RICE UNIVERSITY

Privacy Concerns in Android Advertising Libraries

by

Theodore Book

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Approved, Thesis Committee:

Dan S. Wallach, Chair
Professor of Computer Science

Eugene Ng
Professor of Computer Science

Lin Zhong
Associate Professor of Electrical and Computer Engineering and Computer Science

Houston, Texas

August, 2013

ABSTRACT

Privacy Concerns in Android Advertising Libraries

by

Theodore Book

This work investigates privacy characteristics of Android advertising libraries. Taking a sample of 114,000 apps, we extract and classify their ad libraries. We then seek to understand how they make use of sensitive user data. First, we study the use of permission-protected Android API calls that provide access to user data. Here, we measure change over time by distinguishing unique versions of each library, dating them, and calculating their permission usage. We find that the use of most permissions has increased over the last several years, and that more libraries are able to use permissions that pose particular risks to user privacy and security. Next, we shift to the application side and consider information passed directly from the application to the ad library. We do this by reconstructing the APIs for our libraries, and examining how those APIs are used in our sample of Android applications. We find that many applications pass personal information directly to their ad libraries, without any need for the library to query the operating system directly. This behavior is most common in more popular applications, suggesting that the promise of advertising dollars encourages application developers to violate users' privacy. In sum, we find that ad libraries make use of both the operating system and their host application to collect sensitive information about their users.

Contents

Illustrations

Tables

Chapter 1

Introduction

The Android operating system is one of the primary mobile device platforms worldwide, accounting for 70% of smartphone shipments [1]. It also supports a vibrant advertising industry, with dozens of advertising agencies providing ad libraries installed in hundreds of thousands of free applications. Indeed, for every paid app downloaded from Google Play, users download 82 free apps, mostly ad supported [2]. Privacy threats on Android are similar to those on iOS, where apps exhibit similar privacy leakages [3]. The diversity of these libraries and the possibility of studying their deployment in real world applications makes Android an ideal platform to study the impact of advertising libraries on user privacy and mobile device security.

Similar to the web domain, the dominant monetization model for applications on the Android platform consists of free applications whose developers receive compensation through the sale of ads [4]. However, the technical infrastructure for this advertising model is fundamentally different from the web model. Rather than displaying ads in iframes using technology built into the web browser, each agency builds its own display software, constructing a mini web browser that is built into participating developers' applications. These advertising frameworks consist of compiled Java libraries with arbitrary code supplied by the advertising networks. The developers who bundle ad libraries into their applications generally can not inspect the library's code to understand its behavior. Instead, the developer uses the library's API to integrate the library into the application, passing information to the library and enabling it to display ads. These libraries, in turn, communicate with servers controlled by the advertising agency, sending request information, receiving and displaying an ad-

vertisement, and handling user interaction with the ad, including redirects to content and accounting functions.

This model lacks many of the mechanisms that protect web content and users from malicious advertisements. On the web, advertisements are generally isolated in an iframe that separates any advertisement code (generally written in javascript) from the content of the surrounding web page. While malicious agents may occasionally find exploits against this architecture, in principle, the same origin policy protects the rest of the page from the advertisement. Likewise, the web browser separates the advertising code from operating system, placing massive restrictions on the way an ad can interact with the host operating system [5].

However, very few of these protections exist in Android applications. Because the library is a binary portion of the application, it has few restrictions on what it can do. Not only can the library download and run javascript, but in some cases libraries have been found that download and execute arbitrary dalvik bytecode [6]. The position of the library within the application allows the library indiscriminate access to the application's resources. Likewise, this status allows the library to interact with the operating system with the same privileges as the application.

The Android operating system does extend standard Unix mechanisms to provide protections against one application accessing the private data of another application or of the operating system. Through sandboxing and the policy of running each application with a unique user ID, applications are protected against each other. In addition to the usual protections on Unix-like operating systems, Android also implements a permission system that restricts applications' access to certain system resources based on requests made in a manifest by the developer and approved by the user at install time [7]. Despite these protections, various forms of IPC and system calls allow applications to communicate with each other and with the operating system, providing the possibility of various security vulnerabilities.

Thus, there are three principal interfaces for an ad library: with the rest of the ap-

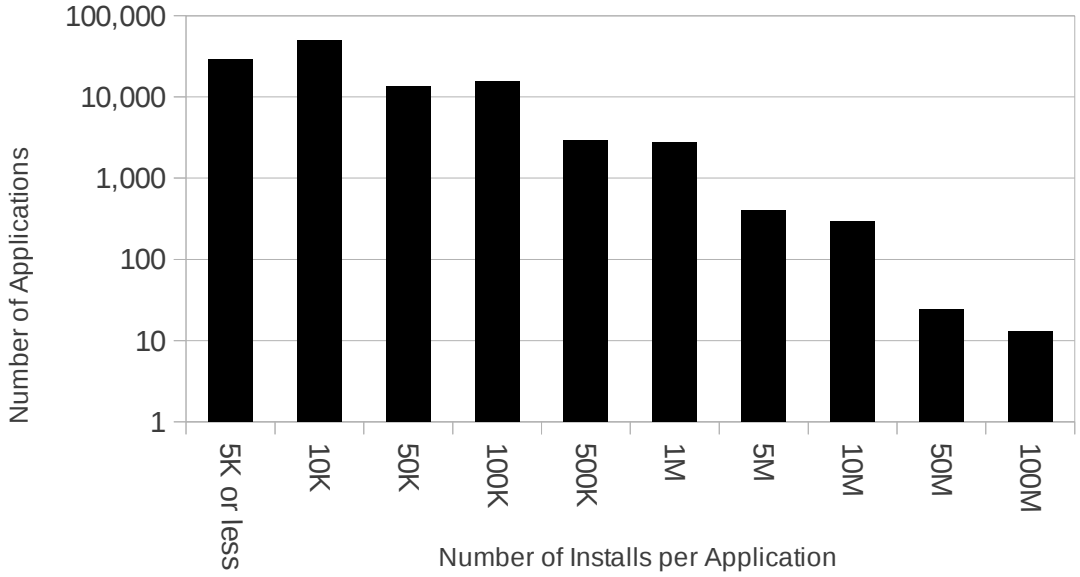

Figure 1.1 : Sample broken down by install count

plication, with Internet resources (principally the ad provider's servers), and with the Android operating system. This paper will explore privacy related behavior regarding the local interfaces.

1.1 Data Collection

In order to conduct this analysis, we needed to obtain an appropriate sample. We downloaded a collection of 114,000 applications from Google Play—the primary Android application store. All were free applications, which are, presumably, the type most likely to contain advertisements.

In order to conduct the downloads, we first scraped the Google Play website to create a database of the available Android applications. This created a data set of 375,000 applications. In addition to the package name, we recorded a number of other features about each application, including information on the number of installs and

the release date for each app. While we were not able to record every Android app through this method, the database did record roughly half of the over 675,000 apps that Google reported on the Google Play overview page at the time [8]. It seems probable that the most popular apps are well represented in the dataset, while the missing apps are among the less popular ones, as the more popular applications are likely to receive a more prominent billing on Google Play.

From within this dataset, we selected the free applications with the highest number of installs, and proceeded to download the applications from Google Play. Because Google Play does not have an interface to directly download applications, it was necessary for us to develop software to register a simulated Android device with Google and use it to download applications. Using the existing google-play-crawler project as a starting point, we built the required software.* Our software registered itself with Google as a Samsung Galaxy SIII running Android API level 16 (JellyBean or Android 4.1), with a carrier ID of "Google." The cell operator and SIM operator codes were set to 310260; the code for T-Mobile. Google Play did not allow us to download applications that were not compatible with that device. In practice, this principally excluded carrier-specific software, for example, applications issued by a mobile carrier for the use of its customers. Likewise, it excluded some device-specific software — either software written by a device manufacturer for its own devices, or software that took advantage of specific hardware features not available on the Samsung Galaxy SIII. Because most Android applications are specified as being forward compatible, we do not believe that our given API level significantly reduced the pool of older applications available for download.

We believe that our policy of downloading all of the most popular applications compatible with our simulated device allowed our sample to reflected the installed app base accurately. Figure 1.1 shows the distribution of the sample according to the lower bounds on the total number of installs for each app, as provided by Google

*https://github.com/Akdeniz/google-play-crawler

Play. Given the caveats noted above, our sample included substantially all of the free applications on Google Play with at least ten thousand installs. We also included a small sampling of applications in the less popular categories in order to enable us to observe any behaviors that were more common in those groups of applications.

1.2 Library Extraction

After collecting the applications, we disassembled them using Gabor Paller's Dedexer.[†] Having manually identified package names for 116[‡] different ad libraries by parsing through the collection of disassembled apps (Included with the libraries were several ad mediation and analytics libraries), we then automatically identified all ad library instances matching our known package names within the entire sample. It should be noted that this method produces an incomplete collection of ad libraries, as there are undoubtedly additional Android ad libraries beyond the 116 that we identified. Of necessity, we omitted any libraries for which we did not have a package name. Out of our data set, we extracted 118,790 ad library instances, an average of approximately one library per app, although some apps contained many libraries, and others contained none.

Our practice of identifying libraries by package name is subject to confusion if a developer uses the package name of an ad library for another piece of software. While the Java package naming scheme, if followed, makes this practically impossible, no mechanism exists to force developers to follow the naming scheme. We did not observe any instances of this occurring. Furthermore, the methods that we use to differentiate library versions (Section 2.1) and our policy of excluding library versions used in fewer than 50 applications (Section 2.2) mean that, even if such code was mistakenly identified as an ad library, any permissions used by that code would not find their way into our chronological results. However, any ad library API calls within the code

[†]http://dedexer.sourceforge.net/

[‡]This number counts com.admob and com.google.ads as separate libraries

mistakenly classified as library code would be omitted from the API call analysis in Chapter 3.

Additionally, our method does not measure any third party library code that may be bundled with an ad library. If an ad library is shipped with code that uses a different package name from the ad library, our system would count that code as part of the application, and not part of the library. This means that we would fail to detect any system API calls made through the third party library, although we would detect any calls from the thrid party library back to the ad library. A non-exhaustive survey of our ad libraries found that, while a small minority do include additional third party libraries, the only permission-protected system API calls accessed through those libraries tend to be those related to internet functionality.

1.2.1 Obfuscation

One difficulty in the process of extracting ad libraries was the use of obfuscation by application developers. Obfuscation changes method and variable names and removes debugging information to make an application harder to reverse engineer. It is frequently combined with optimization software that further modifies the code, removing unused methods and restructuring the code. This means that the same version of the same ad library may have different code when incorporated in different applications. Thankfully, none of the obfuscation software that we have observed changes package names, which means that we are able to identify ad library software even when obfuscated. If software that changes package names is in use by developers, it would result in our failure to identify some ad library instances.

