Does this App Really Need My Location? Context-Aware Privacy Management for Smartphones

SAKSHAM CHITKARA, Carnegie Mellon University, USA
NISHAD GOTHOSKAR, Carnegie Mellon University, USA
SUHAS HARISH, Carnegie Mellon University, USA
JASON I. HONG, Carnegie Mellon University, USA
YUVRAJ AGARWAL, Carnegie Mellon University, USA

The enormous popularity of smartphones, their rich sensing capabilities, and the data they have about their users have lead to millions of apps being developed and used. However, these capabilities have also led to numerous privacy concerns. Platform manufacturers, as well as researchers, have proposed numerous ways of mitigating these concerns, primarily by providing fine-grained visibility and privacy controls to the user on a per-app basis. In this paper, we show that this per-app permission approach is suboptimal for many apps, primarily because most data accesses occur due to a small set of popular third-party libraries which are common across multiple apps. To address this problem, we present the design and implementation of ProtectMyPrivacy (PmP) for Android, which can detect critical contextual information at runtime when privacy-sensitive data accesses occur. In particular, PmP infers the purpose of the data access, i.e. whether the data access is by a third-party library or by the app itself for its functionality. Based on crowdsourced data, we show that there are in fact a set of 30 libraries which are responsible for more than half of private data accesses. Controlling sensitive data accessed by these libraries can therefore be an effective mechanism for managing their privacy. We deployed our PmP app to 1,321 real users, showing that the number of privacy decisions that users have to make are significantly reduced. In addition, we show that our users are better protected against data leakage when using our new library-based blocking mechanism as compared to the traditional app-level permission mechanisms.

CCS Concepts: • **Security and privacy → Privacy protections**; **Usability in security and privacy**;

Additional Key Words and Phrases: Privacy, Permissions Model, Third Party Libraries, Purpose, Android

ACM Reference Format:
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 . *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 3, Article 42 (September 2017), 22 pages.
https://doi.org/10.1145/3132029

Author's address: S. Chitkara, N. Gothoskar, S. Harish, J. Hong, Y. Agarwal, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, P.A. 15213; emails:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2017 Association for Computing Machinery.
2474-9567/2017/9-ART42
https://doi.org/10.1145/3132029

1 INTRODUCTION

The phenomenal popularity of smartphones over the past decade has been driven by their significant computation, communication, and sensing capabilities, as well as the rich data they have about their users. As a result, platforms such as iOS and Android now have several millions of apps developed for them and billions of users [33]. This meteoric rise of smartphones and apps has created many new opportunities and new usage modalities, but have also led to their own set of challenges. For example, the rich sensing capabilities and numerous sources of data about the user have led to significant privacy concerns over the past few years [1, 6, 21, 32, 35–37]. iOS and Android have responded to these concerns by introducing several privacy-related enhancements over the years, including fine-grained App Level permission controls [10, 11], APIs to specify why an app needs access to the user's data, and privacy notifications, to name a few. Researchers have also proposed a number of enhancements to these stock OS privacy primitives and extended them in several significant ways, including providing anonymized data to apps [1, 25], giving users privacy recommendations [1, 2, 24], or even looking at new notification modalities to help with user privacy awareness [2]. Most of these mechanisms, however, are still geared towards controlling access to user data on a per-app level granularity.

In this paper we show that these app-level permission mechanisms are in fact suboptimal at protecting user privacy since more often than not, it is various third-party libraries included within apps that access sensitive user data rather than the apps themselves. The Google Admob library may, for example, access location to deliver targeted advertisements. Our insight is that in the current model of app-level permissions, if the user allows access to their private data in *even a single* app that uses the third-party library, their privacy is not protected since the library developer now effectively has access to their private data. What is even worse is that it provides users with a false sense of privacy since they may have denied access in certain apps but not necessarily all of the apps with that particular library. Furthermore, users are still unable to distinguish whether a privacy-sensitive data item, such as location or their identity, is needed for a core functionality of the app or is just being sent to various third-party libraries.

We show that in order to provide users with a privacy primitive that is effective, we need an approach that can distinguish between private data accesses by the app developer themselves in their native code versus those by third-party libraries. This information can then be presented to the user, providing them key contextual information on whether the app or a third-party library is responsible for data access, enabling them to allow or deny them accordingly. By itself, this added granularity of privacy control may seem to add to the user burden by increasing the number of decisions they have to make. However, our second insight is that while there are millions of apps, there are only a handful of popular third-party libraries that are commonly used, providing a mechanism where a user can make a privacy decision about each library once and then have it apply across all apps. We show that indeed, for the common case of users with many apps, the number of decisions they need to make is reduced substantially. In addition, the added transparency makes users more likely to allow the app themselves to have access to their sensitive data if a clear purpose is identified, as is becoming more common with both iOS and Android supporting strings to specify purposes of data access [4, 17].

We have designed and implemented an Android App, called ProtectMyPrivacy (PmP), which supports this new capability of privacy control based on detecting third-party libraries included within apps. In contrast, our earlier PmP app for iOS [1] only supported app-level permissions and controls along with a novel recommendation system. PmP for Android enables users to allow, deny, or fake their private data being accessed by the app or third-party libraries. In addition, we have built a scalable backend to collect user decisions as well as runtime stack traces for library detection and classification. Our evaluation on 1,321 real users who have downloaded and used our app shows that we reduce the number of privacy leaks to third-party libraries and at the same time reduce the number of decisions that users need to make.

In summary, this paper makes the following contributions:

- We show that current app-level mechanisms are suboptimal in terms of protecting user privacy from various third-party libraries, many of which are solely responsible for all the app's accesses to sensitive user data.
- We present the design and implementation of ProtectMyPrivacy for Android, which combines runtime instrumentation of stack traces and integration with the current Android permission mechanisms, to provide a new capability of purpose-based permissions specifically based on accesses by the apps or third-party libraries.
- We deployed PmP to two App stores and over 1,321 users in the wild. We evaluated PmP with these users, and show that the number of decisions that users need to make is reduced by 25% for users with a significant number of decisions when compared to conventional app-based permissions alone. In addition, we show that sharing of sensitive user data with third-party libraries is reduced by approximately 25%.
- We show that due to the added transparency provided by PmP on which entity is accessing their data, users were on average more comfortable sharing data with the app developers themselves, citing reasons such as maintaining app functionality.

2 MOTIVATION AND BACKGROUND

2.1 Motivation

We developed and released the ProtectMyPrivacy App for iOS in 2012 [1]. We used the data collected from that first study to understand the privacy preferences of iOS users, as well as build a recommender system for privacy decisions [1]. In April 2016, we released a version of PmP for Android with similar features as those in iOS, which allowed users to control app-level permissions. For the ideas proposed in this paper, we subsequently released a new version of PmP for Android in February 2017, which added an additional feature of collecting stack traces when privacy sensitive data items, such as location or contacts, were accessed by an app using standard Android APIs. These stack traces collect a list of all functional calls preceding the sensitive data access along with class names and also include the UserID, app package name and version (e.g. com.angrybirds, v1.3), the specific data being accessed (e.g. location), and the decision made by the user (allow/deny/fake). If multiple third-party libraries accessed a particular data item such as location within the same app, we get multiple stack traces for each of them. Each user sent us stack traces only once, and our app checks to see if the particular stack trace was not previously uploaded by that user to our backend server and only then uploads it to save bandwidth and for deduplication. Based on our analysis of this stack trace data, we motivate why the current privacy mechanisms of app-level permissions are particularly ineffective. Note, the actual details on how we analyze stack traces and extract library names from them are provided later on in Section 3.3.

We first investigate the occurrences of particular classnames across the entire set of stack traces, observing that a large majority of stack traces can be attributed to a handful of libraries. Figure 1 illustrates the popularity of the top 100 libraries present across the 11,000 apps that we have collected stack traces from since February 2017. The y-axis shows, in log scale, the number of unique apps that use each library. As can be seen from the figure, a handful of libraries are extremely popular and are used by thousands of apps. Flurry, an analytics Library, and Admob, Google's advertisement library are the two popular libraries we observed. Interestingly, the popularity of libraries falls off quickly, with the 100th most popular library being present in only 11 apps in the data set of 11,000 apps used by the users of our PmP app.

Next, we investigate how many of the privacy-sensitive data accesses in our dataset were actually caused by the top 100 third-party libraries, as opposed to those by the app's custom code. Note, for this paper we consider any information that can be used to personally identify the user (e.g. various device or user specific identifiers, contact information) or their behavior (e.g. various sensor data from their phones, their location, camera) to be privacy

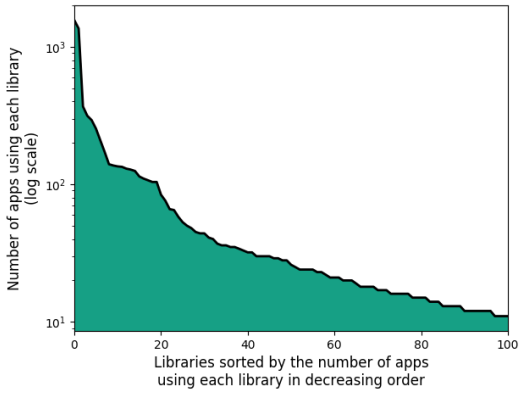

Fig. 1. The number of apps using each library falls off very quickly. The most popular libraries are used across many applications while the rest are less frequently used. The 100th most popular library is in 11 apps.

