Party Pooper: Third-Party Libraries in Android

Gabriel Nunez

Acknowledgement

This work was completed with help and advising from Anthony D. Joseph (research advisor), Paul Pierce (student), and Eric Brewer (second reader).

Party Pooper: Third-Party Libraries in Android

by Gabriel C. Nunez

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Anthony D. Joseph
Research Advisor

(Date)

* * * * * * *

Professor Eric Brewer
Second Reader

(Date)

Party Pooper: Third-Party Libraries in Android

Gabriel Nunez
Computer Science Division, UC Berkeley
gnunez@cs.berkeley.edu

Abstract

Third-party libraries (3PLs), such as advertising networks, gaming networks, and analytics engines, are an integral part of modern mobile platforms. If Android developers want to integrate functionality provided by 3PLs, they must bundle opaque binary code into their applications. Unfortunately, developers must in essence overprivilege their Android applications by requesting dangerous permissions, such as full Internet access, solely for the purpose of supporting 3PLs. Mixing 3PLs and dangerous permissions introduces vulnerabilities and risks to potential compromise of private user data, especially in an uncurated application marketplace. This work presents *AdDroid*, a proof-of-concept implementation that applies the principle of least privilege to mobile applications and advertising 3PLs by introducing the notion of third-party privileges directly into the Android API. AdDroid minimizes the burden of change to application developers and consumers, improves privacy, and supplies independent controls for 3PLs. AdDroid eliminates overprivileging in 44% of advertising-supported free applications. We also study how much advertising-supported "free" applications may cost users in terms of their limited monthly data plans and how AdDroid addresses this concern. Finally, we present possible deployment plans of the new system.

1 Introduction

In traditional commodity operating systems, applications run with the full authority of the users that invoke them. Such a design means that a user invoking an application must rely on the application to not abuse its authority and be bug-free. If an application is malicious it can take action beyond what the user originally intended. Likewise, if an application is buggy and is exploitable in some way, it can use the full user's authority to perform dangerous actions.

Modern mobile operating systems such as the Android Open Source Project (i.e., AOSP, or Android) have identified this problem and limit the actions a smartphone application can perform based on certain permissions declared by the application developer. When a user installs an application Android presents them with a list of requested permissions. If the user chooses to install the application, the operating system confines the application's behavior to those permissions.

Unfortunately, the AOSP permission system was not designed with third-party libraries (3PLs) in mind. In the current state of the AOSP, if a developer compiles his or her application (i.e., host application) with a 3PL, such as an advertising network's software development kit (SDK), then the 3PL inherits the same permissions available to the host application. Advertising on mobile platforms is an integral part of the mobile ecosystem, enabling developers to publish free and low cost applications in exchange for advertising revenue. Gaming networks and analytics engines similarly provide important components and features to application developers and the end-users. Third-party library providers require application developers wishing to integrate the enhanced features into their applications to embed proprietary SDKs (i.e., 3PLs), distributed in binary-only form, directly into their applications. Buggy or malicious code puts the application and user at risk of potential compromise. DreamDroid is a recent example of mobile malware [53].

In order for 3PLs to provide their intended functions (e.g., retrieve an advertisement) they must request dangerous permissions, such as full Internet access, location based information, and access to phone state. The Android documentation defines a dangerous permission as "A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user" [45]. Private user data, or privacy-sensitive data, describes data a user may wish not to share with others and includes, but is not limited to: geolocation, list of contacts, age, gender, financial data, calendar entries, browser history, email, SMS, and unique device identifies (e.g., IMEI, UUID or UDID). Since application developers bundle these 3PLs as part of their host applications, they must now request these dangerous permissions for their entire application. This means a simple host application that includes desktop wallpapers, for example, that requires no network functionality must now have multiple dangerous permissions in order to become advertising-supported. Likewise, a single player game reporting high scores to a gaming network now requires the `INTERNET` permission and can send arbitrary data to remote hosts. Additionally, privacy controls on the Android platform are less than adequate, and this inadequacy aids in the leaking of private user data. This is a problem of both privilege separation as well as overprivileging. Ideally, Android should treat the host application and 3PLs as separate entities with an independent set of permissions based on their behavior.

Privacy- and security-conscious users view permissions on the Android platform more critically than Android's top competitor, iOS. The major contributing factor to this difference has to do with the way Google and Apple distribute applications to their respective users. Google allows any registered application developer to publish their applications or application updates directly to the Android Market, which are then immediately made available to the public for download. Google treats all submitted applications as equals, regardless if the application is functional, buggy, benign, or malicious. Only after a number of users report or flag an application as inappropriate does Google review the application for potential threats. In juxtaposition, Apple reviews each submitted application and update before making it available to the public. In other words, Apple has a curated App Store, comprising applications vetted or certified as safe, while Google's Android Market is susceptible to abuse.

AdDroid is a proof-of-concept implementation to solve overprivileging and privilege separation for 3PLs; it is a custom branch off of the AOSP version 2.3.3 (Gingerbread) source. The AdDroid system is a flexible application programming interface (API) built directly into the AOSP framework that provides advertising support for multiple advertising networks. AdDroid also provides 3PL-specific permissions since we built the API into the AOSP. Application developers can request these lesser-privileged permissions, which give them access to the AdDroid API. AdDroid's "advertising privileges" include two new permissions: `INTERNET_ADS` and `TRANSMIT_LOCATION_ADS`. The API is narrow, and allows for requesting advertisements from multiple advertising networks, while optionally specifying tracking data used in targeted advertising. The new backing AOSP advertising service then handles all network and location based requests, abstracting that functionality away from the application and 3PL. In return, applications receive advertisements fetched by the backing service without needing to have generic dangerous permissions. In essence, AdDroid serves as a privilege separated advertising service while denying 3PLs access to privacy-sensitive user data.

Even though we implemented AdDroid on the Android platform, we see the concepts in this work as a generic approach applicable to other smartphone platforms: iOS, Windows Phone 7, BlackBerry, etc. AdDroid also targets advertising 3PLs, but we can extend it to provide functionality for other common 3PLs (e.g., gaming networks, analytics, game engines).