One advertising library, AirPush, recently adopted a scheme to obfuscate package names. Each developer receives a binary with a unique package name when they download the library, presumably with the intent of making it harder for analysts to detect the presence of the Airpush library (which is frequently maligned for putting

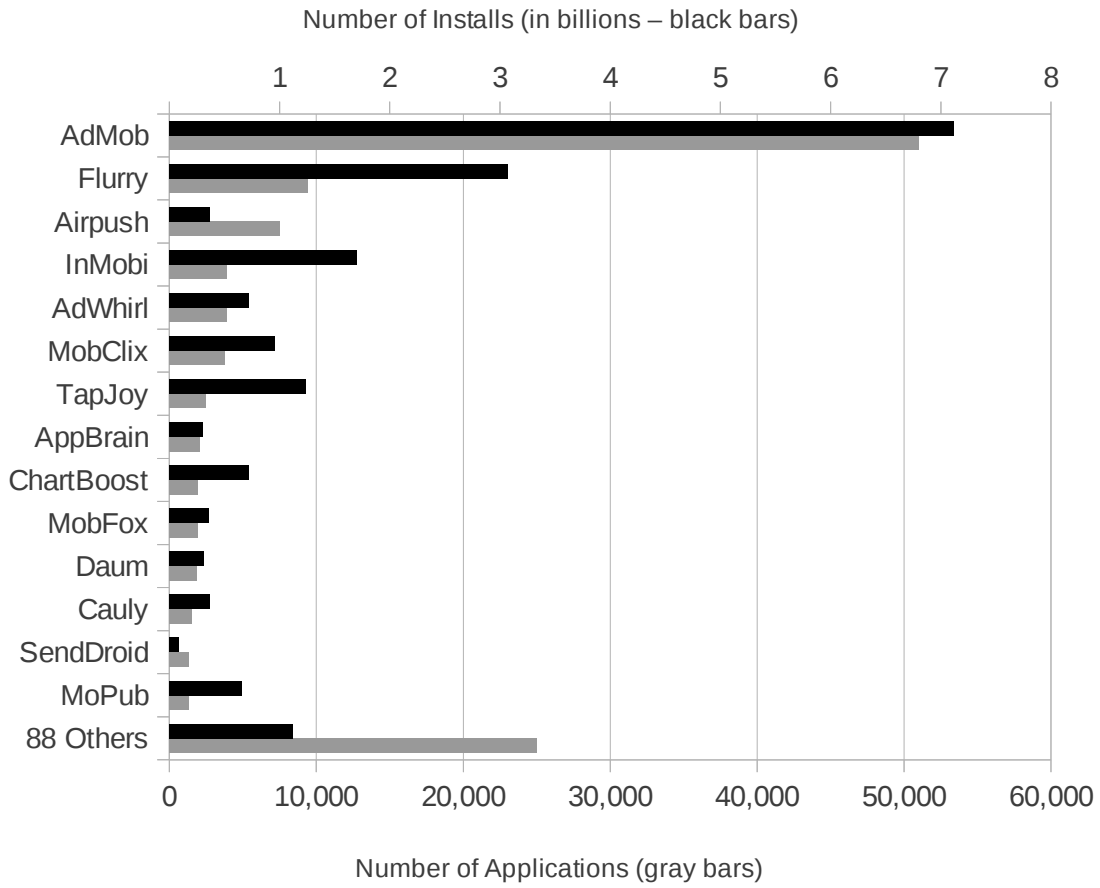

Figure 1.2 : Ad Libraries by Number of Apps

"push ads" into a user's notification bar at random intervals).[§] However, as the library code remains the same, we were able to determine some characteristic features that allowed us to recognize instances of the AirPush library despite the obfuscated name.

1.2.2 Library Market Share

In order to determine which of the libraries to focus on for our analysis, we needed to understand the relative market share of each library. Market share can be measured

[§]A search of Google Play reveals 6 apps whose sole purpose is to detect or remove Airpush from one's device.

in at least two different ways—by the number of apps using a library, and by the cumulative number of installs for a given library across all apps that include it. Figure 1.2 shows the number of apps within our dataset where each library was detected, and an estimate of the number of installs for each library. The top 10 libraries represent 74% of all measured installs. Both package names for AdMob (com.google.ads and com.admob) are combined into the AdMob listing. Numbers for AirPush include both obfuscated and non-obfuscated package names.

Google reports app install data based on ranges with a factor of two or five between the top and the bottom of the range. (e.g., 1,000–5,000 installs or 5,000–10,000 installs.) For our analysis, we chose the low end of the range as a conservative estimate of the number of installs for a given app. Because we do not have more precise install counts, the actual numbers may be greater by as much as a factor of five. Additionally, because the install count only considers apps within our dataset, when other apps are included, the numbers undoubtedly become larger. Nevertheless, these counts give us an effective means to compare the relative popularity of different libraries.

Because our set of apps focused on those apps with higher numbers of installs, this data should accurately reflect the popularity of ad libraries among the most popular apps. It is possible that a different distribution is found among less popular apps. However, as the more popular apps are, by definition, more likely to be found on any given device, we believe that this distribution reflects the distribution of ad libraries likely to be found on a typical user's device.

It is worth noting that while a few libraries (particularly Google's AdMob) have a clearly dominant position in the marketplace, there are a large number of libraries with significant market share. The top 25 libraries have approximately 87% market share, while the top 47 have 95% market share. It is, of course, likely that we failed to identify some ad libraries, pushing these numbers even lower, although we assume that we were able to correctly identify the most popular ones.

It is difficult to estimate the number of libraries that are likely to be found on a "typical" device. Even if we knew the number of apps installed on all devices, the fact that some apps contain many libraries and some contain none means that it would have little relevance. The number of libraries on a user's device is probably highly dependent not only on how many apps he chooses to install, but also on which apps he selects. Anecdotally, certain families of apps, such as live wallpapers and pornographic applications, seem to tend to contain large numbers of ad libraries. Others, such as applications designed to interact with a business with which the device user has an existing relationship, such as banking or airline applications, tend to contain no ad libraries at all.

Given the fact that there are dozens of different ad libraries with significant market share (nearly all of the libraries we surveyed had tens if not hundreds of millions of installs), it becomes clear that the security of a user's data and device depends on the behavior of a large number of principals. While we did not attempt to estimate the number of libraries likely to be found on a typical device, this data shows that it is likely that many libraries are present, each with different security concerns.

1.2.3 Other Library Types

In addition to libraries whose primary purpose is to display ads, this study included two other types of libraries: **analytics** and **mediation** libraries. Because the functionality of these three groups of libraries is similar, and many libraries perform several of these functions, it seemed appropriate to include them in the study. Analytics libraries collect information regarding application performance and user interactions and return it to the developer, usually through a third party vendor. Ad mediation libraries interact with other libraries and ad networks and choose which one to use to display an ad based on factors specified by the developer, such as the current payout from various ad networks. The most common analytics library was Google Analytics, found in 1,256 apps within our sample. The most common ad mediation library was

AdWhirl, found in 3,911 apps. At the same time, many of the other libraries provide one or both of these functions in addition to displaying their own ads.

Chapter 2

The Library – Operating System Interface

The Android operating system limits application privileges through a permission system. Privileges are requested by the application developer in a manifest which is incorporated into the application package and approved by the user when the application is installed. The user may choose either to accept all of the permissions or not install the app—he can not grant or deny specific permissions. The Android API reference lists 130 permissions, which regulate the ability of an app to make various system API calls. The functions covered by a permission range from the clear to the obtuse—for example, the read phone state permission also gives access to a commonly used device identifier.*

Because the granularity of these permissions is at the level of the application, an advertising library necessarily shares in all of the privileges of its hosting application. As an ad library is able to probe for the presence of a particular permission, it is potentially able to make use of permissions in its host app on an opportunistic basis [10]. Of course, an ad library can also require the presence of certain permissions to function.

Concerns have been raised about the interest and ability of users to make sound security judgments on the appropriateness of an app's behavior based on the permission system [11]. Indeed, the permission system has not prevented significant negative publicity regarding Android apps making inappropriate use of personal data, even

*Much of the material in this section was presented at the 2013 MoST workshop in San Francisco, California. [9] Many of the numbers in this section are slightly different than presented at that workshop as the data was re-analyzed with a larger collection of ad library package names.

when the user approved the permissions requested in the manifest [12].

A number of studies have documented that Android ad libraries are able to use application permissions to gain access to sensitive user data [6]. However, these studies have focused on the state of the Android ad ecosystem at a single point in time. In this section, we will extend that work by looking at change over time in library permission usage. By doing so, developments in the ad ecosystem become more explicit.

2.1 Identifying Library Versions

Having extracted the various library instances as described in Section 1.2, we then set out to group the libraries according to version. First, we grouped all libraries according to package name (combining the various obfuscated AirPush package names). We then hashed each library to generate a unique identifier that allowed us to group libraries with the same code base. Because the code base of a given library should be the same across a single version, and vary between versions, we assumed that each hash corresponded with a unique version of a given library. See below for some complications. This method allowed us to identify library type and version without the need for any knowledge of the internals of a given library—i.e. it was not necessary to parse the libraries for version strings, which would have been difficult, and might not have captured minor releases for which the version string remained the same.

We encountered some difficulties in identifying ad library releases by hashing the decompiled code. Due to the way that an Android APK is assembled, virtual register numbers within each library instance varied, based on the other components of the app. We compensated for this by ignoring the actual names of the virtual registers and only considering the presence of a field.

Beyond this, however, additional challenges presented themselves. Because the process of assembling an Android application, especially when obfuscating and optimizing software such as ProGuard is used, may omit or rename portions of the library,

a single library version may vary in size and content after being included in an app.[†] This means that a single library release may produce multiple installed versions, each with different capabilities and signatures. Rather than attempting to correlate which version hashes derived from the same library release, we allowed different hashes to stand independently. While this made it impossible for us to count accurately the total number of releases of a particular ad library, it allowed us to measure permission use and date library releases as desired.

Some negative impacts from this methodology should be noted. Some of the code of the original ad library release may not be included in a given app, including Android system API calls. This means that it is possible for our methodology to under-measure permission usage of a given ad library version release. Additionally, because a library release that becomes fragmented into multiple versions will have fewer apps recorded for each version, the dating becomes less accurate. However, to the extent that our dating is correct, all of the different hashes produced by a single release should be dated to approximately the same period. Because our analysis takes the union of all permissions used by library versions dated to a given month, the permissions available to the most complete version of the library are the ones actually measured.

2.2 Estimating Library Release Dates

Once we had identified library names and versions, it became possible to date them. Because an app could only incorporate a library version that was available when it was built, it was safe to assume that the library version was released before the first app which used it. Our Google Play data included the release date for each app version that we had downloaded. Thus we were able to establish a *terminus ad quem* or latest possible release date for each library version by assigning it a date equal to

[†]http://developer.android.com/tools/help/proguard.html

the release date of the earliest app that made use of that library. In a few cases, it appears that an app may have been updated between the time that we downloaded the catalog data and the time when we downloaded the app (a time span that ranged from hours to weeks). This introduced some variability into this dating process.

Because we needed a sufficient number of data points to accurately date a library version, we discarded library versions that were not included in more than 50 apps. This means that our chronological findings consider only the more widely used library versions, and exclude versions that never had a large market share, or that have been mostly phased out by developers. This includes marginal versions of large libraries, as well as many versions of less popular libraries. Therefore, our final results do not include marginal behavior that was difficult to date, as it was found in only isolated library versions.