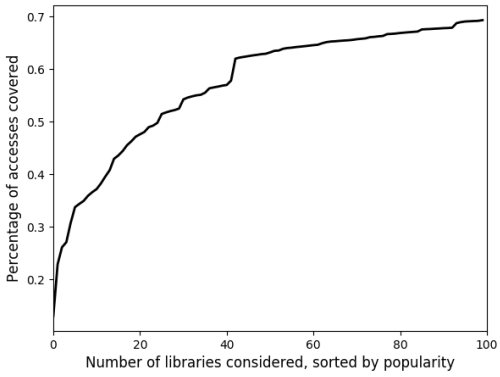

Fig. 2. Contribution of popular libraries towards sensitive data accesses across apps in our dataset. The most popular 30 libraries account for more than half of all private data accesses, while the top 100 account for 70%.

sensitive data. These privacy sensitive data accesses are currently protected by stock Android Permissions that developers need to explicitly request in their app. In addition, we have added a number of other privacy sensitive data items (e.g. Android ID, MAC address, Device IMEI) that were not protected by Android permissions to this list. Figure 2 illustrates how many of the total private data accesses by the apps in our dataset can be attributed to the top n libraries. The libraries are sorted in descending order of popularity in terms of the number of apps they are present in. As can be seen from the figure, the *top 30 libraries account for over half of all privacy-sensitive data accesses* and the top 100 libraries account for close 70% of all accesses. This finding captures the importance of private data accesses by third-party libraries in Android apps.

Next, we motivate how ineffective the current model of per-app based permissions is in terms of protecting user data from being shared with third party libraries. Note that in the current model, for a user's sensitive data to be shared with a third-party library all they would have to do is to allow access to *any* single app on their device that includes that third-party library. In other words, if Flurry accesses a user's location and it is included in 10 apps on a user's device, all the user would need to do is to allow location access for 1 of the 10 apps and Flurry would have the user's location. For this analysis, we used the privacy decisions that our users made on a per-app basis from February 2017 to April 2017, before we introduced the library based blocking feature. Figure 3 shows, the average percentage of sensitive data accesses allowed by users for the top 100 libraries. The libraries are sorted in ascending order of percentage of accesses allowed for clarity. As can be observed from the figure, in 30 libraries, users allow 50% of accesses to that library and as a result, leak data to that library.

In summary, Figures 1, 2, and 3, show that: (a) the set of popular third-party libraries included in apps is quite small; (b) a handful of third-party libraries are responsible for a large fraction of all private data accesses in apps; and (c) users unknowingly tend to allow many of the accesses made by these popular libraries thereby likely giving them access to their sensitive data. Taking these findings into consideration, we observe that detecting whether an access is caused by an included third-party library or by the app itself is key in being able to provide effective privacy controls to users.

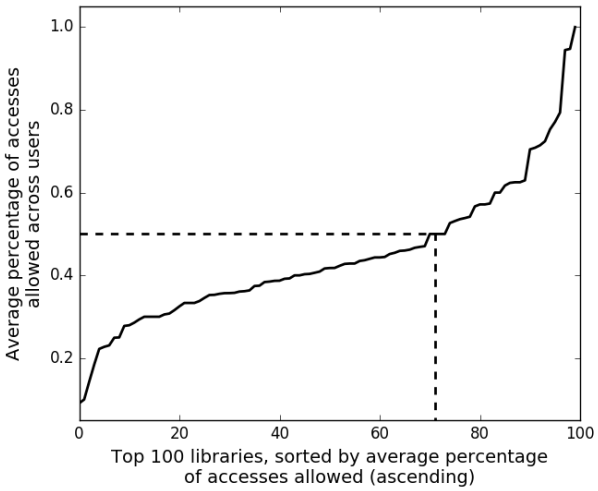

Fig. 3. Average Percentage of sensitive data accesses allowed by users for the top 100 libraries. Each data point shows the percentage of allowed accesses, averaged across all users with apps that have that particular library. As can be seen, users allow many accesses to these libraries, and in fact for 30 libraries users allow more than 50% of the total accesses.

2.2 Android Background and Threat Model

Android Permissions: The stock Android permissions are supported by the native AppOps permission manager [16]. AppOps was first introduced in Android 4.3 but was later hidden from developers in version 4.4.2. AppOps is a fine-grained permissions manager which manages 61 low-level permissions for apps, including for example 'COARSE_GRAINED_LOCATION', 'FINE_GRAINED_LOCATION', 'MONITOR_LOCATION', etc in place of the generic 'Location' permission. Users and even developers do not see the 'Ops Permissions'. Android OS maps out the manifest permissions (as defined by the developers) into the Ops permissions (as used by the OS). In addition, Android groups multiple Op permissions that are related into a single permission for ease of use. In our PmP App, we utilize and significantly extend the underlying AppOps permission manager to enable context driven, purpose based controls.

Threat Model: In this paper we assume that developers are generally honest and are not deliberately circumventing the protection provided by PmP by obfuscating their apps. We also assume that developers are well intentioned and typically offload tasks like Analytics and displaying Ads to third-party libraries to simplify development and monetization. The data being collected by third-party libraries is an unfortunate consequence of this. In essence, we are assuming that the primary goal of most, if not all, legitimate developers is to build compelling apps using user data they truly need for their app's functionality rather than just collecting this data.

3 METHODOLOGY

As we have motivated in the previous section, app-based Android permissions alone are often not effective in terms of being able to protect sensitive user data from being shared with different third-party libraries. We now describe the design and the implementation of our PmP Android app, which not only detects the presence of third-party libraries, but also provides users the controls to decide what data is shared with them. We also

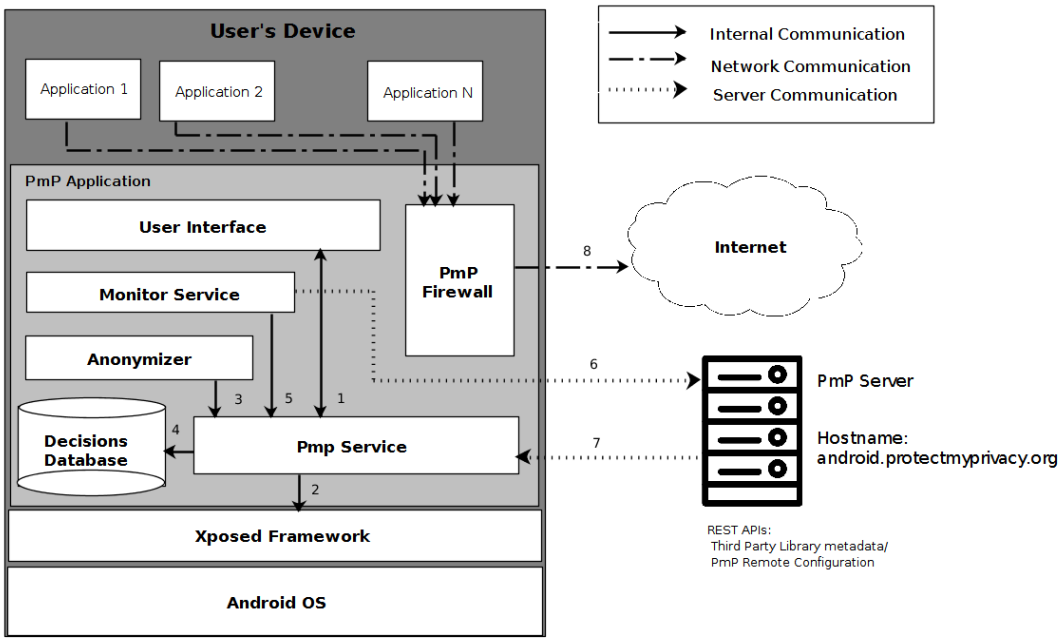

Fig. 4. The architecture of PmP. We highlight different components of the PmP Android app as well as various processing steps including (1) collecting user decisions, (2) enforcing the user decision, (3) getting fake data, (4) storing user decisions, (5) preparing to send user decisions, (6) send decisions to the server opportunistically, (7) reading remote server configurations, and (8) restricting other app's access to the Internet.

describe our PmP scalable backend that collects, stores, and analyzes various pieces of data that PmP sends about user's decisions, stack traces, and results for in-app surveys.

3.1 ProtectMyPrivacy App

Our ProtectMyPrivacy App for iOS [1], released in 2012, implemented app-level privacy controls for iOS and included a novel recommendation feature to help users with privacy decisions. In contrast, PmP for Android is designed to infer the context around data accesses, such as the entity within an app accessing sensitive data and its purpose, and in turn using this information to design more effective privacy protection mechanisms. To get a sense of why a given app accesses some sensitive data, we capture the stack trace whenever the user makes a decision (Allow, Deny, or Fake) for various privacy sensitive data items (e.g. location, contacts, identifiers, etc). Our PmP app currently works on devices running Android Ice Cream Sandwich (API 18) to Android Marshmallow (API 24). Our App utilizes the stock Android permissions manager (AppOps) [16] and additionally uses the Xposed code-injection framework [39], which requires rooting the device. We leverage and extend Android App Ops since it provides a uniform substrate of how the OS currently manages App level permissions.