The key goals of AdDroid include minimizing the burden to application developers and consumers, improving privacy controls, having independent controls for 3PLs, and reinventing the permission-system user interface (UI). As part of our work we conduct a case study of the top free applications in the Android Market, showing that the AdDroid system could reduce overprivileging (with respect to Internet access) in 44.0% of free applications that currently use at least one advertising network. We also conduct a study to determine the amount of data usage and cost advertisements incur on users' data plans.

This work began as a class project with Paul Pierce, hence the pronoun inconsistency (i.e., we versus I). I thank him here, as well in my acknowledgments section, for his input and hard work. Specifically, Paul had major contributions in sections: 2.3, 2.5, 2.6, 5.1, and 5.2.

In § 2 we discuss background, permission deficiencies, overprivileging with 3PLs, lacking privacy controls, and topics concerning application developers. Then, in § 3 we outline the related work in the space. Next, we introduce the design and architecture of AdDroid in § 4 and follow up with our implementation in § 5. In § 6 we offer a discussion on possible deployment options and the related challenges with each approach. Section 7 contains some thoughts on future work, and we conclude in § 8.

2 Motivation

TaintDroid [20] and AppFence [29] have shown that advertising networks are among the worst offenders when it come to accessing users' privacy-sensitive data; for this reason we developed AdDroid. AdDroid is a proof of concept implementation that addresses the lack of user control over their data and what they choose to share with advertising networks. A user might not want to share their location with any third parties, or perhaps they are willing to share location information with well-known third parties while denying it to others. Conceivably a user is willing to share such information when out shopping, with the potential to save money through targeted advertising, but is not willing otherwise. Developers should give users greater control over their data, as this is something currently lacking in Android. AdDroid is an important first step because advertising on mobile platforms is an integral part of the mobile ecosystem as it enables developers to publish free and low cost user-installable applications in exchange for advertising revenue.

I begin in § 2.1 with background information on the Android permissions system and reasons why it is deficient. Then I continue in § 2.2 with a discussion on the structure of Android and how 3PLs inheriting host application permissions creates problems. I expand on this topic with § 2.3 and explore how the bundling of 3PLs into host applications introduces vulnerabilities and the risk of application compromise. I cover the conflicting interests on sharing user data between advertising networks and consumers in § 2.4. In § 2.5 I present the need to extend the AOSP with lesser-privileged, finer-grained permissions. Section 2.6 contains a case study that provides proof that a significant number of applications can benefit from a form of limited Internet access. The second case study in § 2.7 explores how limited control over advertisements can cost users money. Finally, I present several arguments in § 2.8 of how this work benefits, eases the transition for, enhances the products of, and improves the Android ecosystem for developers and device manufacturers.

2.1 Android Application Permissions

There are two main types of permission schemes found in systems today: user permissions and application permissions. Traditional user permissions, such as those found in Unix and Linux, enable applications to run with the full user privileges of the user who ran the program. The running application inherits the permissions owned by the user and runs with access to all the same resources available to that user. Security risks and vulnerabilities become a major concern in these types of systems with the introduction of third-party software. Risks introduced by any malicious code from the third-party software includes unrestricted access to the system and all resources available to the user. Likewise, even if the third-party software is not intentionally malicious, any of its vulnerabilities welcomes potential compromise, after which the adversary gains the same access as the user who ran the application. In juxtaposition, application permissions enable each application to have and run with its own customized set of permissions. The advantage in such a system is that it is generally the case that the application only requires a subset of the overall user's permissions. The limited subset of permissions available to the application hampers any malicious or compromised third-party software from nefarious activities. Android, among other systems (e.g., BlackBerry, Google Chrome), implements application permissions for this exact reason.

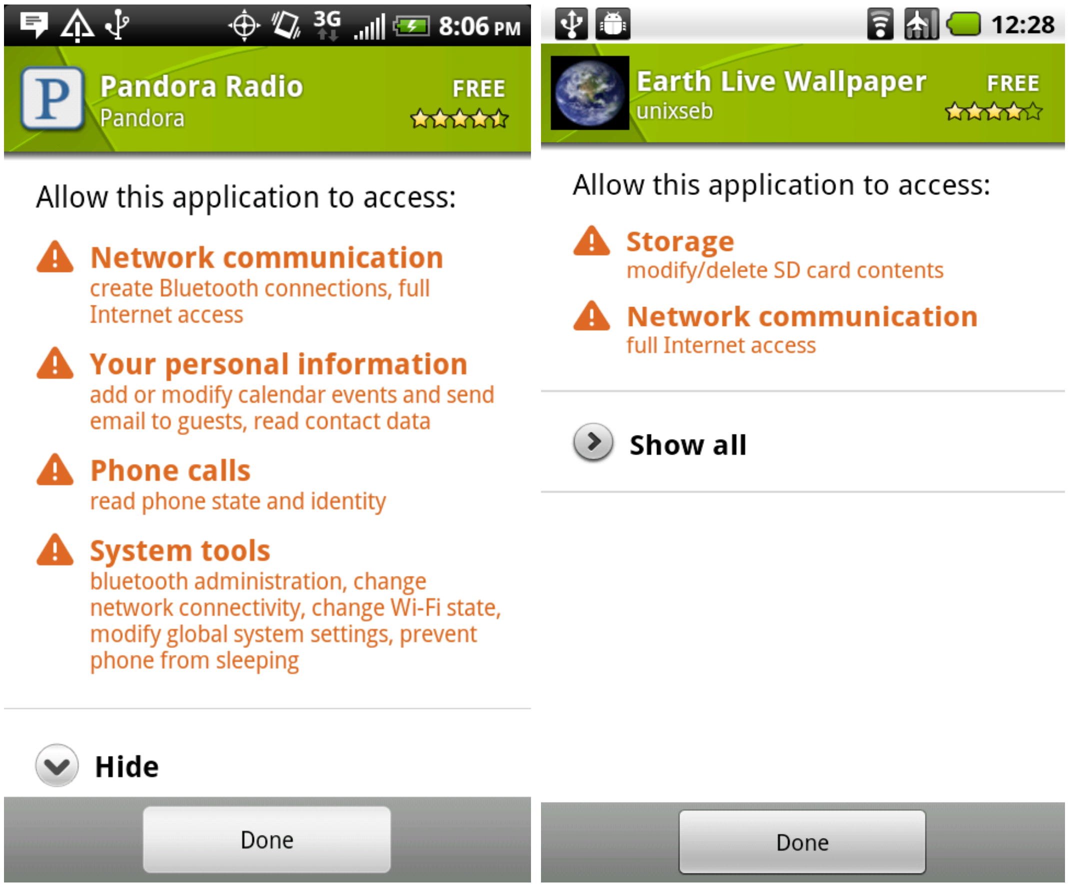
The security model employed by the Android system requires applications to explicitly request permissions for discrete system operations in a manifest. When a user installs a new application or when the permissions for an existing application have changed, the system prompts the user to authorize the requested permissions as found in the manifest bundled with the application. During installation, the Android system shows the user a list of permissions requested by the application. The user, at this time, now has the ability to choose whether or not to install the application. Accepting the installation signifies that the user approves the list of permissions per the application's manifest. Once granted, the system does not prompt the user for permission again at the time of use. The permissions of the application are a do-or-die, or an all-or-nothing, proposition; a user cannot accept some permissions while rejecting others. Further, the different permissions can be vague (e.g., `READ_PHONE_STATE`). The way Google implemented the Android permission system may not be the best option for users. Figure 1 shows a sample of permission installation screens for applications in the Android Market.