In order to verify the accuracy of this approach, we extracted library version codes from the Google AdMob libraries we identified. This enabled us to compare the calculated release dates with the actual release dates as reported by Google [13]. On average, our calculated date was 18 days before the official release, while several major versions were dated within one day of the official release. We also identified several AdMob versions not listed by Google in its release history.[‡] The earlier dates could be a result of beta/testing releases of the APKs, or of inaccuracy in the app release dating provided by Google. For libraries included in fewer applications, our dating approach will necessarily be less precise. However, the demonstrated accuracy of the AdMob dating suggests that our methodology has value in practice, enabling us to collect and date ad library versions released over a long period of time, despite having collected our data at only a single moment.

[‡]This included several versions numbered 1.x, 3.x, and 5.x

2.3 Measuring Permission Usage

We measured permission usage by analyzing the disassembled libraries. A list of system calls requiring permissions was obtained from the API call mappings produced by the PScout tool developed by Au, et al. [14]. This included permissions required for both documented and undocumented Android API calls. We then measured permission usage by collecting the set of all Android API calls made by an application, and using it to generate a list of all of the permissions that that library was able to use. This method of generating permission mappings is similar to that used by Felt, et al. [15] in their work on Android permissions; however, we considered permission usage only within the ad library code, and not within the entire application. Because we only consider static API calls, our analysis is conservative, and would miss API calls invoked via reflection, direct generation of Dalvik bytecode, dynamically downloaded bytecode, etc. Using this method, however, we were able to identify a large number of system calls and the permissions required to make those calls.

At this stage, we were interested in the libraries, themselves, and not in the way they were incorporated into any given application. For this reason, we did not conduct control-flow analysis to identify whether these Android API calls were, in fact, invoked in any particular app. Nor did we correlate the permissions necessary to use the calls with the permissions present in the manifest for any particular app. Rather, by showing the presence of API calls requiring a certain permission, we showed that the library was capable (under some set of circumstances) of making use of that permission. While it would be interesting to see which applications actually contain control flow paths that reach these API calls, we chose to focus on the structure of the libraries, rather then the way they were employed in individual applications.

Some API calls require one permission out of a set of multiple permissions to function. For these, we included all of the permissions in the set as permissions that the library was capable of using. This is because the scope of some permissions (such as the SOCIAL STREAM and CONTACTS permissions) overlaps, and an ad library

would be capable of making an API call if any of those permissions was present in its host application. While the permissions used by some API calls, such as some of the location calls, vary depending on the input given to the call, most API calls require a fixed set of permissions that do not change depending on their input. In cases where the input to the API call influences which permissions are checked, we include all permissions that might be used, admittedly a maximal set.

2.4 Analysis

Having identified library names and versions, dated them, and mapped their permission usage, it then became possible to map the change in permission usage by various ad libraries over time. In doing so, we discovered a pattern of increasing permission usage by applications.

2.4.1 Permission Usage by Library

In order to develop a preliminary understanding of the data, it was useful to consider the number of permissions used by each library. While for many libraries, the number of permissions changed over time, the maximum number of permissions (or sets of equivalent permissions) that we found any library able to use was seventeen, exploitable by the Chinese firm Fractalist.[§] Two libraries (Papaya and MobClix) had versions that could exploit as many as 15 permissions.[¶]

[§]http://www.fractalist.com.cn/. The permissions used included: ACCESS FINE LOCATION, VIBRATE, WRITE HISTORY BOOKMARKS, READ PHONE STATE, KILL BACKGROUND PROCESSES, ACCESS WIFI STATE, ACCESS COARSE LOCATION, WAKE LOCK, GET TASKS, CHANGE WIFI STATE, ACCESS NETWORK STATE, INTERNET, READ HISTORY BOOKMARKS, WRITE SETTINGS, RESTART PACKAGES, GET ACCOUNTS, AND RECORD AUDIO

[¶]MobClix was able to exploit INTERNET, ACCESS NETWORK STATE, WAKE LOCK, GET TASKS, ACCESS FINE LOCATION, ACCESS COARSE LOCATION, CAMERA, READ PHONE STATE, VIBRATE, WRITE SOCIAL STREAM, WRITE CONTACTS, READ SOCIAL STREAM, READ CONTACTS, READ SYNC SETTINGS, and GET ACCOUNTS. Note that multiple sets of permissions can give access to a single

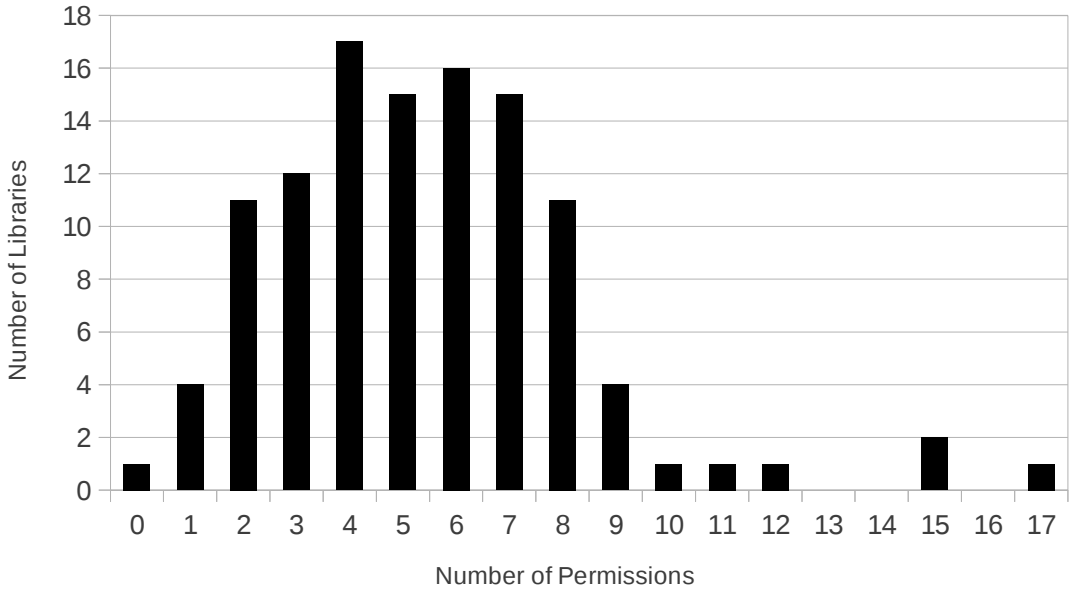

Figure 2.1 : Number of Permissions Usable by Libraries

The mean of the distribution was 5.5 permissions per library, while the median was 5. While the INTERNET permission was required by almost all libraries, one library (MobWin) made use of a separate library (com.qq.taf) to handle communications functionality, and so did not make use of any permissions within the ad library code, making the minimum 0. While some libraries are much more popular than others, previous researchers have generally not calculated the number of installs for each library. We include this per-library data to enable better comparison of our results with theirs.

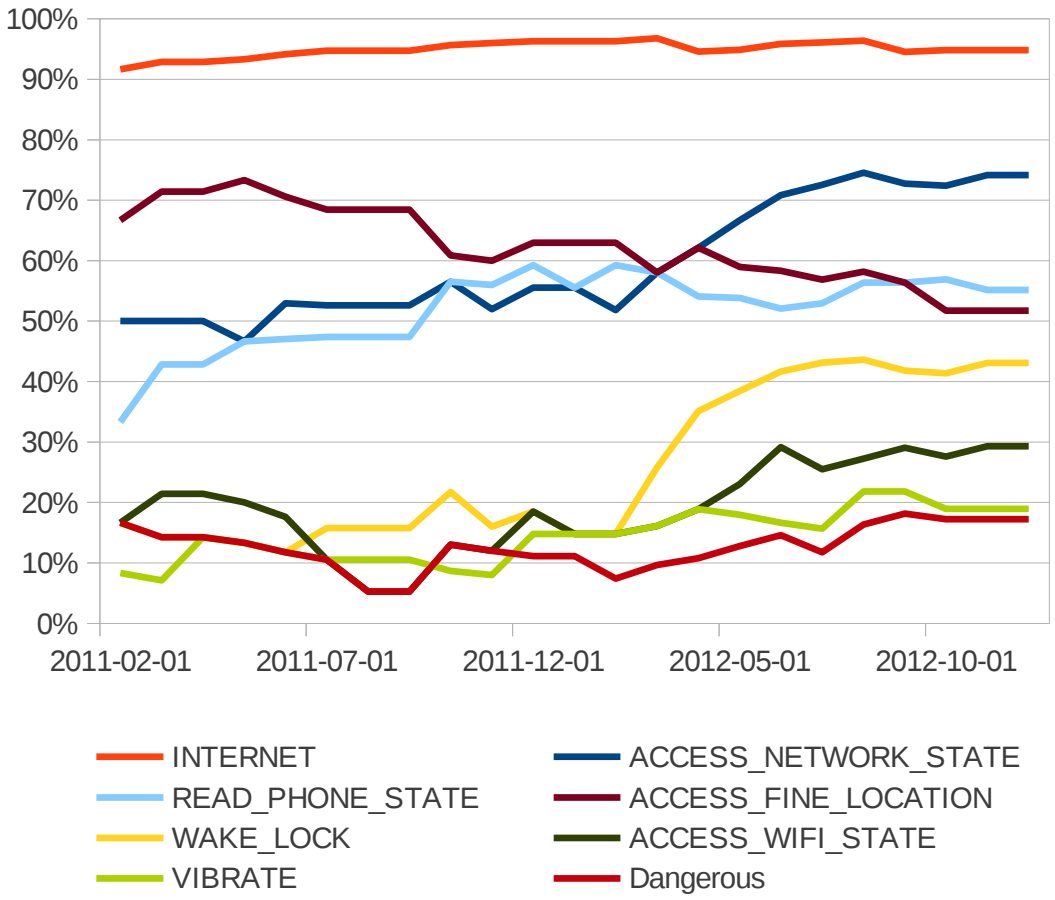

Figure 2.2 : Percent of Libraries Able to Use Permission

2.4.2 Permission Usage over Time

More interesting than the static data is the changing number of permissions that a single library can utilize. By simply examining the number of libraries able to use a single permission, we are able to see which permissions are accessible by the greatest number of libraries. Figure 2.2 represents the percent of known libraries using a given permission for the months where we had data on ten or more libraries. The data becomes more precise for later months where we were able to measure additional ad libraries. As can be seen, most permissions, including the dangerous permissions, show an increasing trend, a potential cause for concern.[‖] A number of less common, but potentially dangerous, permissions are grouped together under the heading of "Dangerous". A library that used one or more of these permissions was categorized as using dangerous permissions.[**]

We categorize permissions as potentially dangerous when they provide access to personal user information, make it possible to directly monitor the user's activity, allow the library to make operations that cost money, or provide access to the system state. For more information, see section 2.4.4. It is important to note that the ability of an ad library to utilize these permissions does not necessarily mean that they were being abused. With appropriate user confirmation, many of them could have legitimate uses. However, as these permissions also enable various sorts of abuse, they

---

API call; for example, the various SOCIAL STREAM and CONTACTS permissions each allow a library to read a user's contacts.

[‖]In the cases where multiple versions of a library dated to a single month, we took the union of all the permissions called for by any version of the library.

[**]These results differ somewhat from our previously published work in that more libraries are included. We also chose to include only library versions for which we had at least 50 installs, allowing us greater certainty in the dating. Furthermore, our previous work relied on measuring permissions usage for a subset of libraries, and propagating that usage to other libraries with the same version. The data presented here includes measurements of the permission usage of every individual library instance, and so should be more precise.

are worthy of special scrutiny.