Control Flow: In Figure 4 we illustrate the overall architecture of the PmP App, including the various components and the control flow within the app. The PmP Service captures the control of private data accesses via the Xposed Framework (2). Then the control goes to the User Interface where the User decides whether to Allow or Block the private data access (1). PmP then either returns a null value or fake value to replace the

Fig. 5. A single app item in our PmP app: The set of icons below the app name (Chrome) corresponds to the permissions Chrome requests: COARSE_GRAINED_LOCATION, FINE_GRAINED_LOCATION, POST_NOTIFICATIONS and so on. The green color for the COARSE_GRAINED_LOCATION icon shows that the user *allowed* Chrome to access this permission. Similarly the blue color represents that FINE_GRAINED_LOCATION information for Chrome will be *faked* and the red color indicates that the POST_NOTIFICATIONS permission is *denied*.

original value (3). The user decision is stored in the local database and opportunistically uploaded to the server (4,5,6). We describe the various steps in further detail below.

3.1.1 User Interface. Our PmP app displays information about the apps installed on the users device, the permissions used by those apps, and privacy decisions made by the user. The apps are separated into tabs for System and Non-System apps, which are displayed alphabetically. Each item in the list contains the name of the app, the app icon, and another set of icons representing the various permissions used by the app (Figure 5). Additionally, based on the decision taken for the permission by the user, different colors are set for the icons. Green represents *Allow*, Red represents *Deny*, Blue represents *Fake*, and Gray represents *Ask*. Note that setting a permission to Ask means that the user will be prompted for a decision the next time the app accesses that particular piece of private data. This scheme of using icons for permissions and colors for decisions conveys the desired information to users in a concise and intuitive manner. When an app tries to access private data, PmP displays a notification to the user requesting them to Allow, Deny, or Fake access for the requested permission. Since, the UI might provide a number of privacy controls and visual cues, we have included an on-boarding tutorial on the first launch of our App, which walks the users through the individual UI elements and the controls it provides.

3.1.2 PmP Permission Manager. As we mentioned before, we leverage and extend the functionality of the stock Android permission manager, AppOps, for the PmP App. Our PmP permissions manager interacts with the permissions at the level of the ops, giving users the flexibility to make decisions on the basis of individual op permissions, as well as permissions that have been grouped together by AppOps. Thus potentially, the user could Allow Coarse-Grained Location to be shared while blocking Fine-Grained Location for the same app.

In addition to the standard privacy sensitive data items (e.g. location, contacts, etc) that are protected by Android permissions, our PmP permission manager defines various additional personally identifiable information (PII) types. For example, we have added a 'Phone Unique Id' permission which maps to the Android ID and the advertising ID. Second, we define a 'MAC Address' permission which maps to the MAC address of the device. Further, we wanted users to be able to control network access on a per-app basis. To support this functionality, we have added a 'WiFi' and a 'Mobile Data' permission to AppOps.

3.1.3 Xposed Framework. Xposed is a code injection framework for modules that can change the behavior of the system and apps without modifying any APKs [39]. PmP uses the Xposed framework to tap into the Android functions accessing private data. We intercept the method calls before and after the access to private data is made (using a Xposed based technique called 'hooking'). To implement the blocking functionality, PmP skips the execution of the original function or modifies the results returned by the original method call.

For example, to deny an application access to 'Contacts', PmP modifies the value of the returned function to return a different value instead of the actual value. To implement the 'Deny' functionality, PmP modifies

the results to return null values. However, since Android ID can be accessed without any user consent in stock Android we do not expect (and have empirically observed) the app developers to implement exception handling in case the Android ID fails to fetch a value. Consequently, we provide an option to return 'Fake' values instead of null to prevent apps from crashing. The 'Deny' functionality is available universally for all ops across all apps, however the 'Fake' functionality is available only for a subset of the ops (described more in the next section).

3.1.4 Anonymizer. The specific fake values returned for various privacy sensitive data items are determined by PmP's anonymization module. This module takes the return values of the original method calls for accessing private data as the input and then returns instead a fake value which matches the original returned value in structure (e.g., total length, characters used).

To anonymize phone unique IDs (Android ID, Mac ID), we generate a random ID based on the patterns followed by these IDs. However, instead of using the same ID for every app, we give a different ID for each app on the user's device. The fake ID for each app remains constant throughout its lifetime, since some apps may use it for core functionality (e.g. user preferences). To anonymize location, we change the input location into a coarser location i.e. instead of giving a specific location, we give the location at the granularity of the city. This approach offers some privacy while still allowing location-based searches to work. Additionally, the user has the option to choose any location as the fake location via a Maps user interface. To anonymize contacts, we parse the characters in the individual fields of the contacts and shuffle them while maintaining the integrity of the field value (for example, the country code in phone numbers or the structure of the email id). Note that we do not fake other permissions like Storage, Wake Lock, or Internet since there is no obvious benefit to the user's privacy by faking these permissions. Moreover, the behavior of the apps under these scenarios can be unpredictable and might lead to crashes. For instance, we can simulate an unmounted storage state, but apps might still try to access storage and can crash.

We verified the effectiveness of our anonymization module by manually going through the Top 300 free apps (including the Top 15 apps in all categories) on the Android play store and provided the apps with fake data whenever they requested permissions. None of these apps crashed when supplied with fake data from PmP's anonymizing module. Since these apps are the most popular apps, they are presumably programmed well to handle exceptions elegantly. Consequently, we ran a Monkey script [15] on the next 700 most popular apps (by number of users) from the PmP database. The monkey performed a random exploration on the App UI for 2 mins per app, and supplied fake data to the apps in place of sensitive data. We realize that the monkey might not provide full coverage of all code paths in each app, therefore, we went to our database and manually exhaustively tested the apps ranked 1000-1020 in terms of the number of users. This was done to verify and test the anonymization functionality among these less popular apps. None of these apps crashed when supplied with fake data. These results provide some confidence that faking data via PmP does not lead to crashes in the top apps. Even if some of the less popular apps do crash when provided with fake data, the impact of those crashes would be minimal since prior work has shown that most of the people's usage is concentrated around the popular apps [23]. In these cases, users can also choose to allow access to the particular app, albeit at the cost of privacy.

3.1.5 PmP Firewall. The Xposed based architecture is used for all ops except the PmP defined Firewall Ops (Wifi and Mobile Data). Android OS delegates the handling of these permissions to the underlying Linux kernel. Therefore, PmP has a separate module to interact with the `iptables` framework in the Linux kernel and implement a network firewall functionality. We define a separate PmP firewall chain containing the modified iptables rules. Blocking Internet access to the apps is done on the basis of their Android process ids. The OS flushes the firewall rules whenever the device reboots, so we set up a boot broadcast receiver to reapply the firewall rules whenever the device reboots. We also receive a Network Change Broadcast from the system and reapply PmP's rules if the iptables rules have changed. The Firewall feature is optional by default. Note that we do not apply the Firewall rules to the system processes to avoid breaking functionality essential to the device.

3.1.6 PmP Service. PmP has a service running in the background which gets a callback from the Xposed framework whenever any app accesses private data. At this point, we first look for the app-permission pair value in PmP's local Ops database. If the user decision has been made, the PmP service calls the PmP modules (either the PmP Firewall or the Xposed framework) to enforce the user decision. On the other hand, if a user decision has not already been made, we look at the global decision for the permission group. If the global decision has been made we enforce that decision and show an advisory notification to the user. However, if the global decision also has not been made, we show a notification to the user asking the user for a decision on the app-permission pair. Note that until the user makes a decision, the access to the private data, and consequently the App, is blocked. The PmP service then communicates to the Monitor Service, which is responsible for storing and sending the user decisions to the PmP server (provided the users have agreed to contribute the data). The Monitor Service communicates with the PmP server and transmits all the user made decisions along with the stack traces of private data accesses to the PmP server. These components and the data flow between them is illustrated in Figure 4.

3.2 PmP Server

The PmP server is used to store the user's decisions anonymously. The server is implemented in Python using the Django framework [8]. We use the Let's Encrypt SSL Certificates for encrypted SSL communication between clients and our server and MySQL as our database. To protect the privacy of our users, we do not capture the IP address or any other personally identifiable information from the PmP app. We do store a PmP generated, one-way hash of the device id, to distinguish the users and their decisions in the database. This id can not be used in any way to track the users. To protect our database against Sybil attacks, the PmP app sends app-specific Read/Write Keys to the server along with the user decisions. The Read/Write key is a hash of a number of data items sent in each user decision by our app, combined with a shared secret. We similarly generate a Read/Write key on the server (using the data sent by our app and the shared secret) and compare it to the Read/Write key sent by the app to ensure that the decisions are indeed being sent by the PmP app. User decisions are only stored into the database if the Read/Write key generated on the server and ones sent by the PmP app on each users device match. The PmP client app also contacts the PmP server once a day for two reasons. First, the client queries the server for updates and whether they are critical (i.e. PmP will force the users to upgrade). Second, it sends a heartbeat to the server giving us the count of the total number of active users.

