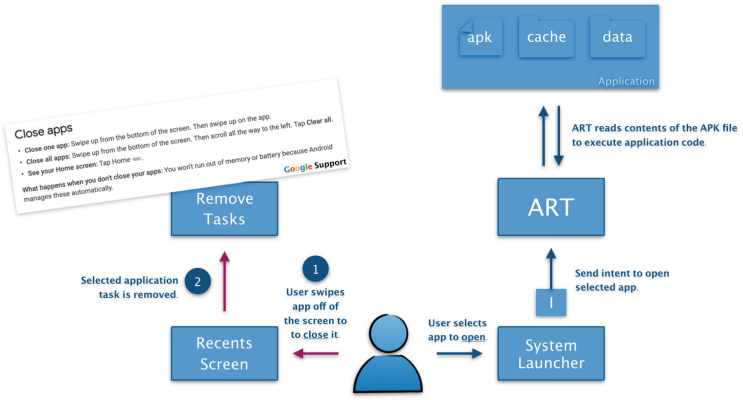


Fig. 2. Users can open apps by pressing the "Home" button and invoking the launcher app. To close applications, the user can swipe it off the screen. While closing an application is the terminology used by Google, the behavior is not equivalent to the application no longer running. Android allows applications to separate their user interfaces into *tasks*, which the user can terminate. However, while tasks can be terminated by the user, the application is still allowed to run on the background.

2.2 Sensitive Resource Access Control

Android protects access to valuable sensitive resources using *access control* policies [2,3]. The goal of these policies is to prevent applications from compromising user privacy. These policies ensure that applications only access sensitive resources allowed by the user. On the context of access control, applications (*subjects*) may or may not be authorized to access (*action*) sensitive system resources (*objects*). Android handles *authentication* and *authorization* as follows:

Authentication. At installation time, applications are given User IDs (UIDs) and are treated as users of the system. UIDs give applications an identity, which is used by Android to identify the application's set of authorized access.

Authorization. Android provides an interface for applications to prompt the user for access to sensitive resources. As mentioned previously, *dangerous permissions* must be granted by the user for an application to be able to access a resource such as GPS data. Android keeps track of the authorized access to sensitive resources for each application. Once the user grants permission, the application is free to exercise this right both when running in the foreground and background. However, as of Android 9.0, applications cannot access the microphone and camera resources while running in the background [54].

While permissions serve as a policy that allows users to explicitly grant access to sensitive resources, research has shown that this model places an unrealistic expectation on users [4,25]. Applications don't always provide clear privacy policies justifying the requested access [28], which presents an *information asymmetry* problem. By granting access to a sensitive resource, the user is not necessarily

giving their consent since they were not fully disclosed the information necessary to make that decision. This inequality inspired the solution presented in this work.

3 Application Transiency Design Goals

In order to implement transiency, we define security, privacy, and usability requirements.

- **Security Goals.** We *trust* that the OS faithfully implements and enforces application transiency. We consider any other party that may tamper with the transient state of an application to be a *threat*.
- **Privacy Goals.** We enforce that applications are not capable of running unless they were explicitly opened by the user. This guarantees that applications cannot access personal information unless they are being used.
- **Usability Goals.** Enforcing transiency should be intuitive and seamless to the user. This guarantees user privacy when expected, without affecting user experience.

4 Application Transiency Implementation

We expand the sensitive resource access control policy of permissions to provide an intuitive interface where users can control their privacy. We propose a new authorization protocol, in which users grant/revoke applications the right to execute by opening/closing them. We address the goals described in the previous section in our implementation of application transiency. Figure 3 provides an implementation overview.

4.1 Enforcing Transiency

To enforce transiency, the user must be the only one who can execute an application. Otherwise, a closed application may attempt to execute code that accesses sensitive user data. We solve this problem by leveraging access control used by the OS to protect files on Android. Specifically, we revoke read access to APKs if they are not explicitly open (shown in Fig. 3, step 13). Once revoked, if the ART unit attempts to read the APK for any other reason aside from user intent, it will fail. Conversely, when a user opens an application, our implementation will restore the read permission before the ART attempts to read the APK file, as shown in Fig. 3, steps 1–6.

Transiency is implemented at the OS level, as a system library only accessible to applications/processes with a system UID (which is 0). Transiency can be realized in the form of a "private-mode" system launcher. The system launcher, when running on private-mode, uses the "swipping off the Recents screen" event as the cue to stop the application from running. A user could select "private-mode" from the Settings app (which is where most users expect to find interfaces