Although we did not conduct a comprehensive analysis of all API calls requiring permissions, some notes may be made on the most popular permissions. The near universal use of the INTERNET permission corresponds to the ad library's most basic function—downloading and displaying advertisements. The ACCESS NETWORK STATE permission is used by a number of libraries to call android.net.ConnectivityManager.getActiveNetworkInfo, presumably in order to determine the connection over which they are requesting data. The READ PHONE STATE permission is necessary for retrieving a device ID from the telephony manager. Likewise, ACCESS WIFI STATE may be used to obtain the MAC address of the device to help identify it (and its user) uniquely. WAKE LOCK is used by some video playback API calls. VIBRATE is self-explanatory. While ACCESS FINE LOCATION is also self-explanatory, it is worth noting that all library calls surveyed that made use of system calls for location data used calls that could use either ACCESS FINE LOCATION or ACCESS COURSE LOCATION, depending on which permission was available.

2.4.3 Install Weighted Permission Usage

Of course, data that only tracks the number of libraries using a certain permission is somewhat artificial, as some libraries are much more popular than others. In order to understand the behavior of the "average" ad library—although most devices will have a number of ad libraries present in their various applications—we gave each of the libraries a weight based on its market share, and considered the combined data.

Using library popularity data to weight the various libraries, it becomes possible to see the permissions that might be accessible to a "typical" ad library install. Because the nature of our data set made it impossible to reconstruct the historic market share of different ad libraries, we made the assumption that library market share remained constant over the period studied. The results of this analysis is shown in Figure 2.3.

We observe that the general trends are similar to the trends seen when we consid-

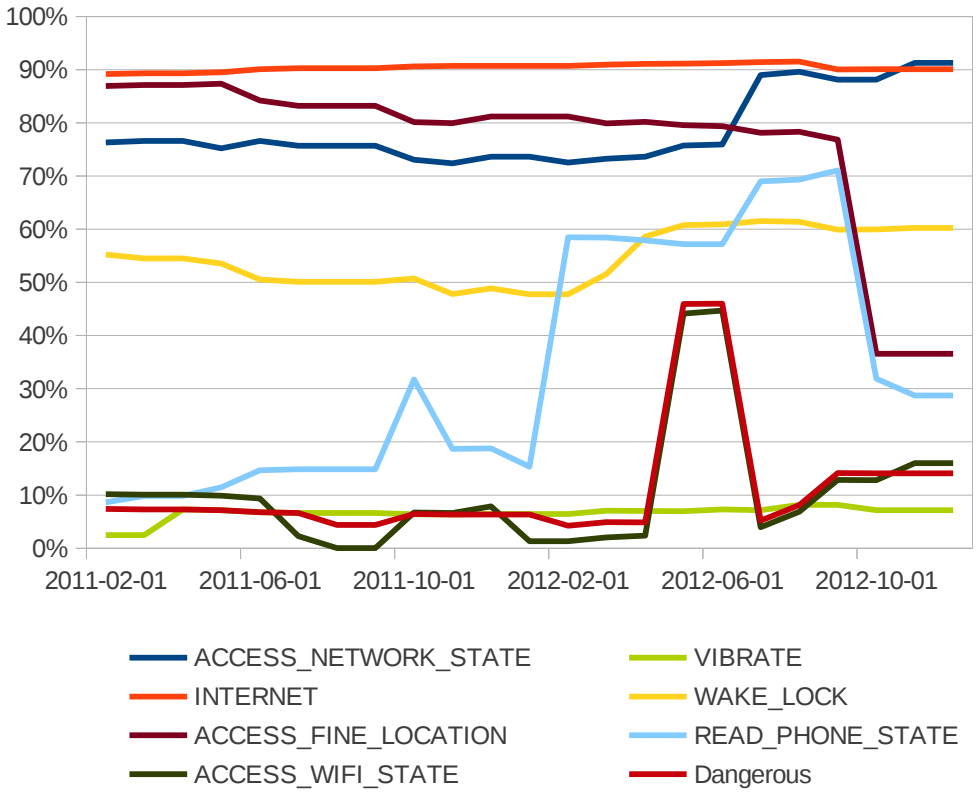

Figure 2.3 : Percent of Library Installs Able to Use Permission

ered individual libraries, but the absolute values for some permissions are different. In particular, ACCESS WIFI STATE, and READ PHONE STATE are used in a lower percentage of installs, as is VIBRATE. WAKE LOCK is significantly more common, as are the permissions used to extract a device ID for tracking: ACCESS FINE LOCATION, and ACCESS NETWORKS STATE.[††]

The disproportionate market share of AdMob results in some disproportionate swings in the data. We detected calls to android.accounts.AccountManager.getAccounts in the mediation code associated with version 6.1.0 of AdMob, resulting in a spike in the percentage of apps using dangerous permissions. The same mediation code makes a call to android.net.wifi.WifiManager.getConnectionInfo, accounting for the spike in the ACCESS WIFI STATE permission. Mediation code in several versions of AdMob call android.telephony.TelephonyManager.getDeviceId, resulting in a period of increased use of that permission. Finally, the removal of location tracking code from AdMob accounts for the significant decline in the ACCESS FINE LOCATION permission in late 2012.

2.4.4 Dangerous Permissions

The subset of permissions which we labeled "Dangerous Permissions" comprises a relatively small set of the permissions used by Android ad libraries, but is still of particular interest. Figure 2.4 shows the install weighted prevalence of the various dangerous permissions among the libraries studied. Although the data is noisy due to the relatively small number of libraries using these permissions, a generally increasing trend can be seen.

A brief review of these permissions gives some indication of their use. GET

[††]Android supplies an install specific ID that can be accessed without any permissions (Settings.Secure.ANDROID_ID). However, this ID may change upon factory reset, or on a rooted phone. Occasional implementation bugs may result in it being missing, or the same across multiple devices. These shortcomings may encourage library designers to seek other IDs.

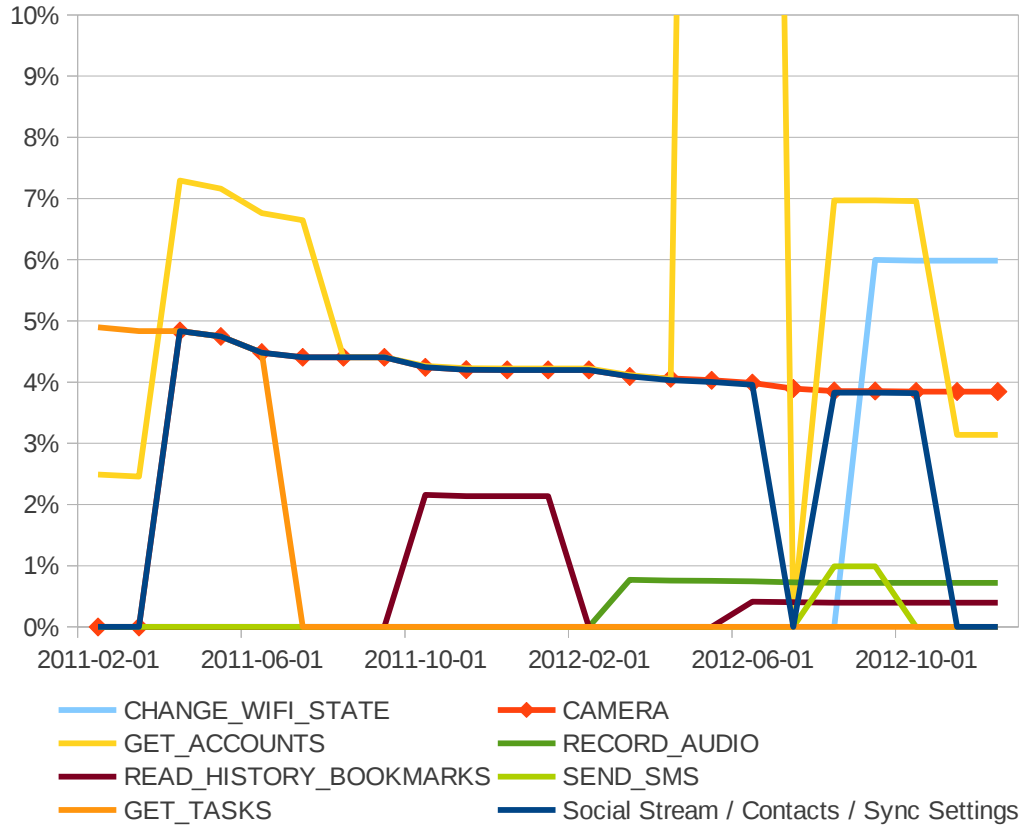

Figure 2.4 : Dangerous Permissions: Percent of Library Installs Able to Use Permission

TASKS refers to processes, not a user to-do list, and seems often to be used for API calls such as android.app.ActivityManager.getRunningTasks and android.app.ActivityManager.getRecentTasks. These may be used in conjunction with PackageManager, which will return the list of installed apps without requiring any permissions, in order to determine which apps are installed on the device, and which are regularly used. This data represents a potential privacy breach, and could represent a security weakness, as knowing the apps installed on a system makes it simple to match them against a list of apps with known vulnerabilities.

The SOCIAL STREAM, CONTACTS, and SYNC SETTINGS permissions are used by calls such as android.provider.ContactsContract.Contacts.getLookupUri to obtain information about the user's contacts. This includes allowing the ad library to read a user's contacts, either for internal use or for transmission to the network's servers.

The GET ACCOUNTS permission allows the library to identify if a user has an account on their device with a provider such as Google or Facebook by making an API call such as android.accounts.AccountManager.getAccountsByType. While full credentials for a user's accounts are not returned, the account name is. For example, on Google accounts, the account name is the user's email address, providing a unique identifier for the individual. Indeed, as these account names can be used to uniquely identify individuals and thus correlate them with data in other databases, it represents a particular privacy threat. Libraries that transfer this information to their servers should be considered to have de-anonymized the user.

The use of the CAMERA permission allows the library to control the camera directly, without the need to send an intent to the camera application. Some libraries advertise the ability for the user to interact with the ad by taking pictures and uploading them to the ad server [16]. SEND SMS, while rarely found in ad libraries, is a particularly problematic permission as sending SMSs can cost the user money. CHANGE WIFI STATE is used by some ad libraries to scan for WiFi access points, which has potential value in determining the user's location. The RECORD AUDIO

permission is likewise used by some ad providers to record audio. Likewise, the READ HISTORY BOOKMARKS is used to read the web browser bookmarks through the android.provider.Browser.getAllBookmarks interface. The privacy issues involved in this are obvious. Although not included in the graph due to its relative scarcity, some ad libraries also make use of the REORDER TASKS permission.

2.4.5 Comparison With Other Datasets

In order to further assess the accuracy of our results, we compared them with a library of 5,000 Android apps retrieved from the Android Market (now Google Play) in May of 2011, as part of the AdSplit project [17]. When analyzed through the same process, these apps yielded 3,374 ad libraries, a slightly smaller ratio than the principal dataset. When the permissions used by those libraries were analyzed, they were substantially the same as the inferred values for May of 2011. The greatest variation was in the VIBRATE permission, which showed up in 15% of libraries in the 2011 sample, while the 2013 data implied a 7% rate. Of course, the 2011 sample undoubtedly contains many apps older than the sample date. Unfortunately, however, metadata to track the app release dates was not collected with the AdSplit sample, so we can not attempt to reconstruct the release dates for the ad libraries found therein.