3.3 Library-Based Blocking

As we have discussed in Section 2.1, the analysis on the stack trace data we collected motivated the need for context specific privacy controls, specifically blocking access to user data by third-party libraries. To implement this feature, we first needed to determine the popular third-party libraries used in Android Apps. From the stack trace data (from 11,000 apps), we were able to find class names that repeatedly made accesses to private information, and were present across multiple apps. We used a threshold and considered only libraries that were present in at least 1% of the apps analyzed, which reduced the number of libraries to 30. As illustrated in Figure 2 earlier, over half of all private data accesses by apps can be attributed to these top 30 most popular libraries. Therefore, we decided to first support privacy controls for these top 30 libraries within the PmP app. Researching each library to determine what data it accesses, as well as the likely purpose of access is currently done manually by going through the details and the APIs of the library and any public information available on it, such as on its website. Our goal is to continue to manually research and add even more libraries as they become popular. While the results presented later in this paper consider these 30 libraries, we have since updated our App (On July 11th, 2017) with support for the top-60 libraries and are in the process of collecting user decisions regarding them.

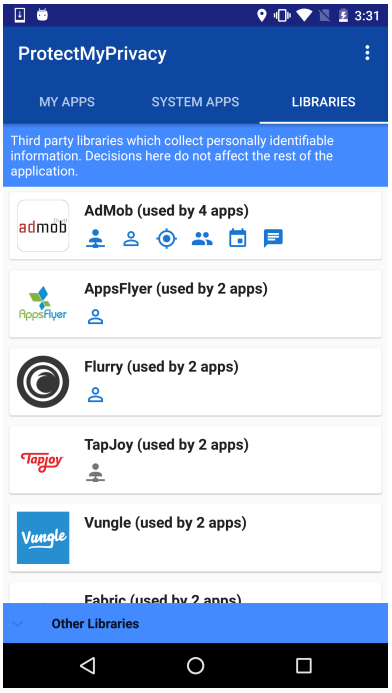

Fig. 6. Screenshot of the PmP app library tab.

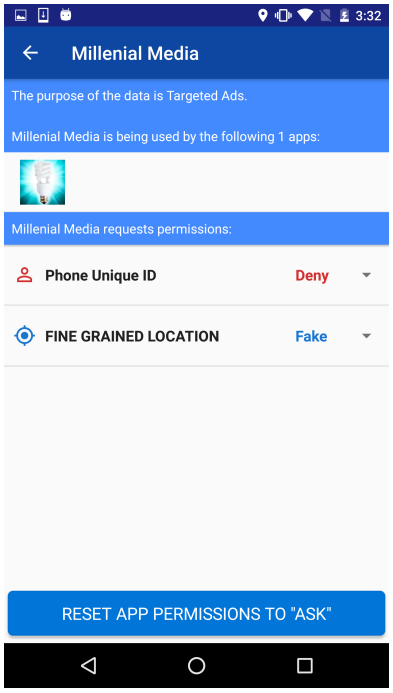

Fig. 7. Screenshot of the PmP app library details page.

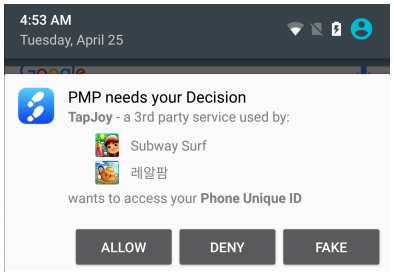

Fig. 8. Screenshot of the PmP notification to make decision for a library.

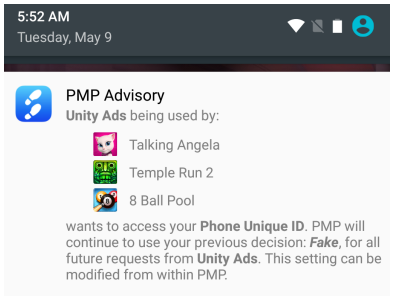

Fig. 9. Screenshot of the PmP advisory notification to inform the users about background accesses.

Note that increasing the list of supported libraries to the top 100 only increases coverage to 70% of all sensitive data accesses, providing diminishing returns (Figure 2).

The PmP App is meant to be used by real-world users, and many of them may not be aware of what a third-party library is. We associated third-party libraries with the apps that include them to provide the user with some context and also provide some information about the library, including its purpose, in the PmP App. In addition, the list of *all* apps that the particular library is in, especially the apps that a specific user has on their smartphone, is a key piece of information that is needed for that user to make informed privacy decisions. Since PmP only knows about sensitive data accesses, and gets stack traces at runtime when an app is launched, we need a mechanism to pre-fetch all the libraries supported and which apps on the user's device have them. To do so, during PmP installation, or when a new app is installed on a user's device, we send the list of apps a user has installed to the server and fetch the list of libraries that have been detected in each of those apps, if available. Unfortunately this would typically mean having this mapping for all million-plus apps in the app store, which is challenging. Instead, we provide this mapping for the most popular apps in the Google play store by manually testing PmP with the top 300 apps in the store and thereby pre-populating the app-library pairs into our database. In addition, we ran a Monkey script [15] on the next 700 popular apps to fetch additional app-library pairs for a combined total of the top 1000 most popular apps. With these mappings already present on the server, a new user can be provided information about the third party libraries present in the app even before the app is actually launched. Furthermore, we add new app-library pairs to this set as PmP detects new apps from our userbase, thereby addressing the cold-start problem to some extent.

3.3.1 PmP UI for Third-Party Libraries. We introduced a separate tab, within the PmP App, called `Libraries` for displaying permissions accessed by third-party components (Figure 6). Similar to the `My Apps` and `System Apps` tabs we display a list of third-party library names along with their respective icons and the sensitive data each one accesses. The list is sorted by the relative popularity of the third-party library i.e., the number of apps on the user's device which use the particular library. We also display the number of apps that use each library, next to its name, so that users get a sense of the prevalence of each library on their device. The rest of the libraries (among the ones we support) that are not in any of the apps on the device currently are shown in a separate view named `Other Libraries`.

When a third-party library tries to access a permission for the first time we show a notification asking the user to make a decision (Figure 8). In this notification we show the library name and the list of apps (with their icons) which use that library. This display is to clearly inform the user that this access is separate from the ones being made by the apps themselves and also to indicate that the library is being used across multiple apps. Once the user makes a decision for a library we use that same decision across all apps in which the library is present. In addition, we display a PmP Advisory notification (Figure 9), when the library accesses the sensitive data for the second time, to inform the users that the data accesses happen in the background and remind them of their decision.

3.3.2 Purpose of Third-Party Libraries. To address the question of what third-party libraries do and why they are used, we display the purpose of each third-party library. To come up with a taxonomy of the purposes, we manually went through the documentation, the APIs, and the information on the websites of the top 30 libraries we observe and converged on the following 5 purposes: (a) Targeted Ads, (b) Analytics, (c) Social Network Service, (d) Game Engine, and (e) Development Aid. We labeled the top 30 libraries with one or more of these purposes.

3.3.3 Enabling Library-Based Blocking. To enable library-based permissions in the PmP app, we use the stack trace collected at the point where the sensitive data is being accessed in the app. We parse the entire stack trace, and extract the specific class names corresponding to libraries, which are then matched with the list of libraries that PmP app has downloaded from our server. If there is no match with known libraries, we mark that access as being native to the app and the user is prompted for a decision as is done with app permission based privacy protection mechanisms. On the other hand, if the stack trace and library prefix match for a known library, we notify the user and ask them to make an allow, deny, or fake decision for **all subsequent accesses** made by that library on that sensitive data, across all apps. A limitation of this method is that when libraries use explicit obfuscation or inconsistent class names we will not be able to identify them. These issues are addressed in Section 5.3. However, by cross referencing our set of detected libraries with other analysis of popular libraries [3], we are confident that we are able to identify the popular Android libraries correctly.

4 EXPERIMENTAL RESULTS

To evaluate the efficacy of library-based blocking in our PmP app, we deployed it to the Google Play and the Xposed stores and evaluated it on real users who organically discovered and downloaded our app. In April 2016, we released the first version of our app and for almost a year, until April 2017, it only had app-permission level control. Overall, our data shows that we have over 10,000 users since the release, using over 50,000 unique apps. We began collecting stack trace data starting from around February 2017 and we have about 11,000 apps for which we have stack trace data. On average we see about 1500 active users per week, who open and use our app at least once and whose decision data we collect. Note that contributing their decision data is entirely voluntary and our analysis only includes the users who choose to turn contribute on. The particulars of our study were reviewed and approved by our Institutional Review Board (IRB).