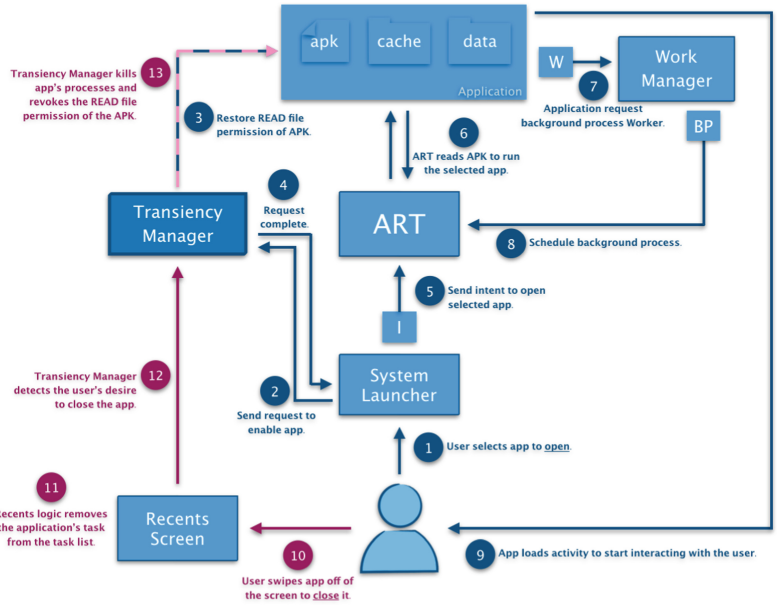


Fig. 3. An overview of the implementation of application transiency. Steps in blue, 1–9, correspond to the event of a user turning privacy off by opening the application. Steps in burgundy, 10–13, correspond to the event of a user turning privacy on by closing the application. **I** corresponds to the intent sent by the launcher to start an application, **W** corresponds to the Worker object specified by the application to perform a task in the background, and **BP** corresponds to a background process being scheduled by the Work Manager API for the requesting application [53]. In this diagram, we depict an application that, when opened by a user, requests a background process to perform some task and then loads the activity to interact with the user. Once the user is finished interacting with the application, they can close it by swiping it off the screen. Opening the application causes the Transiency Manager to restore the READ permission of the APK file to the system. Closing the application causes the Transiency Manager to revoke the READ permission to prevent execution. (Color figure online)

to change/enable/disable features, as pointed out by Lei et al.). In addition, malicious 3rd party applications cannot leverage the library functionality to perform attacks such as DoS (since each app is given a UID depending on the key used to sign the application, so only apps signed with manufacturer/system keys are given the UID 0).

This solution is efficient because revoking and restoring the read permission from the APK file causes a negligible overhead. In addition, enforcing transiency does not require a re-design of the system architecture to support it.

4.2 Making Transiency Intuitive

To meet the usability requirements defined in the previous section, we designed our implementation of transiency to meet user expectations of privacy. As pointed out by Oates et al. in 2018, for most users, regardless of demographics and technical experience, barriers are seen as the most common conceptual metaphor of privacy. Many related privacy with closing doors, locks, or curtains. This study served as an inspiration on our design to find a mapping from the mental models of what privacy means to users, to features in Android that can make this a reality.

Closing Doors. One of the common occurrences for most users was turning on privacy by closing a door or a curtain. Inspired by this, we equate closing an application to enabling privacy. Closing an application on Android refers to the action of selecting the "Recents" UI and swiping the application off of the screen [51,52]. However, closing an application does not mean it is no longer running. Applications have the option to schedule background processes that will cause them to run again, unknown to the user. Therefore, the current application model does not meet the potential user expectations of privacy. For this reason, we chose the event of closing an application as the defining moment of turning on privacy. By closing the application, the user is guaranteed that the application will not run. This prevents applications from reading sensitive information when the user is not aware.

Opening Doors. Analogous to closing doors, we use opening doors to disable privacy. Android devices have a launcher application that is invoked when the "Home" button is pressed. This launcher application displays all installed and launchable applications on the system. A user can then touch one of the application icons to open it. Once an application is open, it can access sensitive information.

By enforcing transiency through opening and closing applications, we give the user an intuitive interface to better manage their privacy. This guides users towards a fair trade of sensitive information for application services.

4.3 Applying Transiency to Popular Applications

In this section, we analyze the behavior of popular applications when they are treated as transient and non-transient on the current Android architecture. We generalize these behaviors and define three main application functionalities that would require non-transiency.

Instagram is an application that displays user-generated content and allows users to communicate through comments and private messages. When treated as transient, Instagram no longer sends real-time notifications to the user. However, transiency does not prevent Instagram from displaying user-generated content. We therefore recommend that Instagram be treated as transient.

Subway Surfers is a game in which users can connect with other friends to form a network. When treated as transient, Subway Surfers no longer sends real-time notifications, but the user can still play the game as expected. Since transiency does not prevent the user from playing the game, we recommend that Subway Surfers be treated as transient.

Spotify offers users a music streaming platform. Treating Spotify as transient does not affect core functionality of the app. Therefore, we can recommend treating Spotify as transient.

Uber is a ride-sharing service app connecting riders with drivers. Treating Uber as transient does not affect the functionality of the app, since users can keep the application open for the duration of the ride. Therefore, we recommend treating Uber as transient.

While the user actively engages with the applications above, there exist other types of applications that passively interact with the user. We describe examples of those below:

Facebook Messenger is an app that enables users to communicate via phone calls, video calls and messages. When treated as transient, Facebook Messenger only notifies users in real-time when the app is open. However, keeping Facebook Messenger open at all times is not ideal for performance. Therefore, we recommend treating this application as non-transient.