2.4.6 Overall Trends

Whether considered on a per-library or per-install basis, there is a slow but real growth in the number of permissions used per library over the three and a half years for which we have good data. Figure 2.5 shows the install weighted and unweighted numbers. The weighted values have a positive slope of 0.29 permissions per year, slightly more than the standard error of 0.26. (The slope for the unweighted numbers is 0.30, with a standard error of 0.08). Moving from the general to the particular, however, we can see that there are some permissions that show particular growth, and others that are declining.

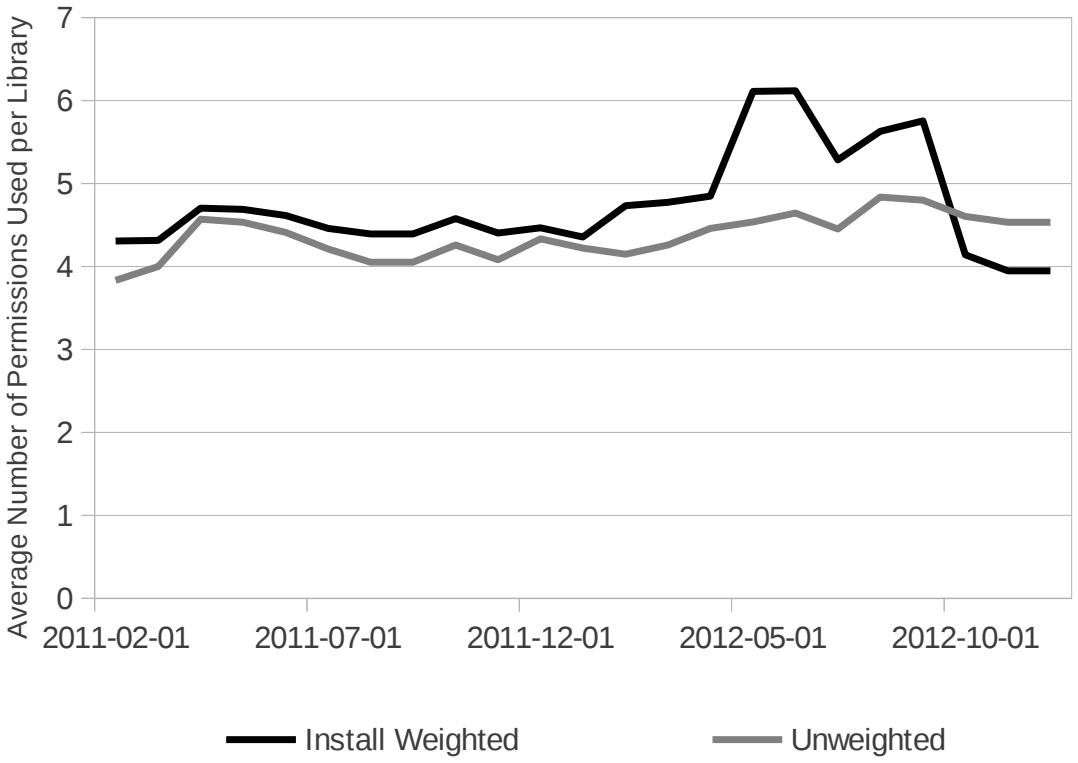

Figure 2.5 : Average Number of Permissions used per Library

Especially strong growth is seen in the WAKE LOCK and ACCESS NETWORK STATE permissions, although almost all permissions show an upward trend. Presumably, this is because ad library developers find that their libraries are more effective when they make use of these additional permissions. Permission growth is not necessarily negative. For example, it would appear that the ACCESS NETWORK STATE permission is primarily used to determine if enough bandwidth is available for bandwidth intensive advertisements; a use that might be considered a service to the user as it should result in smoother functionality and potentially lower data usage over cellular networks.

Unfortunately for users, however, most of the permission creep seems directed not at making the ad libraries work more smoothly and efficiently, but at extracting additional data about the user. Permission growth occurs largely in the permissions needed to uniquely identify the device (and, by extension, the user), such as READ PHONE STATE and ACCESS WIFI STATE, and in permissions that make ads more obtrusive, such as VIBRATE. It is possible that some of the permission growth relates to an increasing ability of ad libraries to incorporate rich media such as video — in particular, the WAKE LOCK permission seems to be related to video API calls. However, most seems to be simply related to the collection of additional data on the user.

When considering permission growth, the permissions identified as dangerous represent a significant area of concern. While the libraries that make use of these permissions comprise a small portion of the total sample, a growing number of libraries do make use of them. The same dynamics seem to be at work as with the more common permissions—a desire for a more intrusive ad experience coupled with a desire to obtain more information regarding the user. MobClix, one of the largest users of permissions, seeks to attract advertisers by advertising to them the ability to access contacts, the user's media library, GPS, calendar, camera, e-mail, and SMS [16].

The principal exception to increasing permission use is a steady decline in the number of libraries making use of the ACCESS FINE LOCATION and ACCESS COURSE

	Missing	Original	Removed	Permissions
EverBadge	81	134	60.5%	2
Hunt Mobile	161	354	45.5%	3
AirPush †	801	1966	40.7%	9
SendDroid	417	1,335	31.2%	8
Waps	57	192	29.7%	8
Tapit	130	458	28.4%	7
AdsMogo	19	67	28.4%	8
Adfonic	186	767	24.3%	5
RevMob	166	706	23.5%	6
Average			11.6%	5.62

Table 2.1 : Libraries found in apps removed from Google Play.

LOCATION permissions. Google has played a significant role by removing location code from its AdMob library (although app developers can still choose to pass location information manually) but this change has also been reflected in a large number of other libraries. It may be that ad providers are able to obtain sufficiently detailed location information from the phone's IP address, or that the commercial value of location information is less than was once expected, but it is also possible that users' privacy concerns have created an environment where app developers prefer to use ad libraries that do not include location code. Overall, however, ad library permission usage remains on an increasing trend.

2.5 App Store Policing

After we collected our original data, Google conducted a purge of some 60,000 apps from Google Play [18]. We were interested in seeing how the apps in our sample fared, and so we re-surveyed the apps to see which had been removed. While many of the

apps re-appeared on Google Play within a matter of days (presumably with changes made to address Google's concerns), we were able to identify 9,980 apps from our sample (almost 10%) that were no longer available. While some of the apps may have been removed by their publisher, it seems likely that many were removed by Google.

After identifying the missing applications, we went back to our original data set to see what ad libraries were used in those apps. We found that apps containing certain libraries were much more likely to be removed than others. Some of the libraries whose apps were disproportionately removed are included in Figure 2.1. The column "Missing" indicates the number of apps containing a library that were removed. "Original" gives the number of apps in the original sample that contained the library. The next column gives the percentage of the total number of apps that were removed. "Permissions" gives the total number of permissions that we identified as being usable by the library.

Further research could determine if the restored versions of these apps lacked these libraries, suggesting that removal of these apps was related to the behavior of their ad libraries. While we have no inside knowledge of what caused Google to select some of these apps for being removed, it's possible that poorly-behaved advertising libraries might be complicit in their host applications' deletion. By looking at the "permissions" column (fig. 2.1), we can see that these advertising libraries seem to use an unusually high number of permissions. However, the fact that some instances of apps using these libraries were deleted while others remain suggests that Google isn't simply banning specific advertising libraries.

Many of the libraries included in Figure 2.1 exhibited additional negative behavior: EverBadge and AirPush place launch icons on the user's home screen; AirPush, MobPartner, and SendDroid all publicize the ability to send advertisements to users in the form of push notifications. Google's ongoing efforts to shape its app store policies suggest that these behaviors were the primary reason for the removal of the apps in question. Neither of these behaviors, however, is regulated by the Android

permission system, and so they were not detected by our analysis. This points to another shortcoming of the permission system—certain behaviors that users might like to regulate are not covered by it.

2.5.1 New Google Play Policies

The effects of this purge demonstrate that there is ongoing development in the behavior of ad libraries on the Android platform, with Google, through its play store, being a significant driver of those changes. Some historical data will further sketch out that dynamic. We will consider Airpush, one of the more agressive advertising vendors, which specializes in "push" advertisements—ads that show up in a user's notification bar even when the app isn't running.

The first sign that Airpush was under pressure to change its policies was the development of "opt-out" infrastructure. Airpush provided a website[‡‡] where users could opt-opt of receiving AirPush ads—by providing their IMEI number—a sure sign that Airpush was using unique device IDs. The next concession was marked by the appearance of appearance of "opt-in" dialogs on apps that included Airpush ads. On first use, users could choose not to receive push notifications, and Airpush could claim that users had accepted this form of advertising. However, even with Airpush's new opt-in policy, their ads generated significant negative feedback from user's. Google appears to have been responding to this feedback when it released a new "Google Play Developer Program Policy," where it banned both icon and notification ads. [19]

Following this clear policy change banning their primary business model, Airpush responded by releasing two new SDKs: one for Google Play, and the other for other platforms. [20] The new SDK for Google Play removed both push ads and icon ads, retaining only more traditional ad formats. The non-Google Play SDK retained push and icon ads. In this way, we see how community perception that a form of advertising is not desirable led to changes in app market behavior that eventually

[‡‡]http://www.airpush.com/optout/

led to the exclusion of libraries that exhibited that functionality. It remains to be seen whether similar concerns will further restrict the ability of ad libraries to gather private information.

The continued provision of an AirPush SDK with support for push advertisements for off-market users illustrates another reality. The extent that market operators are able to control the functionality of ad libraries is a function of their own market share. To the extent that users turn to other sources for apps, those restrictions no longer apply. However, the ability of market operators to ensure the quality and security of their offerings also gives them a competitive advantage over other sources.

2.6 Conclusion

The Android environment was carefully designed to provide security by separating applications from each other and limiting their permissions. It also provides strong infrastructure for patching and updating applications, either in response to security flaws or to fix bugs and provide new features. However, the development of ad libraries exposes a significant weakness in that design. Because an ad library is bundled within an application, it operates with the same permissions as the application. There is also no way for the library developer to push updates, aside from releasing a new library version and encouraging developers to adopt it. It was this flaw that made it possible for us to sample older library versions by looking at apps currently released for download.

Our research showed that ad libraries are increasingly taking advantage of permissions that may be requested by the host application. With the exception of location data, we observed a steady increase in the number of permissions ad libraries were capable of utilizing. Particularly disturbing is the growth in the usage of various dangerous permissions that pose particular privacy risks. While these permissions are used by only a small number of ad libraries, their use is steadily growing, and should be considered with particular scrutiny. Given the increasing intrusiveness of

advertising libraries on the Android platform, the importance of community norms to protect user privacy becomes ever clearer. Without an effective response from Google, via its Play Store, the only viable alternative to protect users' privacy would seem to be government regulation.