Felt et al. performed a related study to better understand the full ramifications of such a permission system [24]. The authors studied a large collection of Android applications with respect to their requested permissions. (The authors also looked at Google Chrome extension permissions, but I will not cover them here.) Their main goal was to "…[A]ssess whether the potential benefits of application permissions are being realized…" and verify that they support the claim made above, that applications generally require less than full permissions. Their study determined that on average free and paid applications request 4.71 and 5.83 total permissions, respectively. Total permissions includes both non-dangerous and dangerous permissions. Of the available 56 dangerous permissions in Android, free and paid applications request 3.46 and 3.99 dangerous permissions on average, respectively. Additionally, even the most highly privileged application requested less than half of the available dangerous permissions.

The authors suggest that the paid applications request slightly more dangerous permissions due to the fact that they offer enhanced or premium functionality over free applications. They conclude that application permissions are effective at reducing the privilege level for Android applications. Among their positive results they discovered some problems. A significant number of applications that requested dangerous permissions either could have provided similar functionality without some of the dangerous permission(s) or never required some of them in the first place. Finally, they found that almost all applications request at least one dangerous permission, 93% and 82% for free and paid applications, respectively. This is significant because this means that practically all applications will display a security warning upon installation, quickly desensitizing users and encouraging them to ignore the implications of the prompts.

2.2 The Current State of Android

Although the permission system as currently implemented in Android has its advantages, there are inherent security- and privacy-related flaws in its design. Unfortunately, the AOSP permission system was poorly designed with respect to 3PLs and is entirely lacking with respect to the advertising model. In the current


8:06 PM

Pandora Radio
Pandora
FREE

Allow this application to access:

Network communication
create Bluetooth connections, full Internet access

Your personal information
add or modify calendar events and send email to guests, read contact data

Phone calls
read phone state and identity

System tools
bluetooth administration, change network connectivity, change Wi-Fi state, modify global system settings, prevent phone from sleeping

Hide

Done

12:28

Earth Live Wallpaper
unixseb
FREE

Allow this application to access:

Storage
modify/delete SD card contents

Network communication
full Internet access

Show all

Done

Figure 1: Sample install screens from two popular Android applications. These screenshots show the install-time permission system, as well as how Android presents permissions to the user.

state of Android, 3PL developers require application developers wishing to integrate additional functionality into their applications (e.g., advertisements, leaderboards, analytics) to embed proprietary binary-only SDKs directly into their applications.

The 3PLs require communication with third-party servers in order to provide the desired enhanced functionality such as retrieving advertisements, reporting scores, or reporting usage statistics for analytics. In order to communicate with the requisite servers, these 3PLs require dangerous permissions, such as full Internet access, location based information, and access to phone state and identity. Since developers integrate these 3PLs as part of their applications, they must now request dangerous permissions for their entire application even if the functionality their application provides does not require these permissions. For instance, a simple single-player game that requires no network functionality to operate must now request multiple dangerous permissions in order to become advertising-supported, participate in a gaming network, or report usage statistics. This is a problem of both privilege separation as well as overprivileging. The Android model does not allow application developers or users to assign one set of permissions to the host application while assigning a different set of permissions to bundled 3PLs. Android does not even attempt to differentiate between application code or code bundled from 3PLs. By design it could not even if it wanted; all code appears the same to the Dalvik virtual machine.

Ideally, the host application and 3PL(s) should be separate entities with different permissions based on their behavior and individual requirements. This is currently not possible with Android; the current model is a do-or-die, an all-or-nothing proposition determined at application install time. Without enhanced mechanisms 3PLs receive full permissions and access to sensitive data (i.e., the 3PLs receive the same permissions as the host application). Consequently, 3PLs now have access to all the systems resources that the host application does. By design, 3PLs inherit all dangerous permissions granted to the host application as well as obtain the rights to application-accessible sensitive data. These permissions and data might include the following: read or write access to contacts, record audio, take pictures, read or send SMS, make phone calls, read phone state and identity, read and write to storage, obtain fine- or coarse-location, modify system settings, and perhaps create network sockets to arbitrary hosts with the ability to transmit any of this sensitive data. This work's focus is to address the contention caused by 3PLs and overprivileging. AdDroid solves overprivileging with respect to advertising.

Users may be unaware of subtleties that exist in the current state of Android with respect to 3PLs. First, 3PLs install and run without user consent. Few developers openly publish what, if any, 3PLs they bundle with their applications. The best example I can think of where publication of 3PLs does occur are games which boast of the inclusion of a gaming network or engine. Users welcome this inclusion because it increases the competitiveness due to leaderboards and achievements, most of which users can publish to their favorite social network.

Second, I have yet to see a developer who publishes, such as in the description found in the Market or third-party application store, the advertising network or analytics engine they bundle with their applications. Some applications present users with a terms of service (ToS) (or similar) that they must read and accept prior to use. One of the purposes of the ToS is for application developers to outline the usage rights granted to the user and, if applicable, what user data the application accesses, how it uses it, how it stores it, and for how long it kept the data. Unlike the host applications, 3PLs do not present users with their own ToS, which may radically differ from the host application's ToS. The end result is that users are not informed about what data 3PLs share with third parties, neither how they use it nor the retention period. Specifically, the developers of the bundled 3PLs either never or only after application installation and execution provide identifying information, as in the case of an advertising network overlaying advertisement images with the text "Ads by foo." There are 3PLs that exist that never provide any means of identification to the consumer.