Gmail provides an interface to read and send emails to other users. Gmail also loses the ability to notify the user in real-time when treated as transient. Therefore, we recommend treating Gmail as non-transient. Likewise, instant messaging applications have the same non-transiency requirements.

Step Counter gives a count of the approximate number of steps taken by the user. The app cannot use the sensors on the phone to constantly measure the amount of steps taken by the user in real-time unless the application is open. Therefore, we recommend treating the app as non-transient.

Clean Master offers users tools to clean files, a notification with a button to remove all other notifications, and other features such as taking a picture with the front-camera when the login passcode to unlock the screen is entered wrong. When treated as transient, this application cannot perform some of its core functionality, like taking pictures of the user that entered the wrong passcode, if the app is closed. We therefore recommend treating Clean Master as non-transient.

After analyzing these applications, we can generalize their behaviors to broadly describe non-transiency classification guidelines.

Recommended Classification Guidelines. Based on our analysis of the behavior of popular applications when they are treated as transient, we define three main functionalities to generalize when applications should not be treated as transient:

A. ***The application provides real-time communication such as calling, messaging or receiving time-sensitive information.*** This functionality enables real-time communication and exchange of information, such as Gmail. A user expects text messages to arrive at a reasonable time after the sender sent them, which requires the application to have real-time access to the network to retrieve messages intended for the user.

B. ***The application requires real-time access to sensors to collect information and report it to the user.*** This includes applications with functionality that depends on real-time information collected by device sensors, such as the Step Counter app. An application that reports the number of steps taken by the user depends on real-time information measured by device sensors. The user expects to see an accurate number of steps when they open the app, therefore the application cannot be transient.

C. ***The application depends on real-time system state or other applications to provide its functionality.*** Some applications may need to be aware of system events to function. For example, the Clean Master app.

While it is possible to analyze applications manually to determine their classification, it can be a time consuming task given that Google Play had a total of 2.6 million available applications in December 2018 [42]. We therefore explore methods to automate the process in the next section.

Automating Classification. To simplify the task of classifying apps, we explored using Google Play categories to recommend what applications should be transient and non-transient. To do this, we looked for categories on Google Play that would match the description of each of our A-C categories based on functionality. Below are our categories mapped to Google Play categories:

A - Communication: Applications in the Communication category depend on real-time delivery of content to the user. Some of these applications include WhatsApp and Facebook Messenger. Therefore, we mapped Communication to A.

B - Health & Fitness: Some applications in the Health & Fitness category depend on reading sensors that measure physical activities of a user, which makes this a non-transient category. An examples of these apps is Step Counter.

C - Tools: Applications in this category depend on real-time sensor and system information to report to the user. Analyzing current popular applications in the Tools category, we found that most applications were related to changing system settings, such as WiFi.

We selected three categories where each covered one of the three cases in which transiency would affect the primary functionality of an application. We evaluate the accuracy of this automated classification method by comparing it to the manual classification of the top 100 free apps on Google Play, which is described in Sect. 5.2.

4.4 Android Implementation: Transiency Launcher

To test our design, we develop a system launcher that implements application transiency.[1] We design the launcher to test the performance and usability of treating applications as transient on the current Android architecture. Figure 4 shows an overview of the architecture of this launcher.

UI. Our transiency launcher has a simple main activity that displays the list of installed and launchable applications on the system. Each entry on the list corresponds to an application, and the user can select an entry from the list to open the application selected. The launcher is invoked when the user presses the "Home" button.

Backend. Our launcher has a database that keeps track of the installed applications and their transient/non-transient classification. For testing purposes, we manually saved a list of names of applications treated as non-transient, which was decided following the criteria described in Sect. 4.3. When the user wants to open an application treated as transient, the launcher executes `chmod` through a shell to change the permissions of the application's APK to be readable by the system. The launcher then sends an intent to start the main activity of the selected application. If the user chooses to open a non-transient application,

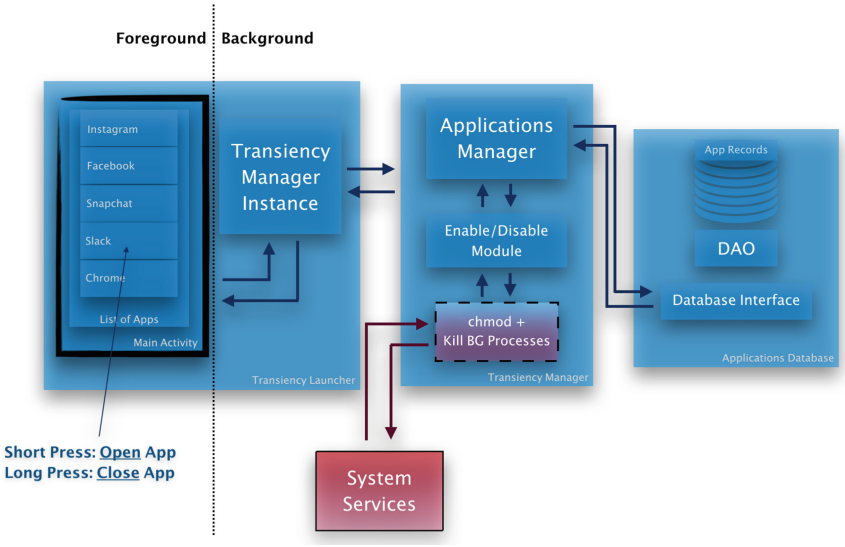


Fig. 4. Overview of the architecture of the transiency launcher. This launcher uses `chmod` to revoke/restore read permissions of the APK file to control when applications can collect user information. The Transiency Manager API can be used by our launcher to enable/disable the applications.