Chapter 3

The Library-Application interface

A growing concern with advertisement libraries on Android is their ability to exfiltrate personal information from their host applications. In addition to libraries' abilities to extract private information from the system, advertising libraries also include APIs through which a host application can deliberately leak private information about the user. We examined this interface by reconstructing the APIs for 116 ad libraries used in the corpus, and studying how the privacy leaking APIs from the top 20 ad libraries are used by the 64,000 applications in which they are included. Notably, we have found that app popularity correlates with privacy leakage; the marginal increase in advertising revenue, multiplied over a larger user base, seems to incentivize these app vendors to violate their users' privacy.*

3.1 Introduction

A great deal of recent research has focused on the relationship between advertising libraries and the Android operating system, with a particular focus on the use of Android API calls protected by permissions [6, 9, 10]. However, very little attention has been paid to the other interface of ad libraries: the API that they use to interact with their host application. This interface represents a significant privacy concern, because host applications have access to confidential user data that extends beyond the information that they request from the operating system. Indeed, applications may have access to a great deal of confidential data, obtained through system calls, direct user

*Much of the material in this chapter was previously presented at the SPSM conference as: A Case of Collusion: A Study of the Interface Between Ad Libraries and their Apps [21]

input, and social network APIs, as well as data shared by other applications, on the device or in the cloud.

Generally speaking, the application developer's consent is necessary for an ad library to access this trove of information. Barring creative run-time inspection by the library of the host application's data structures, it is up to the developer to pass data to the ad library through the ad library API. However, developers have every incentive to hand personal data to advertising agencies, as it has the potential to increase advertising revenue for their applications. In this study, we set out to understand these interfaces, and the extent to which developers make use of them.

3.2 Methodology

Making use of the collection of applications and ad libraries described in Section 1.2, we took a different approach, focusing on the application code instead of the library code. Because we knew the package names for our libraries, we were able to parse through our applications and identify all API calls made from any of our application to one of its ad libraries.

By assembling the set of all observed API calls, we were then able to reconstruct the API for each ad library, generating a complete record of all API calls that were actually used within our dataset. We were also able to track the frequency with which each API call is used. In extracting data from the apps, we were able to retrieve not only method names, but also parameter and return types, giving us a profile of the interface between the application and its libraries. For this analysis, we did not consider different versions of the libraries, choosing to differentiate libraries at the level of package name. This means that if a certain API call was only included in some versions of a library, we still included it in our API reference. When we measure frequency of use, we measure it across all versions of a library, even if a given call might not be included in some versions.

In theory, there are other ways that an app can communicate with its ad library

aside from API calls—direct manipulation of class variables, shared memory, calls to the app from the library, etc.—but the need to interact with a variety of developers effectively prohibits the use of more indirect techniques. Nonetheless, our detection of interactions based on API calls does not capture any such interactions, and thus may only study a subset of the information flowing from applications to their ad libraries.

In addition to the library API, the Android platform provides one additional interface between the application and the ad library—the Android XML layout files included in the application. These layout files allow the developer to instruct the library to display an ad and to pass various parameters. However, because the parameters are static and fixed at build time, their ability to pass significant personal information is limited. Nonetheless, to the extent that the app developer is able to predict the demographic of his users, the XML layout file does present a potential point of information leakage. Indeed, it seems that some ad library designers have sought to exploit this interface: at least one ad library, Jumptap, allows developers to embed demographic information, including age, gender, household income, and postal code, in the static XML files for the application [22] In this paper, we do not analyze information passed through the layout XML files.

3.2.1 Obfuscation

The use of obfuscation by application developers created difficulties in reconstructing ad library APIs in much the same way as it had done in studying the libraries, themselves. When obfuscation and optimization software was used, the ad libraries were re-written, often changing the method names. This, in turn, changed the calls to the ad library from the application. Because package names were not generally obfuscated, we were still able to detect these calls and their parameters, but had lost the method names, preventing us from correlating them with the same calls in un-obfuscated applications. Thus, we were unable to count the number of privacy-related ad library API calls in obfuscated applications. An analysis of calls to AdMob

indicated that approximately 5% of API calls were obfuscated in this way. Given this, our findings represent a lower bound on the number of applications leaking privacy related information through ad library API calls.

3.2.2 API Analysis

Once we had reconstructed the API for each ad library, we set out to identify privacy related API calls. We did this by a manual inspection of each API call, using the method name and parameters to form an initial evaluation of whether the API call constituted a privacy risk. In cases of uncertainty, we turned to official API references and sample code available on the Internet in order to form a better assessment. While it would be trivial for a devious library developer to choose method names that would confound this sort of analysis, of necessity the API is designed to be accessible to application developers, and so such a choice would make it difficult for developers to use the API, causing it to fail in its primary purpose. Anecdotally, looking at the API calls that we identified, there was no indication that library designers sought to present method names that hid their true function.

We conducted this analysis on the top 20 libraries, which together account for 84% of all installs. 64,000 apps within our corpus contained at least one of these libraries. By excluding the smaller libraries, we necessarily failed to detect some API calls that present the potential for privacy leaks, as well as some of the applications that made use of those APIs.

In order to understand how these API calls are used by applications, we then counted the privacy related ad library API calls made by each app, recording the number of apps making use of each type of call. This enabled us to determine the percentage of apps passing personal information directly to ad libraries, and so to quantify the scale of the privacy concern represented by these libraries. Our analysis has its limits of course:

Key/value maps. Some ad libraries allow passing general key/value pairings (e.g.,

Java "map" objects), allowing a variety of different privacy-relevant items to be passed at once. We did not implement the static analysis and analysis over Dalvik bytecode that would be necessary to track these values. All we see is that the general-purpose call is made, not what keys are included. In our results, we report use of these APIs in a separate category.

Ad mediation libraries. Another complication arises from ad mediation libraries, which switch among different advertisers. Since we're interested in leaks from applications to ads, and not from mediation libraries to other ads, we excluded calls from one ad library to another. Note that this methodology also excludes information paths where one ad library retrieves information from the underlying Android system (for example, by querying the device's location) and then passes it to another ad library.

3.3 Results

For developers, the multiplicity of ad libraries on the Android platform provides options, allowing them to select libraries based on privacy concerns as well as functionality and revenue potential. However, for users, who do not choose the ad libraries in their apps, this represents expanded vulnerability, as the presence of any library that exfiltrates personal data means that the data is potentially compromised. We show that many libraries have the potential to do exactly that.

3.3.1 Ad Library APIs

Our reconstruction of the APIs for 116 ad libraries represents a useful achievement in the study of the behavior of these ambiguous pieces of software. While many APIs have publicly available documentation, many others restrict access to their API documentation to registered developers. Additionally, our method of extracting calls actually used by applications allows us to understand the "working API"—those calls, documented or not, that are actually used in applications, together with their popularity. While app developers' use of obfuscation software that modifies method

names makes it somewhat difficult to count accurately the number of API calls in a given library, it is worth noting that many have a very simple API, with 17 of 116 libraries having 10 or fewer API calls. Promotional literature for ad libraries often stresses the ease of incorporating the library into an application, which may account, in part, for the small APIs.

The most common API calls for nearly all libraries are, as might be expected, calls related to laying out and requesting advertising content. To take AdMob as an example, the constructor for the AdRequest object is the most frequently called method. After this, the loadAd method for an AdView follows closely, together with the constructor for an AdView. These are followed by various layout and lifecycle methods. Other libraries have a similar distribution of calls.

3.3.2 Privacy related API calls

This study is not concerned with the most common calls, but those that represent potential privacy leaks. Of the top 20 libraries, 11 include calls in their APIs that have the potential to leak user data. The sort of data leaked included everything from location to household income. For a list of all factors identified, see Table 3.1.

Most of these categories are self-explanatory, but a few deserve additional detail. The category labeled "Arbitrary Data" refers to calls that allow the developer to send arbitrary data to himself. These calls are common in libraries with analytics functions, and are presumably intended to be used to transfer information regarding the usage and performance of the application. They could, however, transfer any sort of data, and so represent a potential privacy leak. However, because a developer can send information to an arbitrary server without the need for an advertising library, their presence in advertising and analytics libraries does not represent an additional threat.

The category "Age" encompasses both API calls that give the user's age in years and API calls that transmit the user's exact birth date. While both have similar

Classification	Percent of Apps	Percent of Installs
Arbitrary Data	3.06%	9.13%
Keywords	2.50%	5.87%
Gender	2.03%	3.06%
Location	1.64%	3.38%
Age	1.50%	2.66%
Multiple Factors	0.50%	1.99%
Postal Code	0.42%	0.49%
Enable Location	0.34%	0.32%
Income	0.12%	0.07%
Interests	0.01%	0.01%
Area Code	0.01%	0.01%
Country	0.01%	0.12%
Education	0.01%	0.01%
Ethnicity	0.00%	0.00%
Name	0.00%	0.00%
E-Mail	0.00%	0.00%

Table 3.1 : Percentage of apps making a call to a top 20 library

value in targeting advertisements, a user's exact birth date provides a much greater risk of de-anonymization.

The category labeled "Multiple Factors," on the other hand, represents API calls that permit the sending of various pieces of demographic data to the ad agency. These calls generally accept a key/value store, such as a Java map, which can contain a variety of factors, generally similar to the ones that can be passed through individual calls. While we did not analyze which factors are passed through these calls by which applications, the relative infrequency of their use makes such counts unnecessary for understanding the overall data flow through the ad library APIs.

The category "Enable Location" is also slightly different from the other calls. Rather than passing information to the library, it authorizes the library to make (or not make) system calls to collect location data directly. Thus, the presence of this call controls a location function that is present in the library.

As can be seen, the privacy related data that is leaked through ad library API calls is data that might be useful in targeting advertising. A few categories, such as postal code and area code, seem to be carry-overs from the world of print and phone marketing, and perhaps point to an advertising model where information associated with a user, outside of the phone through some other business relationship, is being send through the phone to the advertising agency.

Without access to internal data from the ad agencies, it is difficult to know to what extent this information is used to uniquely identify users or to link their identities with external databases. However, previous research has shown that as of the year 2000, 63% of the US population could be uniquely identified by gender, zip code, and date of birth, without reference to other factors [23]. Given that many of these libraries collect other identifying factors beyond these basic demographic indicators, it would appear that some are collecting a sufficient amount of data to de-anonymize users.

When one remembers that libraries can also collect information directly from the

underlying Android system, and that ad agencies can use unique device identifiers to track users from one application to another, it becomes clear that the amount of data collected on an individual user could be significant. When combined, this could again be used to de-anonymize users.

It is also worth noting that most of these calls transfer information that the libraries could not themselves obtain via privileged Android calls. Thus, they extend the dataset that an ad agency is able to build around a user by bringing in information from other sources. While this may be information directly entered by the user into the application, it can also be information from social networking sites or other cloud data sources that the user authorizes the application to access.

3.3.3 Number of applications making privacy related API calls

Table 3.1 shows the percentage of the total universe of apps that make a call in a given category to one or more of the top 20 libraries. It can be seen that certain API calls are called much more frequently than others. Aside from the "Arbitrary Data" category, whose seriousness depends on the way that individual developers choose to use it, the most common category is keywords, which may relate to user activity inside an app, but is not fundamentally different from the sort of personal data habitually collected in web advertising. The next most common category is gender, which, together with age, provides demographic information without being, by itself, personally identifying (It should be noted, however, that the age category includes API calls that transmit birth dates, which go a long way towards uniquely identifying an individual.) Location information is also frequently passed to ad libraries, although many libraries also collect this information directly from the Android operating system [9].