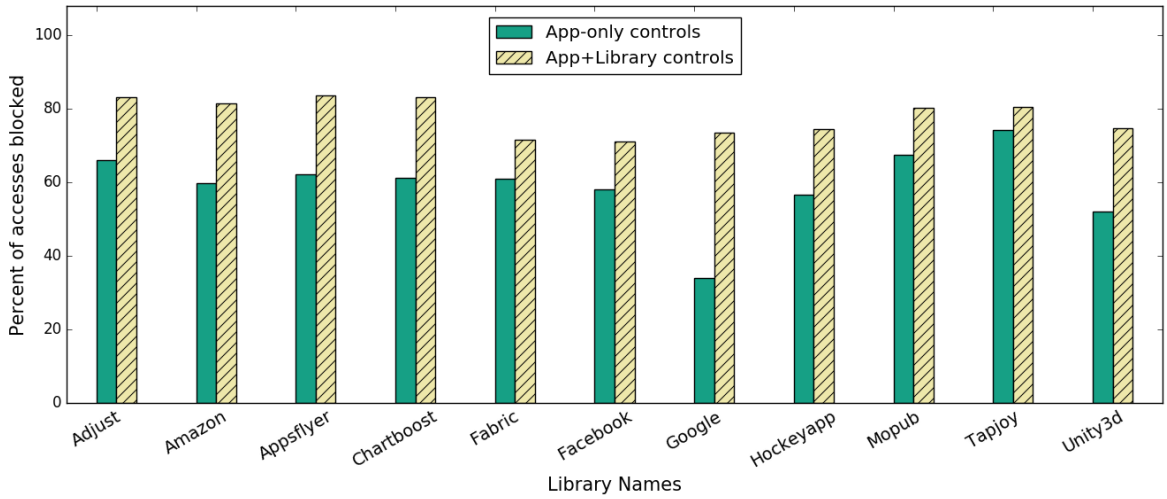

Fig. 10. Percentage of sensitive data accesses denied across all users, shown per library. In general, the percentage of accesses that are denied increase substantially in the App+Library model, as compared to the App-only model.

For the first phase (April 2016 - April 2017) our users only used the app-level permissions and made decisions. In April 2017, we released a new version of our App that implements both App- and Library-based permissions and reset the decisions made in the previous version after notifying the users. We evaluate the efficacy of this novel approach across several metrics. We first compare and contrast app-level only privacy controls with the App+Library level controls, showing that users are better protected under the new model. Next we evaluate the number of decisions users have to make with both privacy models, showing that users have to make significantly less decisions in the App+Library model. Finally, we evaluate the decisions themselves in both models to see how they have changed for both native-app accesses, as well as for libraries. We show that in general, users are more conscious in terms of giving their data to third-party libraries. Finally, we study how the App+Library model impacts user's decisions and report on the experience sampling based on user feedback through an in App survey to understand the reasoning behind those decisions.

4.1 Effectiveness of App+Library Based Controls

We first compare how many of the sensitive data accesses made due to libraries within apps were blocked (i.e. by setting deny or fake as their permission setting) in the default app based controls, as compared to the new App+Library based controls. We filtered our data to only include libraries for which we had at least 25 decisions made in the new App model, which led to a total of 11 libraries. Figure 10 shows the percentage of accesses blocked for each of these libraries in both models. As can be seen from Figure 10, in the new App+Library model, a much higher percentage of sensitive data accesses made by third-party libraries were blocked resulting in less data flowing to the libraries. The difference is quite prominent in libraries like Chartboost, Appsflyer, and Google Admob, while its less so for Facebook, Tapjoy, and Mopub.

Next, we evaluated the effectiveness of our App+Library model by comparing the percentage of users actually protected in terms of their sensitive data not being sent to these third-party libraries based on their decisions, for both models. Being fully protected is particularly important since in the App-level model, a user would only be protected if **all** of the accesses by the library across their apps were blocked. Figure 11 compares the percentage

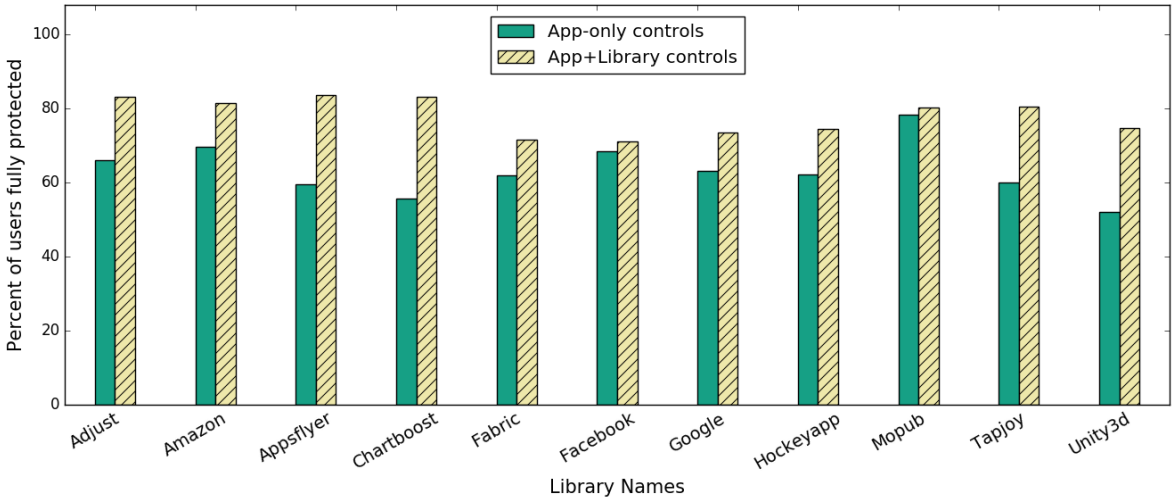

Fig. 11. Percentage of users protected, shown per library. In the App+Library model, the average percentage of users whose data is sent to various third-party libraries has been reduced substantially, as compared to the app-only model, showcasing that users are better protected from privacy invasive libraries in the new model.

of users who are fully protected (no data leakage) between the App-only controls and the App+Library controls. As can be seen with the App+Library controls, a significantly higher percentage of users are protected from their data being sent to these libraries as compared to the status quo of App-only controls. We attribute this change to the fact that PmP now makes sensitive data accesses much more transparent within each app, enabling users to make informed decisions, and any data shared with these libraries is based on conscious decisions.

4.2 Number of User Decisions

Next we study the number of decisions users have to make in the App-only model versus the new App+Library model. The purpose of this evaluation is to show that indeed the number of decisions users need to make in the App+Library model are reduced due to libraries being shared across multiple apps.

We first study all the stack traces we have collected to categorize how many of the sensitive data accesses happen due to third-party libraries, and how many of them are due to native app functionality. In Figure 12, for each of the 11,000 or so apps in our dataset that access some sensitive data we plot the percentage of accesses within each app that is native to the app and those that are due to an included third-party library. For example, if an app accesses location for an internal purpose, and also includes Flurry which accesses location itself, the percentage of accesses belonging to a third-party library (Flurry) within it would be 50% for that App. As can be seen from Figure 12, 82% of apps either access sensitive data for only internal purposes (40%) or only third-party libraries (42%). Fewer apps have a mix of both third-party and native accesses (18%). This result is important since apps with both third-party and native accesses could require two or more decisions (1 app-level and 1 for each library) for protection. On the other hand, applications with only third party or only native accesses can be protected against by one decision, either at the library-level or at the app-level, respectively.

Under the new App+Library model, decisions at the library level only need to be made once and apply to all the apps that have the same library. We expect that users that have more apps, and consequently make more decisions, will benefit from the shared library decisions since its likely that many apps would have the same

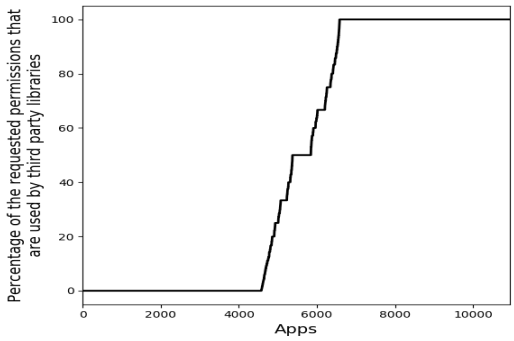

Fig. 12. Percentage of accesses within an application that are from third-party Libraries. Most apps are either composed of all native access or all third-party accesses.

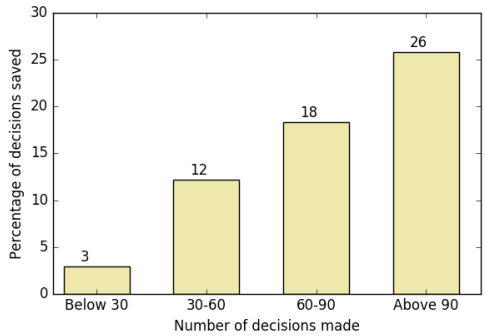

Fig. 13. Percentage of decisions saved bucketed by number of decisions made. The percentage of decisions saved are averaged across all users in a bucket.

library. In contrast, in the App-only model, they would have had to make many decisions. Figure 13 illustrates how the percentage of decisions that users have to make are reduced by using the new App+Library model and users that have to make more decisions experience a greater benefit. For example, users that had to make more than 90 decisions in the app-only model have to make 26% less decisions on average in the new model of control.

This finding is important since prior work has shown that the number of user decisions are a limited resource, therefore the number of decisions reduced by our new App+Library model will help in improving usability [24].