[1] Source code: https://github.com/rva5120/TransientLauncher.

the launcher sends the intent directly, as the APK permissions are not modified for non-transient applications. When the user long presses an entry to close an application, the launcher will execute `chmod` through a shell to change the permissions of the application's APK to no longer be readable by the system.

5 Evaluation

In this section, we will evaluate the classification and characterization of applications currently available on the market.

5.1 Characterization of Market Applications

We now evaluate the concept of application transiency by applying it to the top 100 free apps on Google Play. We analyzed these applications manually, and classified them according to the criteria defined in Sect. 4.3.

Classification Statistics. After manual classification, the top 100 free apps on Google Play (total of 105 apps) consisted of 88 transient and 17 non-transient, see Fig. 5.

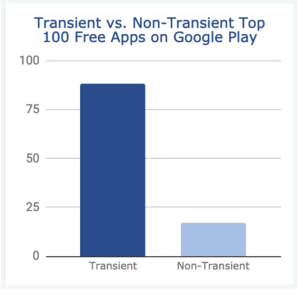


Fig. 5. Manual classification of transient/non-transient applications of the top 100 free apps on Google Play (total of 105 ranked apps), as of February 11th of 2019.

Characterization of Transient Applications. We find that over 50% of transient applications belong to the Games Google Play category, as shown in Fig. 6 (left). Our criteria led to all games being classified as transient. All games request the INTERNET permission, over 17% request the ACCESS_FINE_ LOCATION permission, and over 12% request the RECORD_AUDIO permission, see Fig. 7 (left). As we will see in Sect. 6, applications that request the INTERNET permission can also approximate the location of the device, therefore, most games can approximate location as well. By treating gaming applications as transient, we prevent them from recording audio in the background (if the device is running Android 8.0 and below) and collecting location data constantly when the user is not expecting it.

Transient applications in other categories also requested CAMERA and READ or WRITE_EXTERNAL_STORAGE permissions, which allows applications to take pictures in the background (for devices running Android 8.0 and below) and accessing pictures and other files. It is also worth noting that applications like Netflix and Facebook request the Activity Recognition permission. When these apps are not treated as transient, they can collect information about the user's physical activity. In addition, Facebook also requests the ACCESS_FINE_ LOCATION permission, which would allow the app to map certain routines such as commonly walked paths or commonly driven roads. However, if we treat apps like Facebook as transient, we can prevent the association of physical activity data with its corresponding user profile. Linking this kind of sensitive data together can pose physical dangers to users, as shown by [27].

Characterization of Non-transient Applications. During our analysis, we found that only 17 apps needed to be treated as non-transient. Out of the total 17, 5 belong to the Communication category (as shown in Fig. 6 (right)). This was not surprising, since applications in the Communication category are expected to need real-time access to network resources to allow users to communicate in real-time.

Non-transient applications also requested the INTERNET permission, and over 88% of them request READ and WRITE_EXTERNAL_STORAGE (see Fig. 7 (right)). We find that most applications also request ACCESS_FINE_ LOCATION and CAMERA. While these are expected based on the functionality provided by the applications, we also find that both Antivirus Free 2019 and

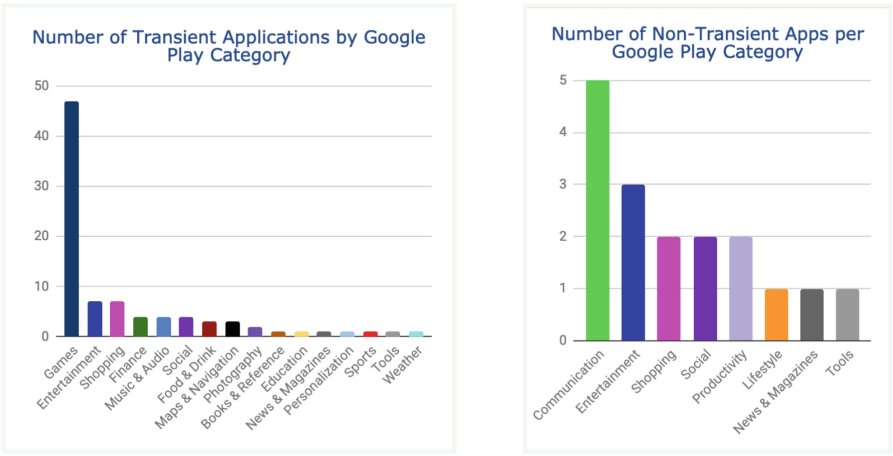


Fig. 6. Manual classification of applications grouped by Google Play category. Left: number of transient applications in each category, with Games being the category with the most transient apps with 47 (53.4% of all transient apps). Right: number of non-transient applications in each category, with Communications being the category with most non-transient apps with 5 (29.4% of all non-transient apps).