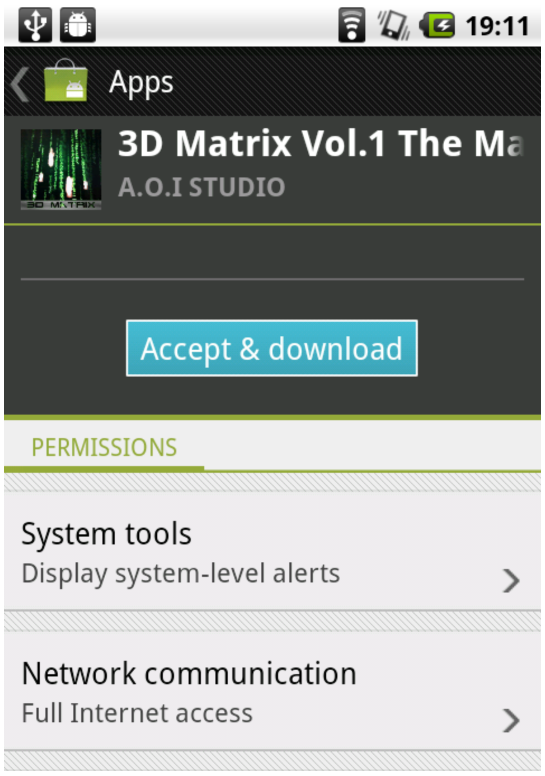


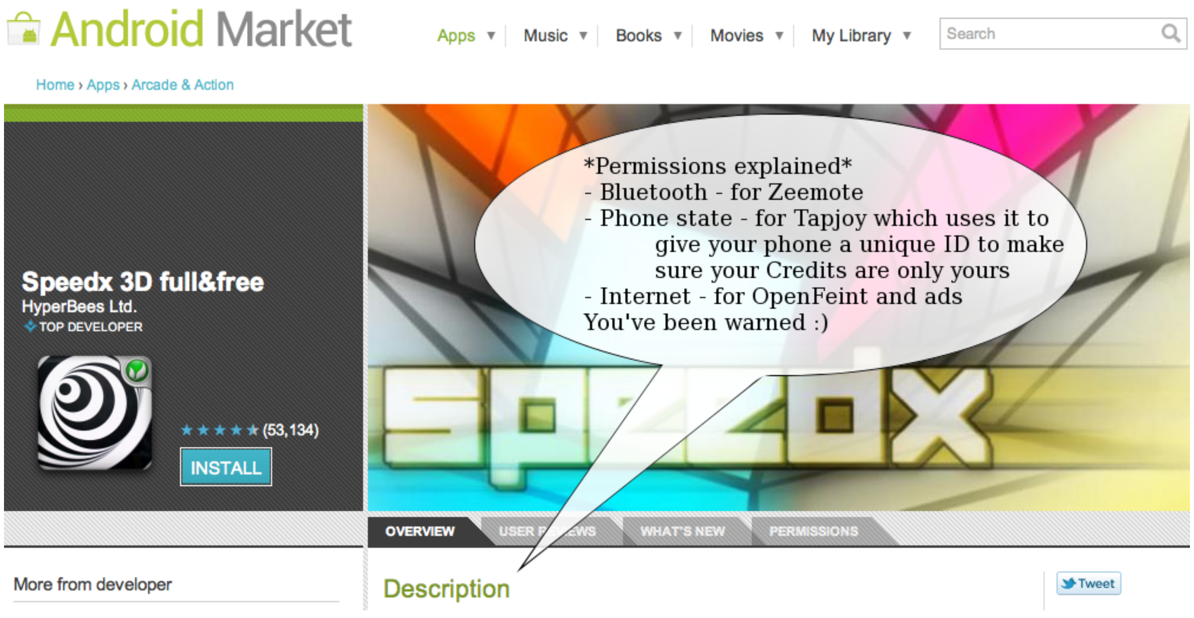
Figure 2: A Matrix live wallpaper requesting the `SYSTEM_ALERT_WINDOW` and the `INTERNET` permissions at install time. Another example of the do-or-die, all-or-nothing proposition.

Third, it is often difficult for users to determine the origin of functionality. Put differently, it is difficult to determine if the application developer or the 3PLs provide functionality x.

Lastly, Android's permissions documentation is incomplete and hard to navigate. Through personal experience, talking with anyone familiar with Android development has lead to at least one or more comments regarding the difficulty of navigating the AOSP documentation and its incompleteness; it has been slowly improving over time. It is difficult for developers to determine the minimal set of permissions their application requires. Application developers will then request permissions because they *might* use them rather than that they *will* use them. Conversely, consumers often question the validity of an application's list of permissions. They wonder why a live wallpaper, for example, needs fine-grained access to their location along with full Internet access. Figure 2 contains a similar example. According to the Android online documentation [38], very few applications should use the `SYSTEM_ALERT_WINDOW` permission. Here, the live wallpaper is requesting access to display alerts and to access the Internet. To combat user confusion, it is becoming common to find developer-provided lists of why their application requires the requested permissions, as seen in Figure 3.

2.3 Malicious Advertising Networks

In the existing Android model, it is requisite that application developers wishing to integrate additional functionality into their applications embed proprietary binary-only 3PLs directly into their applications. Figure 4 shows the current layout of the AOSP system, how applications integrate 3PLs, or advertising



Permissions explained
- Bluetooth - for Zeemote
- Phone state - for Tapjoy which uses it to give your phone a unique ID to make sure your Credits are only yours
- Internet - for OpenFeint and ads
You've been warned :)

Figure 3: Speedx 3D is a popular application in the Android Market. In the application's description the developer provides brief details on why the application requires certain permissions and for what the application uses them. There is no proof that someone has proven the validity of these statements. Note the disclaimer "you've been warned," followed by a smiley face.