4.3 Effect on Users' Decisions

Next, we analyze how the decisions that users made vary across the two different models, and what effect App+Library controls had on the overall decisions across our entire population of users. Figure 14 shows the percentage of sensitive data accesses blocked (e.g. Deny or Fake) by users across All Apps for three different cases. "App-Only controls" denotes the status quo where only App based decisions can be made. "App+Library controls on Native Access" and "App+Library controls on Third-Party Library access" denote the percentage decisions to block accesses that are done by the App themselves, and those by libraries respectively. As can be seen from Figure 14, across all features, the percentage of app-native decisions increased from 40% to 49%, while the increase was 47% to 57% and 39% to 43% when Phone Unique ID and Location features are separated out. We believe that overall, users are a bit more cautious with the new App+Library model when it comes to giving App's access to their sensitive data. More importantly, the percentage of data accesses blocked when they come from libraries are 78% for all features, and 77% for Phone Unique ID and 81% for Location respectively. This data shows that users really don't want to allow libraries access to their data. Note, we have only separated out Phone Unique ID and Location as features since they are the most common in libraries and often cause the most privacy concerns.

As noted earlier, there are a number of apps in which sensitive data is accessed by both the native app code as well as by the included third-party libraries within them. In the current App-only control model, users would not be able to differentiate between these accesses and make a decision based on incomplete information. For these set of apps, we wanted to evaluate whether users decisions changed in any particular way, with the added transparency that PmP provides. For this analysis, we only pick apps with both native and library accesses. In Figure 15 we show a scatter plot where the x-axis is the percentage of decisions regarding native-app accesses

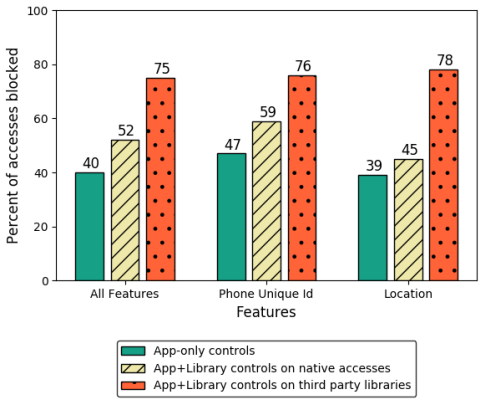

Fig. 14. Percent change in the number of block decisions, across different features, from the app-level blocking model to the library based blocking model. Overall, the users have become more cautious and block third-party libraries.

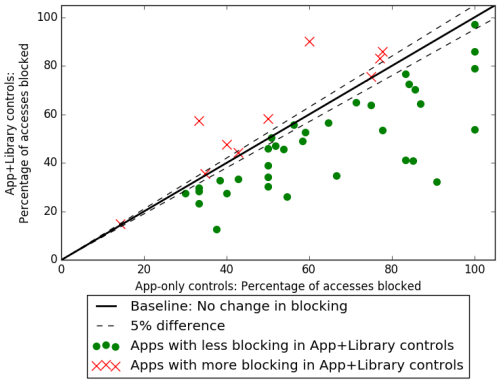

Fig. 15. The percentage of native blocks with app-only controls vs with app+library controls. This is across apps which have **both** native accesses and third-party library accesses. Users have become more lenient when it comes to giving information to app developers (however more cautious with third-party libraries)

that are blocked using App-only controls, and the y-axis plots the percentage of accesses within the native app blocked using the new App-Library model. Note that, for this analysis we further filtered the decisions by the users who had used the same apps with both the models of PmP. This reduced the number of apps to 46. Since we reset the user decisions when updating the version, the users would have had to make similar decisions. While the total number of apps represented in this analysis is relatively low due to multiple filters, a clear trend is still visible. A point on this graph represents a single app. As can be observed from Figure 15, with the new model our users were generally more comfortable allowing access to sensitive data by the native app code, than in the App-only model. We believe that this is likely due to our users becoming more comfortable with these native accesses, now that they know those accesses are not by third-party libraries.

4.4 Comparing Behavior for Users who have used Both Models

For the results presented earlier, we included users who have only used the App-only model of control in our PmP App, those that have used both App-only and App+Library models since they upgraded PmP in April 2017, and finally new users who have installed and only used the new App+Library model. We wanted to study the users who have used both the models and how their decisions changed. Note, for users who upgraded to the App+Library model we notified them of the new feature of library based blocking and then reset all their decisions, one time. Table 1 buckets based on the percentage of native accesses within their apps that this set of users blocked. We put these users into three buckets, those that blocked less than 25% of sensitive data accesses, 25% - 75% of accesses, and over 75% of accesses, in the App-only model. The table shows that users who were more restrictive, i.e. 25% - 75% and more than 75% , tended to block less native app accesses on average when compared to the previous model. We believe that this is a result of users increased trust in the app's native access after protecting against libraries. For the most permissive users i.e those with less than 25% denies, we saw a increase in the blocked percentage. We hypothesize this is due to a better understanding of third-party libraries and as a result being more cautious.

Table 1. Change in percent of native accesses blocked for users who have used both the app-only and app+library based model. The users are bucketed by their percentage of native accesses blocked.

Users grouped by the percent of native accesses blocked	Percentage of native accesses blocked	
	App-only controls	App+Library controls
Below 25%	8.02	20.37
25-75%	35.80	64.19
Above 75%	87.03	79.01

4.5 Understanding the Reasoning Behind User Decisions

While we have evaluated quantitatively how users' decisions change as App+Library based blocking is introduced, we wanted to understand why users are making these decisions about third-party libraries. To do so, we designed an experiment where users were prompted right after they made a decision asking them to specify a reason for their choice. In addition, we used experience sampling to only ask users for a reason for every third decision (33%) and only for the library based decisions. These techniques have been used in prior work to get additional context about a user's privacy decision [24] on smartphones.

To come up with a set of reasons, to both allow and block sensitive data access to libraries, we recruited 5 privacy researchers from our university and asked them to independently come up with a list of reasons they might allow/block a library. Then, we asked them to discuss among themselves to come up with a combined list of reasons that were most representative and non-overlapping. Ultimately, we had three reasons for allowing and three reasons for blocking access that users had to choose from. Users could also choose not to specify any reason if none of them matched their mental model. The list of reasons to block access include: (a) I don't trust this library; (b) I'm not ok with sharing this specific data (the notification also mentions the specific data item such as location); and (c) I don't want to share any data (Figure 16). In contrast, the decisions for allowing access include: (a) I allowed data access for functionality; (b) I'm ok with sharing this private data (the notification also mentions the specific data item such as location); and (c) Sharing data supports free apps (Figure 17).

The first reason of the 'Allow' group seems counter intuitive, since the decisions made for third-party libraries, do not impact the usage of the apps. However, 4 of the 5 researchers came up with it independently, with the reasoning that they were okay with sharing their 'Coarse Grained Location' to get somewhat relevant Ads or that they wanted to share analytics data with the library so that the app developer could improve their apps. In essence, option 1 is in support of the current ad based revenue model, in the context of a specific app while option 3 is a more generic call for substituting money with data.

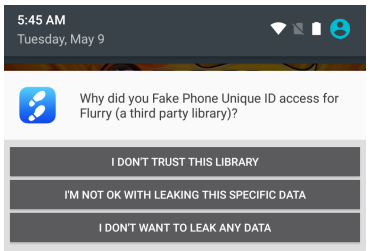

Fig. 16. PmP Decision Rationale for Fake (Block)

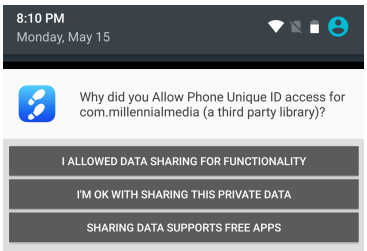

Fig. 17. PmP Decision Rationale for Allow

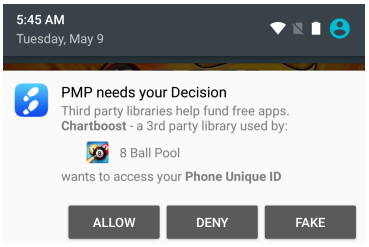

Fig. 18. PmP Decision Request for Third-Party Libraries

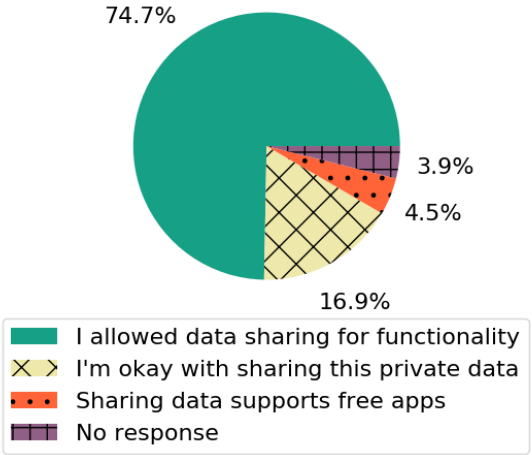

Fig. 19. User responses on 'WHY' they allowed sensitive data access to various third-party libraries.