The only other pieces of data that are collected with any frequency are postal code and income. While the value of income for ad targeting is obvious, it is interesting to speculate on the value of post code information in situations where the user's current location is known. It undoubtedly allows the ad agency to link the user to a great

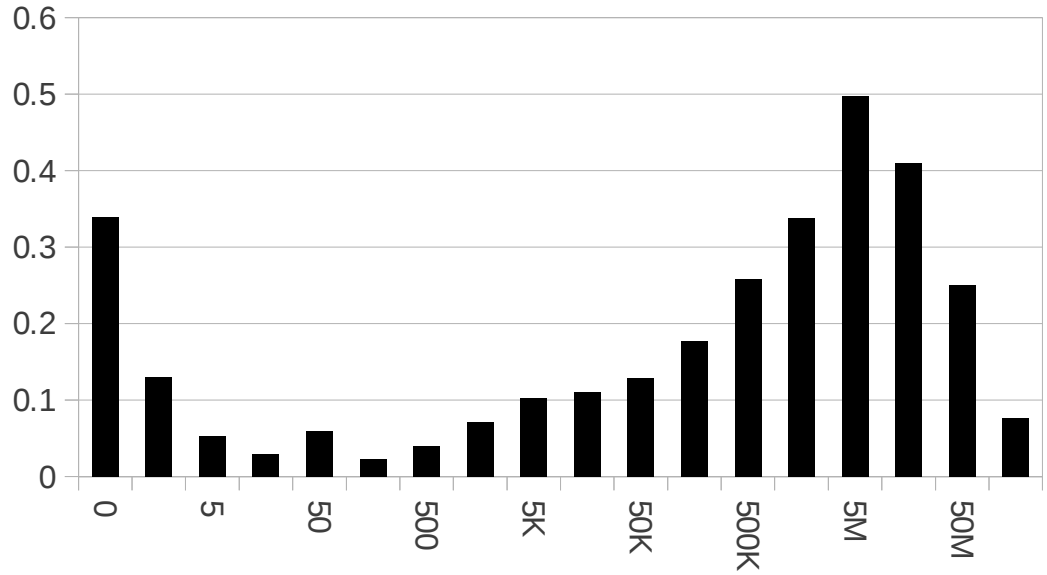

Figure 3.1 : Average number of privacy related API calls per app by number of installs

deal of post code specific demographic information, but it also provides a significant vector which, when combined with others, could be used to de-anonymize users.

There are several pieces of data which are almost never shared with ad libraries, some of which, such as name and e-mail, are nearly sufficient by themselves to de-anonymize users. The scarcity of the use of these API calls may be attributed to their presence in only one less-popular library, as well as the rarity of their invocation even in apps that include that library. Perhaps this represents a sense of ethics on the part of library and app developers, or perhaps it simply represents a pragmatic choice regarding the usefulness of that information combined with the costs of collecting it.

3.3.4 Popularity of applications making privacy related API calls

An additional question that comes to mind when considering the use of privacy-related API calls concerns the popularity of applications using these calls. Are these rarely-installed rogue applications, or do the represent the “mainstream” of Android apps—those with a large number of installs?

Figure 3.1 shows the number of unique privacy related calls per app. The apps are divided into the usage categories given by Google Play, with the lower bounds of the category given for each app. Thus, we assume 5,000 installs for an app that Google lists as having 5,000 to 10,000 installs. If a single app makes multiple calls of the same type to the same library, each set of calls is only counted once. However, calls sending different sorts of information to the same library, or calls sending the same sort of information to different libraries are counted separately.

As can be seen from Figure 3.1, there is a bimodal distribution, where the least and most popular apps are leaking the most privacy-relevant data to ad libraries. These leaks peak at nearly 0.5 calls per app in apps with 5 million to 10 million installs. While our sample contains only a small number of apps with fewer than 5,000 installs, making the data for the lower end of the spectrum less accurate, it contains most of the free applications with at least 10,000 installs, providing very accurate data for those categories.

It is reasonable to assume that the applications with the largest number of installs are also the applications which have received the largest amount of development resources. They are also the applications whose developers have the most to gain from marginal increases in ad revenue for user. Given these facts, it is perhaps not surprising that these applications are the ones which are most likely to contain privacy related API calls. However, it also indicates that the average app install is more likely to contain these calls than might be otherwise expected, as the more popular apps are proportionally more likely to be installed on any given device.

3.3.5 Correlation with permission usage

As discussed in Chapter 2, ad libraries can obtain sensitive information from system calls as well as from their own API calls. We desired to know whether there was any correlation between these two forms of intrusive behavior. Is it the case that libraries that use large numbers of permissions also seek to gather personal information through

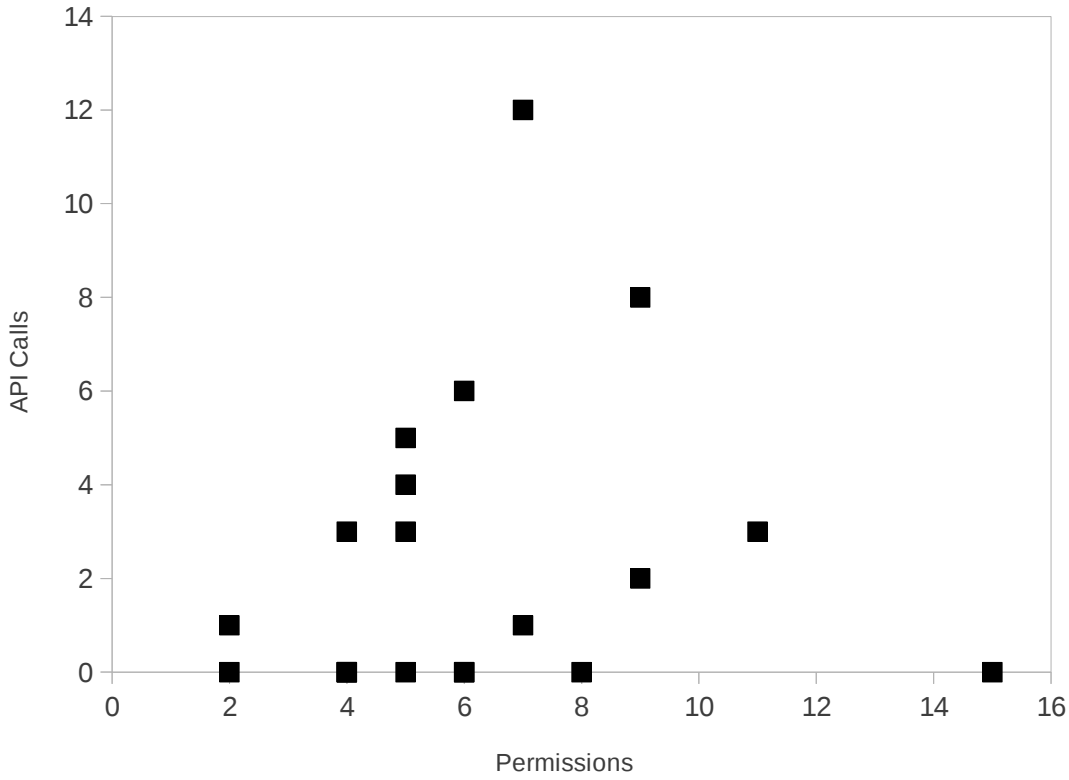

Figure 3.2 : Advertising libraries, showing correlation between permission usage and the presence of privacy related API calls

developer accessible API calls, or does the use of one sort of information replace the need for the other?

Figure 3.2 compares the number of API calls and the maximal number of permissions used by any version of a given library. We found that there was a very low correlation coefficient (0.14) between the two vectors, although all libraries with a large number of API calls did make use of at least a moderate number of permissions. Some libraries made abundant use of both interfaces, while others primarily made use of one or the other. The decision to make use of one interface or the other appears to have been based on independent decisions by the developers.

3.3.6 Library Comparison

Looking at our data in greater detail, we can examine the varying behaviors of the different libraries. Table 3.2 shows the percentage of apps using a given library that make use of the various API calls. It can immediately be seen which libraries provide API calls allowing developers to share privacy related data. Some offer a much broader interface than others. Of course, the cooperation of the developer is necessary for the ad agency to receive this information. Our results show that most developers either do not have access to this information or choose not to share it.

It is worth noting that nine out of the top twenty libraries—that is to say, nearly half—do not appear to provide any API for developers to pass personal data to the library. While these libraries may present other privacy or security related concerns, they present no risks in this particular area. We can see that privacy-related API calls are invoked more frequently for some libraries. While we do not have any data that would directly explain this phenomenon, some agencies may do more to encourage developers to provide personal information. There might also be some correlation between developers who use certain libraries and those who desire to leak personal data.

It is important to note that we are only counting calls made directly from the

	AdMob	Flurry	AirPush	InMobi	AdWhirl	MobClix	TapJoy	AppBrain	ChartBoost	Daum	Cauly	SendDroid	MoPub	Google Analytics	SendDroid	JumpTap	AppLovin	Mediba	Ineractive	GreyStripe
Name																	0.1			
Location	2.7	4.8	0.0	1.3									4.8				0.2			
Gender	2.5	3.4		2.4	15.6					0.0						13.8	0.5			
Age	1.5	3.4		1.8	13.7					0.1						13.8	0.5			
Education				0.2																
Ethnicity				0.1																
Income				0.1												12.8				
Postal Code	0.0[a]			0.4	9.1											9.9				
Area Code				0.2																
Country																0.6	0.3			
Interests				0.3													0.1			
E-Mail																	0.1			
Keywords	3.2			0.6	27.7								12.7				0.3			
Arbitrary Data[b]		31.1												47.5						
Multiple Factors			0.0	1.7							34.9									
Enable Location				0.5						1.4			0.5			14.3				

Table 3.2 : Top 20 Libraries: Percentage of apps making privacy related API calls

[a]The older com.admob package included this feature. The newer com.google.ads does not.

[b]To the app developer

application, and not calls that are made from one library to another. This may account for the much higher rate that these API calls are made to AdWhirl. AdWhirl is an ad mediation library, which allows a developer to incorporate various ad libraries in an application, and display ads from one of the networks based on factors such as ad availability and value. In this way, information passed to AdWhirl may be passed, in turn, to the other libraries called by AdWhirl.

3.4 Discussion

Although we have determined that the percentage of applications that pass personal information to ad libraries is relatively low, the impact of this interface may be greater than one might first think. Once the information is transmitted to the ad agency, it does not disappear, but is presumably logged in a database. Because ad libraries often transmit a unique device ID, this information can then be correlated to information transmitted by other applications using the same ad library, and then be used in targeting ads to the user. To the extent that the user's identity may be revealed by the ad libraries, this data can then be correlated to other demographic databases, not only allowing data from those databases to be used for targeting ads on the mobile device, but also allowing those databases to be updated with data obtained from the mobile libraries (for example, the location information that may be obtained through the library, the system, or the user's IP address).

One factor that is not addressed in this study, and that must be left to future work, is the question of the accuracy of the data provided by app developers to ad library vendors. They might infer certain demographic characteristics of the user from the nature of the application, itself—for example, a children's game could have a hard-coded age value that it provides to the advertising library. There might also be instances where applications entirely falsify keywords or demographic information in the hopes of obtaining greater revenue from the ad agencies. We have not, however, investigated this possibility.