SDKs in this example, and how they interact. This integration requires an application developer to trust 3PLs with the integrity of the application itself. This introduces the risks associated with potentially vulnerable or malicious third-party software entering the application. An unscrupulous 3PL developer could inject unexpected code into its host application without the application developer's knowledge. Popular advertising networks have previously served malicious advertisements [42, 46]. There is no reason to think an attacker could not attempt a similar attack directed towards the Android platform. Should such an attack be possible, it would serve to strengthen our motivation for an integrated advertising framework. Furthermore, the 3PL (i.e., SDK) source code is not made publicly available for review to protect proprietary and intellectual property. If an application is later deemed malicious or in violation of users' privacy, it becomes difficult to determine who is to blame. It could equally likely be the application developer or any of the included 3PLs, making it difficult for users to know whom to trust and whom to avoid. The decoupling of host application and 3PL permissions along with the integration of a 3PL API into the AOSP removes this need for blind trust of a third party.

2.4 Additional Concerns

Generally speaking,[1] advertising networks primarily make a profit through two interactions with consumers: advertisement impressions and advertisement clicks. An impression is simply when a web site or, in this case, a smartphone application loads an advertisement and displays it to the user (i.e., the application impressed the advertisement upon the user). Advertisement clicks occurs when a user clicks on one of the

[1]First and foremost a disclaimer: I do not make claims to having any insider knowledge about how advertising networks function nor do I understand the decisions behind the development process of the SDKs that they make available to application developers.

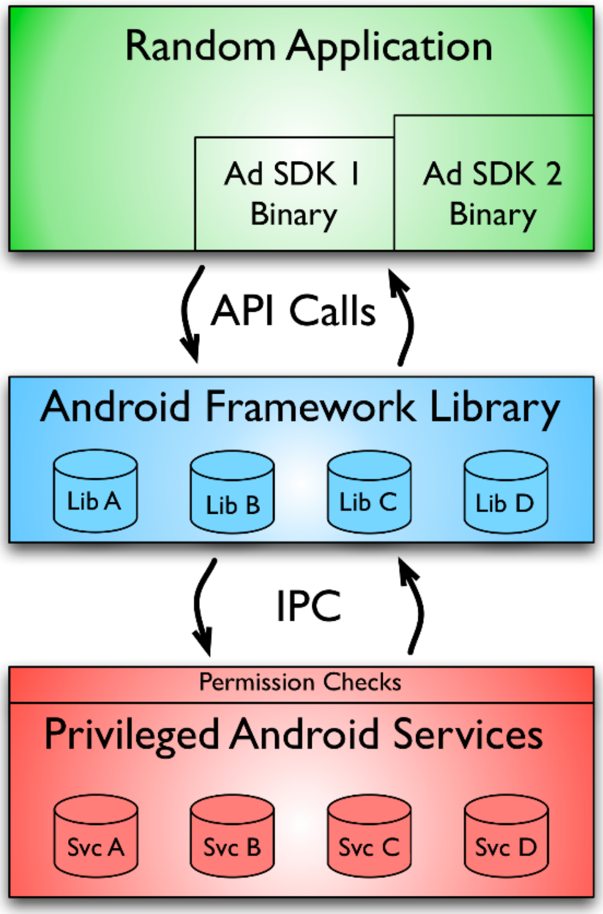


Figure 4: Diagram showing how applications currently work in AOSP. Developers integrate SDKs directly into their application, and that combined codebase acts as a single program, invoking the Android framework library. The application then invokes the Android framework with the permissions of the user, which then communicates with the privileged Android system services via inter-process communication (IPC). The Android system service performs the permission checks.

displayed advertisements which generally forwards the user to a webpage with content related to the advertisement that he or she clicked. For each advertisement impression or per larger quantity of impressions (e.g., per thousand) the advertising network awards the requestor of the advertisement (e.g., website owner, application developer) a sum of money. Advertising networks pay larger sums of money to the requestor for advertisement clicks than for impressions because they have a higher chance of generating revenue for the advertising company. An example of this is if a user clicks an advertisement promising ten percent off any same day purchase from their favorite e-tailer, they are more likely to make a purchase than those users who did not click the advertisement.

Advertising networks make it possible to display "targeted" advertisements to users, rather than generic advertisements which may not match the consumers' interests. Advertisers pay more money for targeted advertisements because the probability of matching a user's interest is higher; this in turn increases the probability of an advertisement click, and therefore a potential new or return customer. On the surface this sounds like a great way to potentially save everyone money, but there exists what I refer to as a "phenomena of contention," or a tug-of-war, between advertising networks and consumers. Advertising networks desire as much information as possible about each consumer to whom they serve advertisements in order to select the most appropriately targeted advertisement. Having a user's age, gender, income, interests, current location, and list of contacts, for example, would allow the advertising network to select the best coupon from the long list of restaurants of which they are actively advertising. The more information they gather, perhaps illegally or unethically, the higher their revenue. The extra information will greatly benefit the advertising network financially because it can display advertisements, which are meaningful to the user, and meaningful advertisements earns them more money. Details about the user enable better targeted advertisement impressions, which in turn improves advertisement clicks (i.e., increases clickthrough rate, or CTR), which then implies they can charge more per impression. Thereupon they can pay application developers more and increase their chance of drawing in more application developers and entities electing them to advertise on their behalf; essentially advertising networks are competing for application developers. Bluntly speaking, advertising networks have monetary motivation to obtain as much detail about a user as possible.

On the other end of the rope are the consumers. I am certain that few people have any reservations about freely saving money, but in the depicted scenario there is a hidden cost. Users must be willing to share their privacy-sensitive information with unknown third-parties (e.g., advertising networks) in order to receive the most appropriate advertisements. The conflict arises in that on one hand there are users who trust the developer of their favorite application (e.g., 'Aggravated Fowls') with their privacy-sensitive information, while on the other hand there are users who are wary about sharing it with others. So advertising networks are pulling in one direction to get as much user information as possible while we have users pulling in the other direction to keep their personal information private. It is unclear at what costs to an individual's privacy he or she willing to view relevant advertisements and potentially save money; we do not understand all motivating factors behind the decisions to allow advertising networks to have and use one's personal information. These remain open problems. However, we do know that companies like Groupon or retailer loyalty cards have been successful [14].