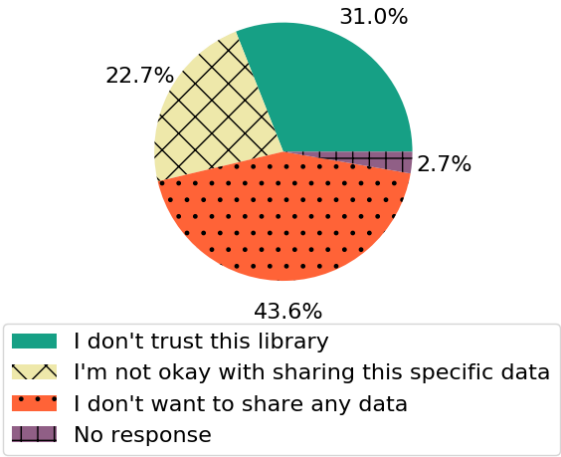

Fig. 20. User responses on 'WHY' they blocked sensitive data access to various third-party libraries.

Figure 19 and Figure 20 show pie charts with the user responses on the various reasons why they allowed, or denied sensitive data access to libraries respectively. From Figure 19 we observe that around 75% of the responses indicate that the user allowed access for functionality, with 17% of responses denoting the reason as the user being OK with sharing that specific sensitive data (e.g. location) with that library. Very few users (5%) specified the reason as sharing their data to support free apps. In Figure 20 we similarly show the responses users provided for blocking access to sensitive data, with most responses indicating a reluctance to share any data with libraries.

4.6 Effect of Positive or Neutral Framing of the Notifications

As a final evaluation, we wanted to test whether different framings of the notification, where we let the user know which library is accessing sensitive data and its purpose, would have any effect on users' decisions. The goal of this experiment was to see whether educating users about how free apps are supported has any bearing on their decision.

We used neutral as well as positively framed messages. For the neutral-framed messages, we displayed "Third Party Library X wants to access Y in apps Z" while for the positive framed messages we added an additional line in the notification as "Third Party Libraries help fund free apps" (Figure 18) since this is the most common reason why developers use third-party Ad and Analytics libraries. We did not test adding a negative-framed notification since we felt it would bias the users even further where they would just block all accesses. We note

Table 2. This table represents how the user decisions are impacted by the positive message in the notifications. Group A gets positive notifications while Group B gets neutral notifications.

	Group A	Group B
Total Users	470	466
Allow Decisions	946	827
Block Decisions	2798	2804
Allow Percent	25.3%	22.7%

that blocking sensitive data from third-party libraries may make them less effective at targeting users, for example those displaying context specific ads, which would likely affect the revenue model of the developers of free apps.

For this evaluation, we divided our users randomly into two groups, and did an A-B test on them. Group A received positively framed notifications, while Group B users received neutral framing notifications. Table 2 shows the result of this study. We did not find a significant difference, since the group receiving positive-framed notifications allowed only marginally more accesses as compared to the other group (25.3% vs 22.7%). Hence our framings seem to have less of an effect on users' decisions.

5 DISCUSSION AND LIMITATIONS

5.1 Users with Rooted Devices

A potential concern is that our current user population of PmP is skewed towards more technically savvy users, since rooting a device is not something that everyday users typically do. In addition, users who install our app are likely more privacy conscious than the average user since they are self selecting and choosing to install our privacy app. We acknowledge both these limitations, but argue that the results that we have presented are still valid and representative for a number of reasons. First, rooted users download and install the same apps that non-rooted users download, from the same Google Play store. This means that the stack traces we get, as well as the behavior of apps that these users use are similar. In addition, the apps that we see in our dataset are diverse (over 50,000 unique apps), and the most used apps are well correlated with popular and well known apps from the Play Store, providing evidence that our users are representative of real world users. Ultimately, in order to do this study, which includes gathering advanced contextual information like stack traces and providing fine grained context based privacy controls, we needed to either have rooted devices, or develop and deploy a custom version of Android to a very small set of users. Since unmodified Android devices do not support the functionality we needed, we chose to build the PmP app on rooted devices using the Xposed hooking framework, and have users organically discover and use our app from Google play and Xposed App stores.

5.2 Indirect Data Access by Libraries

In some cases, it is possible that an app developer accesses sensitive data such as device ID or location themselves and then pass it onto the libraries as a parameter to one of the library functions. To use this method, the app has to repeatedly collect and pass the sensitive data (for e.g., Location) to the library. In such cases, the PmP stack trace analysis would incorrectly mark this access as being made by app and not the third-party library. However, this is not the typical model of third-party libraries since they are implemented to be easy to invoke and include, and perform their tasks (e.g., building a profile to display relevant ads) without any input or data from the developer. In fact, we ran an experiment to verify how many apps pass data to the third party libraries by explicitly passing it on to them. We used the decompiled apks for the top 5000 apps on the Google Play Store from the researchers of PrivacyGrade [23]. Then, we searched the decompiled apks for strings used to transmit the PII to the library (as found in the documentation of the various third-party libraries we detect) and the presence of a third-party libraries. In the top 5000 apps we found only 15 apps (0.3% apps) that were explicitly passing the private data to the libraries. A possible solution is to flag these apps *a priori* and inform the user that the data granted to those apps could end up flowing to certain third-party libraries.

5.3 Obfuscated Code and Custom App-specific IDs

As mentioned earlier, PmP only protects against honest developers who include various third-party libraries to ease development. PmP can not protect against apps or libraries which obfuscate their code explicitly. Fortunately, based on our stack trace data we see that obfuscation by third-party libraries is very uncommon. Even if the internal library code is obfuscated, the user facing interfaces of the libraries are not obfuscated. This behavior is

observed in Admob (owned by Google). In addition, the other libraries' method calls are not obfuscated even if the native app code is obfuscated. Therefore, PmP would be able to detect that whether the access was made from the library or the app itself.

Furthermore, PmP only protects Android-based permissions. That is, if a developer generates a custom unique random id to track users, PmP would not be able to detect it. However, the developer can only track the user's activities inside this specific app via an app-generated id. The developers can not link the user to their larger ad profile which requires the global device identifiers like Android ID. In fact we believe that this is the right way for the developer to track their users for customizing their app and its behavior on each user's device if needed for their application.

5.4 Privacy Challenges after Data Exfiltration and Collusion

In addition, there are no guarantees on what happens to private data once it leaves the users devices over the network. Developers can analyze the data for legitimate uses, or simply sell the data collected by them to others. The third-party library based permissions model introduced in this paper is meant to be an addition to the app-level permission model and not replace it. If the app accesses sensitive data for its own use, PmP currently assumes that it is needed for internal app functionality and lets the user make the app-level decision. Therefore, we assert that the users should be cautious even while sharing their data with the app developers. Furthermore, even though unlikely, if the library developers collude and share sensitive user data among themselves, we can not control it. However, if the parent company of the libraries are the same, we can essentially merge them and provide this info to the user. One direction for future work is to use static analysis techniques on decompiled apps along with tools like natural language processing and mining app descriptions to infer what the purpose of data accesses are [38] and show that information to the user to help them with their app level decisions.

6 RELATED WORK

There is an extensive body of research on evaluating Android's permission model and showing that a large number of applications access sensitive data that is not required for their core functionality [1, 9, 22, 26, 35, 37]. Prior work also shows that many applications share this data with third-party analytics and ad providers as well, often without notifying the users [6, 13, 20, 28]. The most commonly collected sensitive resources in Android applications include hardware identifiers like SSAID (Android ID) and IMEI, MAC address, location, contacts, messages and camera.

The majority of techniques used for enforcing privacy in Android can be broadly classified as either Static Analysis, Network Flow Analysis, or Android API Instrumentation.

Static Analysis: Techniques based on static analysis rely on analyzing the bytecode or source code of an application to determine accesses to sensitive user data. FlowDroid [5] uses a novel static-analysis system that precisely models the application's lifecycle. This helps produce better analysis results with fewer false positives. DroidSafe [18] improves on FlowDroid [5] by tracking information flow from Android API calls that inject information (source) to the calls that may leak information (sink) based on *accurate analyses stubs* to effectively abstract the semantics of the Android runtime. AndroidLeaks [14] uses taint-aware slicing to find potential PII leaks. Other efforts by Lin et al. [23] and Wang et al. [38] concentrate on inferring the possible purpose behind a permission request via static analysis to educate users and enable them to make better decisions. These tools tend to be more informational in nature rather than aiding users at runtime in enforcing access policy restrictions. In general, static analysis tools are ineffective against obfuscated code and cannot handle reflection and dynamic code loading gracefully either [40].

Network Flow Analysis: These tools focus on identifying and mitigating PII leaks at the network level rather than the system level. Haystack [29] uses standard VPN APIs available in Android to gain full access of the

device's network traffic and local context. It operates locally on the user's device intercepting, analyzing and then forwarding non-sensitive data to its intended destination. ReCon [30] relies on machine learning classifiers that have been trained by analyzing more than 72,000 mobile app network flows to predict and notify users of traffic that is leaking PII. The ReCon service is deployed on a cloud instance which allows it to be cross-platform and provide support for Android, iOS and Windows devices. A major drawback of such network level approaches is that they fail when faced with encrypted or obfuscated traffic, as well as impose significant processing overheads due to deep traffic inspection. Completeness is also hard to guarantee using these techniques. Dynamic analysis techniques backed by API instrumentation are able to deliver a more complete analysis.