Much existing research into mitigating the privacy risk of Android ad libraries has focused on restricting the ability of ad libraries to make use of application permissions [17, 24]. It is worth noting that these techniques do not address the privacy risks posed by data passed through the library API. On the other hand, approaches that use static analysis to identify privacy and security concerns can easily detect this behavior, as do methods that look for malicious behavior on the level of the application, without considering the distinction between application and library [25, 26]. These methods can be automated and applied on a large scale [27].

Chapter 4

Conclusion

4.1 Related Work

A number of authors have analyzed Android applications to assess security vulnerabilities. For example, Enck et al. [7] conduct a broad study of security vulnerabilities in 1,100 disassembled apps. Gibler et al. [27] use static analysis to identify potential leaks of personal information in Android applications.

A significant amount of research has been conducted into Android ad libraries. Stevens, et al. [10] analyze 13 ad libraries, comparing their permission usage with their documented permission requirements, and show privacy weaknesses related to user tracking as well as system APIs exposed to Javascript exploits. Grace, et al. [6] conduct a similar analysis on 100 ad libraries selected from 100,000 apps. They likewise identify permission usage and other security risks, such as the potential to load and execute arbitrary bytecode through the ad interface. Neither of these papers, however, consider the development of ad libraries over time, or even make reference to variations among different versions of the same library.

Additional research has targeted other aspects of the Android ad ecosystem. The data traffic [28] and energy [29] usage involved in displaying ads has been examined. Leontiadis, et al. [4] look at the other side of the issue, considering the importance of advertising, including access to personal information, in maintaining the Android app ecosystem, while examining various ways of regulating and monitoring an Android application's access to personal data. Vallina Rodriguez et al. [30] study mobile ads through an analysis of network traffic on a major European mobile carrier, considering energy implications and ways of optimizing the ad delivery system.

Significant work has also been done on resolving the security issues involved in the current Android ad model. One of the most prominent approaches has been to advocate separating ads from applications, enabling isolation of permissions and data [17] [24]. In particular, the AdSplit framework considers the value of providing advertisers with protection against fraudulent behavior on the part of app developers, while at the same time separating apps and ad libraries, allowing each to function with separate permissions.

4.2 Future Work

Much remains to be done in tracking the development of the Android ad ecosystem. Ongoing sampling of Google Play would allow the construction of a dataset of apps providing greater detail for diachronic studies. Similar sampling of third party app stores would provide an additional longitudinal dataset of significant value.

Additionally, more precise measurement of the available data may yield interesting insights. Static analysis could reveal any changes in the conditions under which the various API calls are made, and help to determine if they are explicitly triggered by the user, an important measurement for privacy questions. It is conceivable that, while apps are capable of making more sensitive API calls, they are also making them in a more responsible fashion.

On the technical front, we expect more opportunities to study app behavior in the wild, either through static analysis, dynamic emulation, or on instrumented handsets. Likewise, our work here has focused on apps from Google Play, which must pass Google's deliberately vague standards. Apps from other platforms might behave worse. Also of interest are new technical means for users to control the behavior of their apps, such as CyanogenMod's experimental new Privacy Guard Manager*, which allows users to restrict apps access to location, contacts, and so forth, regardless of those apps' required permissions.

*https://plus.google.com/+CyanogenMod/posts/86LLXrDpVWY

On the policy front, what level of privacy (required notifications, explicit consent) should users expect on their mobile devices? What standards should govern ad agencies and application developers seeking to collect personal data and who should administer them? Additionally, awareness of the risks of ad libraries should not cause us to overlook their contribution to the Android ecosystem [4]. Unfortunately, there is currently very little in the way of norms or standards to protect this information. While the Google Play Developer Distribution Agreement does allow Google to remove applications that are "deemed to be ... spyware," the only mentions of privacy regard Google's ability to collect data on the developers, not on the developers' respect for user privacy [31]. It is clear that standards regarding user privacy are needed, whether they come from entities that manage application stores (such as Google) or from some other source.

Chapter 5

Conclusion

Android advertising forms an important part of the platform ecosystem, supporting the development of hundreds of thousands of free apps relied upon daily by hundreds of millions of users. Nonetheless, those free apps come at a price, not only in terms of screen real estate, user experience, and battery life, but also in terms of user privacy. We have shown that ad libraries have access to significant quantities of personal data through two interfaces: system calls to the Android operating system and API calls by their host applications.

The use of Android system calls by ad libraries shows an increasing trend, with a number of API calls, including those which expose personal user information, showing a steady increase in use. This is only partially balanced by a decline in the use of API calls that collect user location. While the behavior of different ad libraries varies considerably, users have little control over which ad libraries may be present on their system, and thus must assume worst-case behavior.

In addition to use of the Android system APIs, library API calls that allow developers to expose personal information are present in most of the top 20 ad libraries. They are generally designed to allow application developers to transmit demographic or targeting information that might be used to target ads at a given user. While few libraries include API calls that would be sufficient, by themselves, to de-anonymize application users, in several occasions, the data provided by some combination of these calls could be sufficient to correlate the user with a real world identity.

Most mobile applications do not make use of these privacy related API calls. However, the number which do choose to include these calls is not negligible. As

such calls are more common in more popular applications, and as ad agencies have the ability to correlate data from multiple applications, a significant portion of users have some personal data exposed through these API calls.

Unfortunately, users have no way to know of, approve, or block any transfer of information from applications to ad libraries and agencies. Because application developers have an incentive to maximize ad revenue, they have a corresponding incentive to leak private user data, with few likely consequences. If users are to have an expectation of privacy for data accessible from their mobile devices, some mechanism to report and manage these data flows is needed.

The primary mechanism that is currently in place consists of regulation by app store owners. We have seen that this has been effective in curtailing certain undesirable behavior on the part of ad libraries, but it remains uncertain to what extent app store owners, particularly Google with respect to Google Play, will protect users' privacy.

Bibliography

[1] Strategy Analytics, "Android & Apple iOS capture a record 92 percent share of global smartphone shipments in Q4 2012," 2013, http://blogs.strategyanalytics.com/WSS/post/2013/01/28/Android-and-Apple-iOS-Capture-a-Record-92-Percent-Share-of-Global-Smartphone-Shipments-in-Q4-2012.aspx.

[2] G. J. Spriensma, *Do Free Apps Really Account For 89% Of All Downloads?*, Distimo, Sep. 2012, http://www.distimo.com/blog/2012_09_do-free-apps-really-account-for-89-of-all-downloads/.

[3] F. Y. Rashid, "iOS apps just as intrusive as Android apps," *Security Week*, Jul. 2013, http://www.securityweek.com/ios-apps-just-intrusive-android-apps-research.

[4] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo, "Don't kill my ads!: Balancing privacy in an ad-supported mobile application market," in *12th ACM Workshop on Mobile Computing Systems & Applications (ACM HOTMOBILE)*, Phoenix, AZ, Mar. 2012.

[5] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing your enemy: understanding and detecting malicious web advertising," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 674–686.

[6] M. Grace, W. Zhou, X. Jiang, and A. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM WiSec)*, Tucson, AZ, Apr. 2012.

[7] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of Android application security," in *20th USENIX Security Symposium*, San Francisco, CA, Aug. 2011.

[8] *Google Play Overview*, Google, Jan. 2013. [Online]. Available: http://play.google.com/about/overview/index.html

[9] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal analysis of Android ad library permissions," in *Mobile Security Technologies (MoST)*, San Francisco, CA, May 2013.

[10] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating user privacy in Android ad libraries," in *Mobile Security Technologies (MoST)*, San Francisco, CA, May 2012.

[11] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Eighth ACM Symposium on Usable Privacy and Security (ACM SOUPS)*, Washington, DC, Jul. 2012.

[12] *These 26 Android Apps Will Steal Your Phone's Information*, Business Insider, May 2011. [Online]. Available: http://articles.businessinsider.com/2011-05-31/tech/29960806_1_malicious-code-android-market-apps

[13] *Release Notes — Google AdMob Ads SDK*, Google, Feb. 2013. [Online]. Available: https://developers.google.com/mobile-ads-sdk/docs/rel-notes

[14] K. Au, Y. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android permission specification," in *19th ACM Conference on Computer and Communications Security (ACM CCS)*, Raleigh, NC, Oct. 2012.

[15] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions

demystified," in *18th ACM Conference on Computer and Communications Security (ACM CCS)*, Chicago, IL, 2011.

[16] *The Next Level of Engagement: Rich Media Interactive Ads*, MobClix, Feb. 2013. [Online]. Available: http://www.mobclix.com/richmedia/

[17] S. Shekhar, M. Dietz, and D. S. Wallach, "AdSplit: Separating smartphone advertising from applications," in *21st USENIX Security Symposium*, Bellevue, WA, Aug. 2012.

[18] S. Perez. (2013) Nearly 60k low-quality apps booted from Google Play store in February, points to increased spam-fighting. [Online]. Available: http://techcrunch.com/2013/04/08/nearly-60k-low-quality-apps-booted-from-google-play-store-in-february-points-to-increased-spam-fighting

[19] E. Protalinski, *Google updates Play policies*, Aug. 2013, http://thenextweb.com/google/2013/08/23/google-updates-play-policies-to-ban-apps-and-ads-modifying-devices-require-games-use-its-in-app-billing-service/.

[20] *Airpush new AirSDK*.

[21] T. Book and D. S. Wallach, "A case of collusion: A study of the interface between ad libraries and their apps," in *3rd Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, Berlin, Germany, Nov. 2013.

[22] *Jumptap Android SDK Integration*, Jumptap, Jun. 2013, https://support.jumptap.com/index.php/Jumptap_Android_SDK_Integration.

[23] P. Golle, "Revisiting the uniqueness of simple demographics in the US population," in *5th ACM Workshop on Privacy in the Electronic Society (ACM WPES)*, Alexandria, VA, 2006, pp. 77–80.

[24] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "AdDroid: Privilege separation for applications and advertisers in Android," in *Seventh ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Seoul, Korea, May 2012.

[25] S. Rosen, Z. Qian, and Z. M. Mao, "AppProfiler: A flexible method of exposing privacy-related behavior in Android applications to end users," in *Third ACM Conference on Data and Application Security and Privacy (ACM CODASPY)*, San Antonio, TX, Feb. 2013.

[26] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets," in *19th Annual Network and Distributed System Security Symposium*, San Diego, CA, 2012.

[27] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale," *Trust and Trustworthy Computing*, 2012.

[28] L. Zhang, D. Gupta, and P. Mohapatra, "How expensive are free smartphone apps?" *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 16, no. 3, pp. 21–32, 2012.

[29] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof," in *Seventh ACM European Conference on Computer Systems (ACM EUROSYS)*, Bern, Switzerland, Apr. 2012.

[30] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, H. Haddadi, and J. Crowcroft, "Breaking for commercials: Characterizing mobile advertising," in *12th Internet Measurement Conference (IMC)*, Boston, MA, Nov. 2012.

[31] *Google Play Developer Distribution Agreement*, Google, Jul. 2013, https://play.google.com/about/developer-distribution-agreement.html.