Even if we perfectly understood the problem, preferences and willingness of what to share and with whom to share will vary from person to person. Therefore, the principle goal of this work is not to provide a means of blocking all third parities from ever receiving useful user information, but to give users control over their own privacy-sensitive information. Not everyone will need or want to use the mechanisms provided to keep their personal information private. On the contrary, some consumers will want to allow transmission of personal information to advertising networks because they view privacy differently or because the payoff is great enough to allow it to happen. Still, others might never be willing to allow such activity. This work is more of an education to users and bestows upon them power over their own information in regards to what and with whom they will share their data. Though this work is incomplete when compared to its overall goals, I envision this work coupled with existing work to offer a complete, usable system: AdDroid-like mechanisms will provide the control and fundamental changes to the Android operating system while a TaintDroid- or AppFence-like system can provide the necessary enforcement mechanisms (see § 5.3 for further details).

2.5 Applying the Principle of Least Privilege to AOSP

The principle of least privilege is just as pertinent to 3PLs in host applications as it is to applications running on Android. Noted earlier, Felt et al. found that Android applications do benefit from application permissions and that the permission system does help to apply this principle [24]. Unlike any other work to my knowledge, this work extends the principle of least privilege one layer further down than currently available in the AOSP. Like Android applications that require less than full privileges (i.e., a subset of the 56 dangerous permissions), 3PLs should follow the same principle. I argue that 3PLs generally require less permissions than the host application itself.

For situations where similar, yet limited functionality is still required, Google should extend the Android permissions system to support these needs. In AdDroid, for instance, we created two new, lesser-privileged permissions to support advertising networks. Advertising 3PLs comprise a substantial portion of all 3PLs. In the worst case scenario, the bundled 3PLs require more permissions than the host application. This is not any worse than the current state of Android because the required permissions will be at worst equivalent. I say equivalent because we can substitute some permissions for new, lesser-privileged permissions. In the average case scenario we can eliminate the majority of dangerous permissions by separating the host application and 3PLs and only requesting the privileges independently required for each to function (details found in § 2.6). Another possible fine-grained permission would be Internet access but limited to a predetermined list of hosts with which the host application and 3PL can communicate. This would follow the example of unsigned Java applets, which may only communicate with the origin server [57], and would provide similar functionality to whitelisting hosts for Google Chrome extensions [27].

Apple demonstrates improved user control over advertising networks in their iAd mobile advertising network. First, Apple allows users to dynamically grant or deny geolocation to Apple by enabling users to turn on or off location services for iAds. An important feature is that users control this option independent of any application's permissions. Second, Apple grants users control over whether the users received targeted, or interest-based advertisements versus generic advertisements. Apple KB HT4228 [30] states that users can opt-out of targeted advertisements by simply following a link which will set the appropriate cookie. Since as early as April 2011 [31, 52], Google started an initiative to allow users to opt-out of interested-based advertising for the desktop browse, the Android Market, and the Google search application on iOS. Google and Apple implement a limited approach, however, because these options apply only to AdMob and iAd advertisements and no other third parties. Google still lacks control over location-specific advertisements. The goal with AdDroid is to overcome these limitations, not for only Google's own advertisements but for all advertising networks.

2.6 Over-Privileging in Advertising 3PLs

In order to determine if the AdDroid system would actually benefit real applications, Pearce and I performed an analysis on a group of applications collected from the Android Market. Our application set is the same set used by Porter Felt et al. in their work on the effectiveness of application permissions [24]. They

Category	AdMob	AdSense	Mill. Media	Mob Clix	Medialet	ZestAdz	Total Apps w/ Ads	Total Apps
Top Free	231	158	32	34	6	4	352	762
Top Paid	4	6	0	2	0	0	9	100
Free Recent	63	10	1	0	0	0	71	100
Subtotal	298	174	33	36	6	4	432	962

Table 1: Breakdown of the number of applications in each category using a given advertising network. Note: Some applications include multiple advertising networks, and we count them in multiple columns.

collected the application set in late 2010, which consists of 962 applications. The collection included the 762 top free applications, 100 top paid applications, as well as the 100 most recently added free applications.

Initially, we identified six advertising networks used throughout the application collection. These advertising networks were AdMob, AdSense, Medialet, Millennial Media, Mob Clix, and ZestAdz. From this application set we disassembled each application and then examined the included namespaces to determine if the application developer had included any advertising network SDKs in the application bundle. Table 1 shows a breakdown of how many applications used each advertising network in each category. 352 of the 762 top free applications (46.2%), 71 of the 100 recent free applications (71.0%), and 9 of the top 100 paid applications (9.0%) used at least one advertising network. Some applications included multiple advertising network SDKs, which is why the individual columns from Table 1 do not add up to the totals. For example, 93 of the 762 top free applications included 2 or more advertising network SDKs. From this data it is clear that advertising networks are prevalent in Android applications, and their usage appears to be increasing.

Once we identified the prevalence of advertising networks, we needed to identify if overprivileging exists. To answer this question we focused on the Internet permission. From the pool of top free applications we knew included at least one advertising network, we selected 25 applications at random for an in-depth case study. Table 2 contains the list of the selected applications. For each application selected, we loaded the application into the Android emulator inside a virtual machine, and then exercised the applications manually. We continued manually exercising until we felt we understood the functionality of the applications, and explored all possible apparent user interface (UI) events. While performing the exploration of the applications, we recorded all network communication with Wireshark and then analyzed the results. From our Wireshark traces we identified the applications that performed network communication to only known advertising networks. From this analysis we discovered that 11 of the 25 applications (44.0%) used the Internet solely for the purpose of fetching advertisements.

This result shows that 44.0% of these applications would benefit from the AdDroid system, as these applications should request the `INTERNET_ADS` permission rather than permission for full Internet access. The permission system would then limit applications with the `INTERNET_ADS` permission to fetching advertisements only, rather than performing arbitrary Internet communication. A good example of this is in Figure 5.