Super Cleaner requested access to the Activity Recognition API. This seems unusual, as a user may not expect for an antivirus application to need their physical activity to perform its functionality.

5.2 Classification Through Google Play Categories

Classifying apps using Google Play categories yields an accuracy of 88.57%, where 93 out of 105 apps were classified correctly compared to our manual classification. Our automatic classification method is less accurate when classifying applications that were manually labeled as non-transient. As expected, some applications place themselves under other categories. For example, TextNow, which enables communication among users, is categorized as Social. This may be caused by the fact that very popular applications like Facebook are in the Social category, so TextNow may be seen by more users and be more likely to get downloaded if it is in the Social category. However, since we label Social to be a transient category, communication apps that choose to be in the Social category will get misclassified and lose their non-transient privileges. Other communication apps, like Tinder, are also misclassified. Tinder is placed in the Dating category, which is expected since it enables communication between users with the purpose of dating.

Other examples of misclassified non-transient apps include IN Launcher, Bitmoji, Super Speed Cleaner and Antivirus Free 2019. Based on functionality, we would consider these apps to be Tools, which would grant them the privilege of

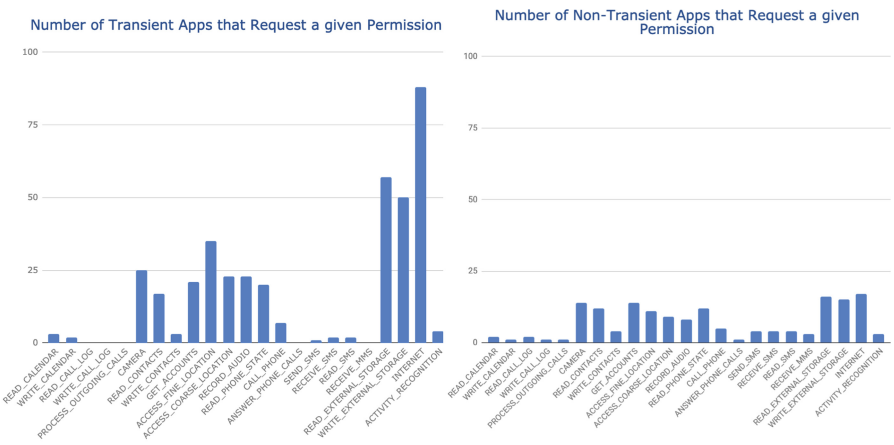


Fig. 7. Number of applications that request dangerous permissions, the INTERNET permission, and the Activity Recognition API permission of the top 100 free apps on Google Play. On the left, we see that all transient applications request the INTERNET permission, and over 25% of them request GPS location information. On the right, we see that all non-transient applications are likely to request INTERNET and location permissions as well.

being non-transient. However, they categorize themselves as Entertainment (for IN Launcher and Bitmoji) and Productivity (for Speed Cleaner and Antivirus Free 2019).

5.3 Implementation Performance

There is no visual delay when opening an application through a regular launcher vs our transiency launcher, which incurres an overhead of 0.02 ms. This performance overhead, which is added by the extra instructions executed to revoke/restore read permissions of the APK file, is negligible.

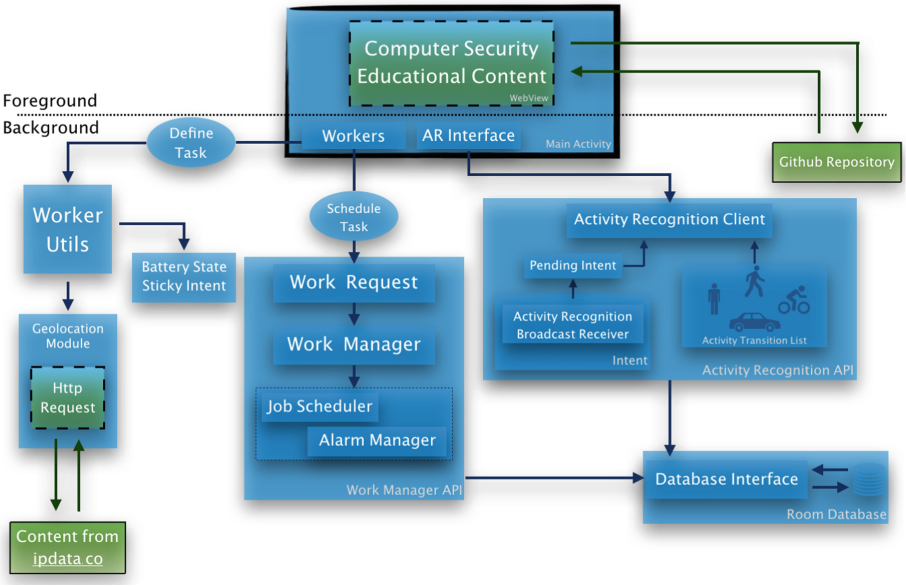


Fig. 8. Overview of the architecture of the Metis app. This app provides a user interface in which users can read weekly tutorials to learn about computer security. However, in the background, Metis is performing some data collection to learn more about your daily patterns, such as where you work, and when and how you get there. Metis only needs the INTERNET and ACTIVITY_RECOGNITION permissions, which are granted by default, to achieve its data collection goals. This makes Metis a good candidate to show that transiency is necessary to prevent apps from collecting data when users are not expecting it.