Android API Instrumentation: These techniques dynamically modify the way in which applications are allowed to access sensitive data at runtime. TaintDroid [9] is a well known work in this field which labels privacy sensitive data at variable, method, message and file levels as tainted and tracks the flow of data throughout the system. When this tainted data tries to leave the system over a network interface, TaintDroid gives real-time feedback to users about the data, the application and the destination to which the data is being transmitted. Since TaintDroid monitors a variety of data across multiple apps continuously, it incurs a 32% runtime performance overhead. A similar work, MockDroid [7] proposed modifying Android to provide users with the ability to "mock" sensitive data accessed by applications at runtime. Although this could affect the usability of the application, they argued that it is the developer's responsibility to handle all such negative cases as part of building a robust application. The Apex system [27] extends Android and gives users the ability to allow, deny or constrain accesses to specific data requested by applications. AppFence [19] presents a dual control model of shadowing user specified sensitive data and blocking exfiltration of private data over the network. PmP follows a similar runtime enforcement technique, along with an enhanced evaluation model based on third party library blocking with a clear intuitive UI.

Permission Managers and Purpose: Permission Managers aim to assist users in demystifying complex privacy settings. Prior works have examined the usefulness of such permission managers and show how users can employ them to better restrict app access to sensitive data [2, 12]. Fisher et al. found that the majority of iOS users in their study blocked about 1/3 of their apps from accessing their location [12]. On the other hand, they also found that the permission manager by itself is not enough to reach adequate levels of privacy protection because the permission manager does not provide the users with enough information to make informed decisions. An earlier version of PmP for iOS implemented App level blocking and was built around a novel recommendation system [1]. In contrast, PmP for Android introduces library based blocking and provides users with more context to make informed decisions. The original design of PmP for iOS was therefore suboptimal in terms of preventing data leakage to third party libraries, which a focus of this paper. Recent versions of Android and iOS have begun nudging developers to provide an explanation for the permissions requested in their applications with hopes that users make informed decisions. Tan et al. evaluated the impact of such explanations in iOS apps and found that users were more inclined towards granting access to permissions when provided with explanations [34]. Shih et al. found that the clarity of the text in the explanation greatly influenced users willingness to grant permission requests. Having no explanation at all was found to be better than having a vague one [32]. Works which model user privacy preferences and generate privacy profiles for users [24, 31], have been more effective in enhancing permission managers. These enhanced permission managers help users avoid what could be perceived as intrusive apps. However, to the best of our knowledge, no prior work has evaluated permission managers on the basis of third party libraries.

7 CONCLUSION

We present the design and implementation of our ProtectMyPrivacy app for Android. PmP supports a novel feature of both App- and Library-based privacy control for accesses to sensitive data. This new model replaces

the existing App-only model used by Android, which we show to be suboptimal in protecting against third-party libraries due to their pervasiveness across multiple apps. In particular, we show that the hundred most popular libraries account for close to 70% of all sensitive data accesses. We evaluate the effectiveness of our system and the new App+Library based controls, showing that users were significantly better protected from their data being sent to third-party libraries as compared to the current App-only model. In particular, users needed to make up to 25% less privacy decisions, with 25% less data flowing to libraries on average. We also show that due to the added transparency provided by PmP in terms of which entity was accessing their data, users became more cautious with regard to libraries and more liberal with native app accesses.

8 ACKNOWLEDGEMENT

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-15-2-0281. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. We would also like to thank the following grants for supporting this work: National Science Foundation grants CSR-1526237, CNS-1564009 and SBE-1513957 as well as multiple Google Faculty Research Awards on mobile and IoT privacy.

The authors would like to acknowledge Kunal Bhuwalka and Dohyun Kim for their comments and help on the early drafts of this paper. In addition, the authors would like to thank Edison Gao, Sandeep Agarwalla, Vikram Shanker, David Simon and Joseph Chan for their work on an earlier version of the PmP App. The authors would also like to thank the anonymous reviewers for their constructive feedback.

REFERENCES

[1] Y. Agarwal and M. Hall. Protectmyprivacy: Detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 97–110, New York, NY, USA, 2013. ACM.

[2] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal. Your location has been shared 5,398 times!: A field study on mobile app privacy nudging. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 787–796, New York, NY, USA, 2015. ACM.

[3] AppBrain. App Brain Ad Libraries List. https://www.appbrain.com/stats/libraries/ad, 2017.

[4] Apple. iOS, developers have to explain the purposes for private data access. https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ExpectedAppBehaviors/ExpectedAppBehaviors.html#//apple_ref/doc/uid/TP40007072-CH3-SW7, 2017.

[5] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *SIGPLAN Not.*, 49(6):259–269, June 2014.

[6] R. Balebako, J. Jung, W. Lu, L. F. Cranor, and C. Nguyen. "little brothers watching you": Raising awareness of data leaks on smartphones. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 12:1–12:11, New York, NY, USA, 2013. ACM.

[7] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 49–54, New York, NY, USA, 2011. ACM.

[8] Django. Django - The web framework for perfectionists with deadlines. 2017.

[9] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation (OSDI)*, 2010.

[10] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.

[11] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[12] D. Fisher, L. Dorner, and D. Wagner. Short paper: location privacy: user behavior in the field. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pages 51–56. ACM, 2012.

[13] FTC. Android Flashlight App Developer Settles FTC Charges It Deceived Consumers. https://goo.gl/Zf18jI.
[14] C. Gibler, J. Crussell, J. Erickson, and H. Chen. Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST'12, pages 291–307, Berlin, Heidelberg, 2012. Springer-Verlag.
[15] Google. Android UI Automation and Testing Tool. https://developer.android.com/studio/test/monkey.html.
[16] Google. App Ops Permissions Manager: Android's permission manager. http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/5.1.0_r1/android/app/AppOpsManager.java/.
[17] Google. Android, show permissions rationale to the users. https://developer.android.com/training/permissions/requesting.html, 2017.
[18] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard. Information flow analysis of android applications in droidsafe. In *NDSS*. Citeseer, 2015.
[19] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These Aren't the Droids you're Looking For: Retrofitting Android to Protect Data from Imperious Applications. In *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS)*, pages 639–652. ACM, 2011.
[20] Y. Li, Y. Guo, and X. Chen. Peruim: Understanding mobile application privacy with permission-ui mapping. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 682–693. ACM, 2016.
[21] J. Lin, S. Amini, J. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expectation and Purpose: Understanding Users Mental Models of Mobile App Privacy Through Crowdsourcing. In *Proceedings of the 14th ACM International Conference on Ubiquitous Computing (Ubicomp)*, 2012.
[22] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 501–510, New York, NY, USA, 2012. ACM.
[23] J. Lin, B. Liu, N. Sadeh, and J. I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 199–212, Menlo Park, CA, July 2014. USENIX Association.
[24] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. Zhang, N. Sadeh, A. Acquisti, and Y. Agarwal. Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *Symposium on Usable Privacy and Security*, 2016.
[25] Marcel Bokhorst. XPrivacy. https://github.com/M66B/XPrivacy.
[26] S. Mirzamohammadi and A. Amiri Sani. Viola: Trustworthy sensor notifications for enhanced privacy on mobile systems. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 263–276. ACM, 2016.
[27] M. Nauman, S. Khan, and X. Zhang. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (CCS)*, pages 328–332. ACM, 2010.
[28] Path. Path official blog. http://blog.path.com/post/17274932484/we-are-sorry, 2017.
[29] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, and V. Paxson. Haystack: A multi-purpose mobile vantage point in user space. *arXiv preprint arXiv:1510.01419*, 2015.
[30] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374. ACM, 2016.
[31] J. Sadeh and J. I. Hong. Modeling usersâĂŹ mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Symposium on Usable Privacy and Security (SOUPS)*, volume 40, 2014.
[32] F. Shih, I. Liccardi, and D. Weitzner. Privacy tipping points in smartphones privacy preferences. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 807–816, New York, NY, USA, 2015. ACM.
[33] Statista. Trends in Consumer Stats. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2017.
[34] J. Tan, K. Nguyen, M. Theodorides, H. Negrón-Arroyo, C. Thompson, S. Egelman, and D. Wagner. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 91–100, New York, NY, USA, 2014. ACM.
[35] S. Thurm and Y. Kane. Your Apps Are Watching You. *The Wall Street Journal*, 2010.
[36] N. Y. Times. Mobile Apps Take Data Without Permission. http://bits.blogs.nytimes.com/2012/02/15/google-and-mobile-apps-take-data-books-without-permission/.
[37] M. Van 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 *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 5208–5220, New York, NY, USA, 2017. ACM.
[38] H. Wang, J. Hong, and Y. Guo. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, pages 1107–1118, New York, NY, USA, 2015. ACM.
[39] Xposed. Xposed Installer. http://repo.xposed.info/module/de.robv.android.xposed.installer, 2017.
[40] Y. Zhauniarovich, M. Ahmad, O. Gadyatskaya, B. Crispo, and F. Massacci. Stadyna: Addressing the problem of dynamic code updates in the security analysis of android applications. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, pages 37–48, New York, NY, USA, 2015. ACM.