2.7 The Hidden Costs of Advertisements

The advertising networks we investigated give application developers the option to choose the style of advertisement they wish the end-user to see; the most common type implemented today is banner advertisements found at the top or bottom of the screen. In-application banner advertisements are commonly located at the top or bottom of the screen. Regardless of how small the advertisements appear to be, users incur

Android Market Name	Namespace of App
Halloween Sound Board	PowellDev.HalloweenSoundBoard
Folder Organizer	com.abcOrganizer.lite
Antivirus	com.antivirus
Love Poems	com.appspot.swisscodemonkeys.love
Music Downloader Lite	com.athene.android.amd
Music Downloader Pro	com.athene.android.amdp
Mortgage Calculator	com.calculator.mortgage
Audiobooks	com.crossforward.audiobooks
APNdroid	com.google.code.apndroid
Jay-Z Ringtones	com.jayz.ringtone
Soccer Scores - FotMob	com.mobilefootie.wc2010
Bars & Clubs	com.mqdp.barsclubs
Ron Burgundy	com.neatofun.anchorman
Ricky Bobby	com.neatofun.rickybobby
The Best Life Quotes	com.socialping.lifequotes
Bruce (Tobi) Soundboard	com.soundroid2012.bruce
Halloween Sounds	com.soundroid2012.halloween
Peter Griffin Soundboard	com.soundroid2012.petergriffin
Transformers Sounds	com.soundroid2012.transformers
Halloween Wallpapers	com.swn.netgallery.halloween1
Mike Epps v6 SoundBoard	com.swn.soundboard.mike_epps_sounds
TuneWiki - Lyrics with Music	com.tunewiki.lyricplayer.android
Caddyshack Soundboard	com.zero1dev.soundboard.caddyshack
Watchdog Lite	com.zomut.watchdoglite
TiKL - Touch to Talk (Beta)	mobi.androidcloud.app.ptt.client

Table 2: The 25 applications utilized in our overprivileging in-depth case study, alphabetically sorted by their namespace.

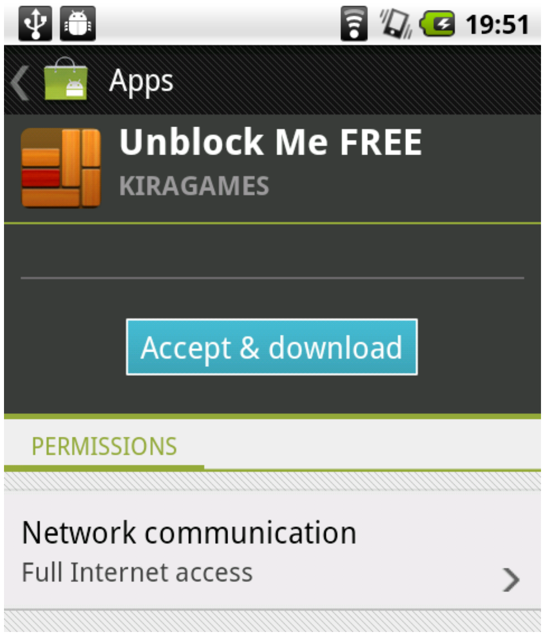


Figure 5: A free application from the Android Market that uses the `INTERNET` permission only for advertisements.

what they believe to be nominal data usages while running advertising-supported applications. Meanwhile, motivated U.S. cellular carriers are pushing their subscribers from unlimited data plans to ones with monthly data usage limits to ease the ever growing saturation of mobile networks. We see a similar data-cap push by "hard-wired" Internet service providers (ISPs) through imposing data caps on subscribers. Consequently, an increasing population of smartphone users find themselves on monthly data plans with as little as 200MB. Given these limitations, users ration their use of data-hungry applications while mobile or postpone their task until a local Wi-Fi connection is available because data overages are expensive. One item of study is to determine whether the hidden costs of advertisements are enough to alter user behavior by when or how they use advertising-supported applications or whether there will be sufficient evidence to convince users to remove advertisements by purchasing applications, which otherwise would have remained "free".

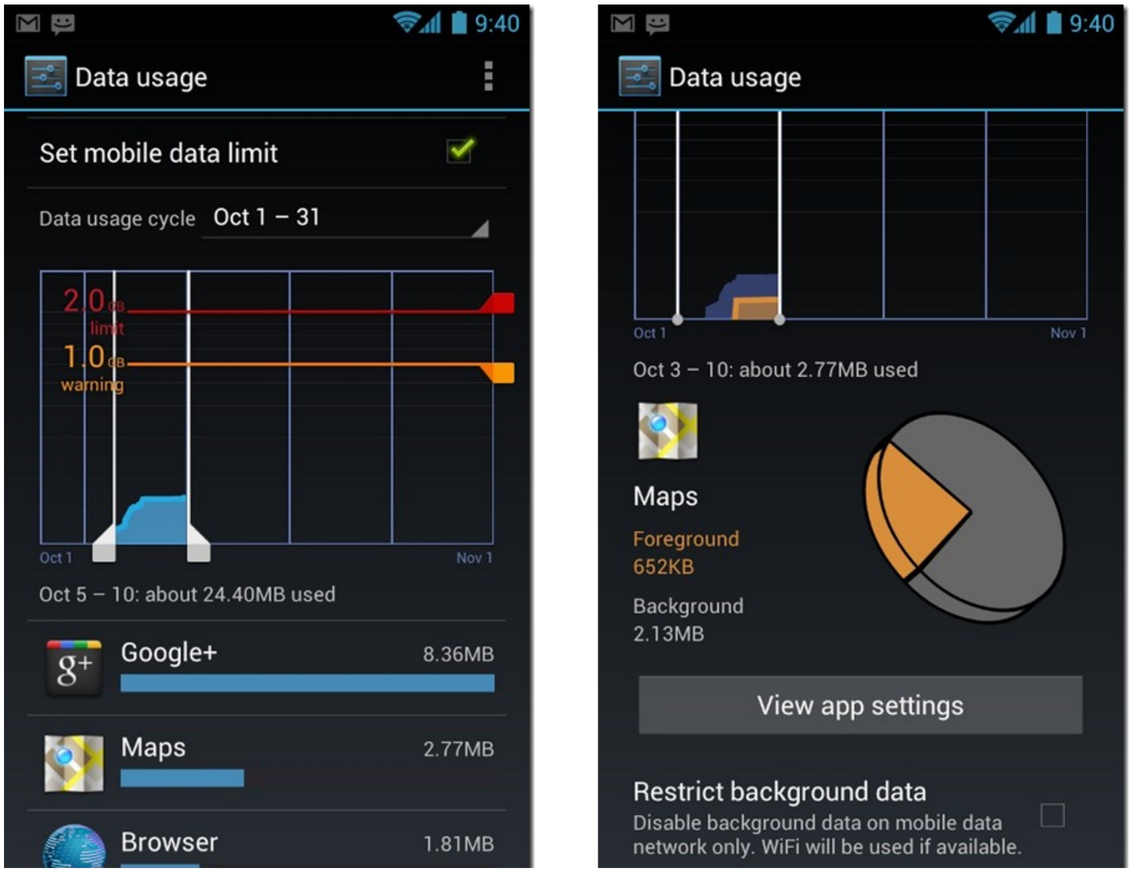
One possible outcome of implementing AdDroid into the AOSP is the ability to segregate the data usage charges between user-generated and advertising-generated network traffic. We can offer an application, like the battery usage application, that shows how much data each application has sent and received over the cellular network: two entries for both application- and advertising-generated traffic. Users who travel internationally or often roam will find these features particularly worthwhile because international rates are costly and substantial traffic while roaming gives the carrier grounds [5] to terminate the user's data services. Subsequent to our work on AdDroid, Google added a similar feature with the recent release of Android 4.0 (Ice Cream Sandwich). Figure 6 previews the user controls available to users. The Data Usage application displays data usage for the device over a billing cycle and allows users to specify warnings and limits. The application also breaks down the data used per application and differentiates between data when the application is actively used or in the background. We could extend the Data Usage application to track advertising-generated traffic as well.