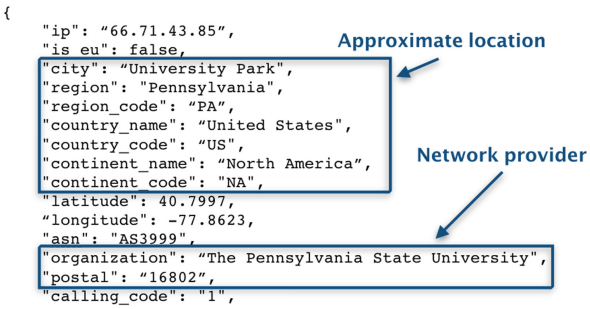
6 Case Study: Measuring Impact of Transiency on Data Collection

To show the kind of impact that treating applications as transient would have on sensitive data collection, we develop the Metis app. Figure 8 shows the overall architecture of Metis.[2]

6.1 Metis, the Knowledge Sharing App

Metis is an application that gives users a weekly article about computer security topics. The Metis UI is a simple webview object that displays the contents of a webpage hosted on a Github repository. The app displays a blog-style article with content to read. While looking like an innocent educational app in the foreground, Metis performs sensitive data collection in the background. The data collection strategy is inspired by the recent work of Chatterjee et al., in which they studied market applications that contribute to intimate partner violence. We focus on finding device resources that would reveal sensitive information without the user knowing or expecting it. We find that the INTERNET and the ACTIVITY_RECOGNITION permissions are good candidates for our purpose.

INTERNET. Metis, since it must request a webpage from Github to display its contents, needs to request the INTERNET permission. We find that we could *approximate the user's location by connecting to a geo-locating website*, such as Ipdata.co [40]. In addition to an approximate location, Ipdata.co reveals other potentially sensitive information: whether the user is on WiFi/cellular network, and the organization providing the IP. The figure below shows the information given by querying the API of Ipdata.co:



ACTIVITY_RECOGNITION. We use the approximate location information and combine it with the information available from the Activity Recognition API [49]. By requesting the normal ACTIVITY_RECOGNITION permission, we are able to setup Metis to *receive physical activity changes of the user*. For example, if the user starts walking or driving, Metis will receive a broadcast. This allows us to *recognize patterns of driving to and from work*, for example.

[2] Source code: https://github.com/rva5120/Metis_v2.

6.2 Data Collection: Transient vs. Non-transient

We compare the amount of data collected by Metis when it is treated as transient vs. non-transient.

Experiment. We run Metis on a rooted device with the transiency launcher described in Sect. 4.4 installed. We install two versions of Metis: Metis-T (treated as a transient application), and Metis-NT (treated as a non-transient application). We run this experiment for 1 day. Starting around 9 AM, we opened both applications. A few seconds later, we close both applications. Then, around 10:20 AM, we open both applications and leave them open. Below are the results captured until 12:30 PM by both apps.

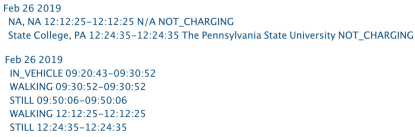


Fig. 9. Results of running Metis-NT.

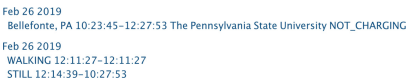


Fig. 10. Results of running Metis-T.

Metis-NT Results. The non-transient version of Metis is able to capture that we possibly commuted to work in the morning around 9:20 AM, which took around 10 min (see Fig. 9). It was detected that we walked for another 20 min, although during the experiment we walked for about 8 min. However, Metis-NT may not have received the "transition to STILL broadcast" until 9:50, so it records that we walked for longer than we actually did. Metis-NT also ran a background process around 12:24 PM where it recorded the geo-location received by Ipdata.co. In this case, Metis-NT detected that we were on a Pennsylvania State University network in University Park, PA which was the correct City, Region and University during our experiment.

Metis-T Results. Metis-T, on the other hand, is unable to detect the drive to work in the morning (see Fig. 10). It was only able to detect the walking activity around 12 because we left the app open after 10:20 AM.

Transiency has an impact over when Metis is able to collect sensitive information. By preventing Metis from running after being closed, we are able to preserve the privacy of an event that was not intended for Metis to detect. Also, transiency can have a major impact on protecting users by treating applications found by [27] as transient.

7 Discussion

By implementing transiency, we learn that most applications do not need the privilege of running constantly to provide their applications services. To prevent their execution while closed, we also explore the idea of installing and

uninstalling applications. Installing an application every time the user opens it safely meets the privacy requirements, since the application is not able to execute code. However, it added too much overhead to the overall user experience. Also, uninstalling apps required users to interact with the application as if it was the first time they opened it. For example, a user would have to login every time they opened the app, which may cause an inconvenience. Therefore, we discarded this idea.

One of the main limitations of transiency for the current Android implementation is the inability to support notifications. In the future, it would be worth exploring what modifications can be made to the notifications API to be able to support transient notifications. Transient notifications would give applications the ability to ask the user for permission to run for a clearly specified purpose, implemented by a fine-grained API that does not allow the application to violate users' expectation of privacy.

We also observed that some applications already display a transient behavior. For example, Spotify will only play music while the application is open. If the user closes the application, music will stop playing. PrivateRide [6] is another example of an application that was re-designed to respect user privacy. These apps show that it is possible to design applications that provide useful functionalities without abusing privacy.

Lastly, we find that another benefit of transiency was addressing that users forget or find it inconvenient to delete unused applications [38]. If applications are treated as transient by default, users can rest assured that installing an application is not equivalent to giving it the privilege of running unconstrained.

8 Future Work

The classification of applications as either transient and non-transient provided was intended as a coarse approximation of the applications that could be treated as transient. In so doing, shed light in the likely impact of transiency on a real system. However, the system used by Google to categorize applications based on functionality is an imperfect medium to perform this analysis. As future work, we plan on incorporating the methods and findings of studies like AWare [7] and Turtle Guard [10] (which extensibly studied usability and contextual cues) to provide a more fine-grained classification methodology.

9 Conclusion

Throughout this paper, we explored the idea of enforcing transiency, which disables apps that were not explicitly opened by the user. Currently, Android applications can collect sensitive information even if they are not being used. Privacy is still an ongoing problem, which starts with the lack of control users have over the amount of information applications can collect. Transiency solves this issue intuitively, moving towards a fair trade of personal information in exchange for application services.

Acknowledgements. Thank you to Kim, Cookie, Bon Bon, and all the SIIS labers for the much needed support on my first paper journey. This material is based upon work supported by the National Science Foundation under Grant No. NS-1564105. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Elenkov, N.: Android Security Internals. No Starch Press, San Francisco (2015)
2. Stamp, M.: Information Security Principles and Practice. Wiley, Hoboken (2011)
3. Jaeger, T.: Operating System Security. Morgan & Claypool Publishers, San Rafael (2008)
4. Nissenbaum, H.: Privacy as Contextual Integrity. Washington Law Review (2004)
5. Enck, W., et al.: TaintDroid: an information-flow tracking system for realt ime privacy monitoring on smartphones. In: OSDI (2010)
6. Pham, A., et al.: PrivateRide: a privacy-enhanced ride-hailing service. In: Proceedings of the 17th Privacy Enhancing Technologies Symposium (2018)
7. Petracca, G., et al.: AWare: preventing abuse of privacy-sensitive sensors via operation bindings. In: Proceedings of the 26th USENIX Security Symposium. USENIX Security (2017)
8. Narain, S., Noubir, G.: Mitigating location privacy attacks on mobile devices using dynamic app sandboxing. In: Procededings of the 19th Privacy Enhancing Technologies Symposium (PETS) (2019)
9. Zhou, Y., Zhang, X., Jiang, X., Freeh, V.W.: Taming information-stealing smartphone applications (on Android). In: McCune, J.M., Balacheff, B., Perrig, A., Sadeghi, A.-R., Sasse, A., Beres, Y. (eds.) Trust 2011. LNCS, vol. 6740, pp. 93–107. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21599-5_7
10. Tsai, L., et al.: Turtle guard: helping Android users apply contextual privacy preferences. In: Proceedings of the 26th USENIX Security Symposium (2017)
11. Liu, B., et al.: Follow my recommendations: a personalized privacy assistant for mobile app permissions (2016)
12. Hornyack, P., Han, S., Jung, J., Schechter, S., Wetheral, D.: These aren't the droids you're looking for: retrofitting Android to protect data from imperious applications. In: CCS (2011)
13. Wijesekera, P., et al.: The feasibility of dynamically granted permissions: aligning mobile privacy with user preferences. In: NDSS (2017)
14. Votipka, D., Rabin, S.M., Micinski, K., Gilray, T., Mazurek, M.M., Foster, J.S.: User comfort with Android background resource accesses in different contexts. In: Proceedings of the 14th Symposium on Usable Privacy and Security (2018)
15. Egelman, S., Felt, A.P., Wagner, D.: Choice architecture and smartphone privacy: there's a price for that. In: Böhme, R. (ed.) The Economics of Information Security and Privacy, pp. 211–236. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39498-0_10
16. Felt, A.P., Egelman, S., Wagner, D.: I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In: 2nd Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (2012)
17. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: user attention, comprehension, and behavior. In: Proceedings of the 8th Symposium on Usable Privacy and Security (SOUPS) (2012)

18. Bonné, B., Peddinti, S.T., Bilogrevic, I., Taft, N.: Exploring decision making with Android's runtime permission dialogs using in-context surveys. In: Proceedings of the 13th Symposium on Usable Privacy and Security (SOUPS) (2017)
19. Pu, Y., Grossklags, J.: Valuating friends' privacy: does anonymity of sharing personal data matter? In: Proceedings of the 13th Symposium on Usable Privacy and Security (SOUPS) (2017)
20. Tsai, J., Egelman, S., Cranor, L., Acquisti, A.: The effect of online privacy information on purchasing behavior: an experimental study. In: 6th Workshop on the Economics of Information Security (2007)
21. Samat, S., Acquisti, A.: Format vs. content: the impact of risk and presentation on disclosure decisions. In: Proceedings of the 13th Symposium on Usable Privacy and Security (SOUPS) (2017)
22. Rao, A., Schaub, F., Sadeh, N., Acquisti, A., Kang, R.: Expecting the unexpected: understanding mismatched privacy expectations online. In: Proceedings of the 12th Symposium on Usable Privacy and Security (SOUPS) (2016)
23. Oates, M., et al.: Turtles, locks, and bathrooms: understanding mental models of privacy through illustration. In: Proceedings of the 18th Privacy Enhancing Technologies Symposium (PETS) (2018)
24. Ismail, Q., Ahmed, T., Caine, K., Kapadia, A., Reiter, M.: To permit or not to permit, that is the usability question: crowdsourcing mobile apps' privacy permission settings. In: Proceedings of the 18th Privacy Enhancing Technologies Symposium (PETS) (2017)
25. Wijesekera, P., Baokar, A., Hosseini, A., Egelman, S., Wagner, D., Beznosov, K.: Android permissions remystified: a field study on contextual integrity. In: Proceedings of the 24th USENIX Security Symposium (2015)
26. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions dymistified. In: CCS (2011)
27. Chatterjee, R., et al.: The spyware used in intimate partner violence. In: IEEE Symposium on Security and Privacy (2018)
28. Bowers, J., Reaves, B., Sherman, I.N., Traynor, P., Butler, K.: Regulators, mount up! analysis of privacy policies for mobile money services. In: Proceedings of the 13th Symposium on Usable Privacy and Security (SOUPS) (2017)
29. Das, A., Borisov, N., Chou, E.: Every move you make: exploring practical issues in smartphone motion sensor fingerprinting and countermeasures. In: Proceedings of the 18th Privacy Enhancing Technologies Symposium (PETS) (2018)
30. Reyes, I., et al.: Won't somebody think of the children? examining COPPA compliance at scale. In: Proceedings of the 18th Privacy Enhancing Technologies Symposium (2018)
31. Venkatadri, G., Lucherini, E., Sapiezynski, P., Mislove, A.: Investigating sources of PII used in Facebook's targeted advertising. In: Proceedings of the 19th Privacy Enhancing Technologies Symposium (2019)
32. Foppe, L., Martin, J., Mayberry, T., Rye, E.C., Brown, L.: Exploiting TLS client authentication for widespread user tracking (2018)
33. Bashir, M.A., Wilson, C.: Diffusion of user tracking data in the online advertising ecosystem. In: Proceedings of the 18th Privacy Enhancing Technologies Symposium (2018)
34. Lifshits, P., et al.: Power to peep-all: inference attacks by malicious batteries on mobile devices. In: Proceedings of the 18th Privacy Enhancing Technologies Symposium (2018)

35. Eskandari, M., Ahmad, M., Oliveira, A.S., Crispo, B.: Analyzing remote server locations for personal data transfers in mobile apps. In: Proceedings of the 17th Privacy Enhancing Technologies Symposium (2017)
36. Brookman, J., Rouge, P., Alva, A., Yeung, C.: Cross-device tracking: measurement and disclosures. In: Proceedings of the 17th Privacy Enhancing Technologies Symposium (2017)
37. Zhou, X., et al.: Identity, location. inferring your secrets from Android public resources. In: CCS, Disease and More (2013)
38. Park, H., Eun, J., Lee, J.: Why do smartphone users hesitate to delete unused apps? In: MobileHCI (2018)
39. Senate: Testimony of Mark Zuckerberg. https://www.judiciary.senate.gov/imo/media/doc/04-10-18%20Zuckerberg%20Testimony.pdf. Accessed Feb 2019
40. https://ipdata.co/ . Accessed Feb 2019
41. Statista: Distribution of free and paid Android apps (2019). https://www.statista.com/statistics/266211/distribution-of-free-and-paid-android-apps/
42. Statista: Number of Available Applications in the Google Play Store (2019). https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
43. Statista: Number of Paying Spotify Subscribers. https://www.statista.com/statistics/244995/number-of-paying-spotify-subscribers/
44. Google (2019). https://developer.android.com/
45. Google (2019). https://developer.android.com/guide/components/fundamentals
46. Google (2019). https://source.android.com/
47. Google: Codelabs (2019). https://codelabs.developers.google.com/
48. Google: Android Open Source Code (2019). https://source.android.com/
49. Google: Activity Recognition API (2019). https://developers.google.com/location-context/activity-recognition/
50. IPData.co (2019). https://ipdata.co/
51. Google: Google Answers. https://support.google.com/android/answer/9079646?hl=en. Accessed Feb 2019
52. Google: The Recents UI (2019). https://developer.android.com/guide/components/activities/recents
53. Google: Work Manager API (2019). https://developer.android.com/reference/androidx/work/WorkManager
54. Google: Android 9.0 Behavior Changes (2019). https://developer.android.com/about/versions/pie/android-9.0-changes-all