We first needed a better understanding of how much in-application advertisements from advertising networks are costing users in terms of their monthly data plan. For this purpose, we assume that most smartphone users are on a limited data plan such as Verizon's 2GB or AT&T's 200MB and 2GB offerings. From that we calculate the price per byte assuming the user consumes less than 100% of the allotment with no overages, which, if they occur, increase the cost to the user significantly. Few carriers (e.g., Sprint) continue to offer unlimited plans, and T-Mobile is unique in that it offers a 2GB plan but throttles network bandwidth after a user exceeds the limit, instead of charging the customer data overage fees.[2]

We decided that the two most relevant statistics in advertising-generated network traffic are the size (i.e., bytes) of advertisements and the potential network traffic generated as the consequence of a voluntary or accidental click on an advertisement. In our testing we only considered banner advertisements since they comprise the overwhelming majority of advertisements found in applications today. We used the same testing environment outlined in § 2.6 for running our subset of advertising-supported applications. To calculate the "size" of an advertisement we used Wireshark to measure the total the number of bytes transferred for establishing the TCP session, sending the advertisement request, receiving the advertisement, and the TCP session tear-down; we then calculated their costs based off of their size and associated data plans.

We chose to base costs on AT&T's domestic DataPlus, DataPro, and International Pay-Per-Use (PPU) data rates as of April 2011: DataPlus (plan) @\$15/200MB, DataPro (plan) @\$25/2GB, International PPU (no plan) @\$0.0195/KB; all prices are in USD. For reference, Verizon's 2GB plan is 20% more expensive than AT&T's domestic DataPro plans. With regards to calculating the amount of traffic generated by an advertisement click, we used the same technique for calculating the advertisement size but started the capture immediately prior to clicking the advertisement and stopped as soon as the web page finished loading. We then calculated the associated costs based on the total bytes transferred. The sample set consisted of 16

[2]A brief reminder: these caps apply only to data usage while utilizing the cellular 3G or 4G network, not Wi-Fi.



Data usage

Set mobile data limit ✓

Data usage cycle Oct 1 – 31

Oct 5 – 10: about 24.40MB used

Google+	8.36MB
Maps	2.77MB
Browser	1.81MB

Data usage

Oct 3 – 10: about 2.77MB used

Maps

Foreground
652KB

Background
2.13MB

View app settings

Restrict background data
Disable background data on mobile data network only. WiFi will be used if available. ☐

Figure 6: The Data Usage application added to Android 4.0 (Ice Cream Sandwich).

					Mean Cost		
Variable	Min	Median	Mean	Max	DataPlus	DataPro	International PPU
Ad Size	4,406	6,890	9,028	22,095	$0.0006	$0.0001	$0.172
Ad Click Size	66,605	559,276	922,265	5,886,432	$0.0660	$0.0107	$17.563

Table 3: Breakdown of the size and cost per advertisement or advertisement click. Sizes are in bytes and we calculated the associated cost for each data plan from the mean of our sample set.

App	Cost/min	Monthly Usage Cost		
		10m/day	1h/day	4h/day
#1	$0.00026	$0.077	$0.460	$1.840
#2	$0.00023	$0.069	$0.416	$1.664
#3	$0.00081	$0.244	$1.467	$5.866
#4	$0.00005	$0.015	$0.092	$0.367
#5	$0.00061	$0.182	$1.090	$4.361

Table 4: DataPlus plan users' cost breakdown for advertising traffic while running our subset of 5 randomly selected applications.

unique advertisements from multiple advertising networks and 16 unique webpages that loaded as a result of clicking on the associated advertisement. Table 3 shows our results.

In-application banner advertisements range from around 4KB to about 22KB, costing the user from one hundredth of a cent to a little more than 17 cents on average for domestic and international rates, respectively. Remember, these are the costs incurred *per* advertisement and only consider traffic between the mobile device and the advertising network (i.e., we ignored other possible legitimate application-generated network traffic). Applications which generate Internet traffic outside of advertising purposes can potentially cost the user much more and is not considered in our study. Advertisement clicks tell a similar story but at orders of magnitude larger in terms of data usages and associated costs. A typical loaded webpage had several images, most of which were static images, while the maximum data item (at close to 6MB) was the result of loading a three and a half minute music video advertisement. An average advertisement click costs the user from one cent to over seventeen *dollars* if located domestically or internationally, respectively.

The last part of this case study was to determine data usage statistics for applications run over a length of time. These numbers are meaningful because an application developer has the ability to select the advertising network, the types of advertisements shown to the user, and how often advertisements refresh (within a predefined range). Likewise, AdDroid could throttle or limit the behavior of advertising networks based on predefined reasonable transfer limits. The most aggressive advertising networks allow developers to request a new advertisement every 15 seconds, although the advertising networks do not always return a new advertisement which saves on some network traffic. We randomly selected five applications, from our subset of 25 applications, that used the Internet only for advertisements and exercised them in our testing environment. We manually exercised each application for a minimum of 10 minutes and averaged the cumulative data traffic over the runtime to get a bytes per minute rate. Noted below are the cost results for the DataPlus, DataPro, and International PPU data rates in Tables 4, 5, and 6, respectively.

Depending on the application and selected data plan, users may find the numbers motivating enough to ration their data usage or purchase the non-advertising-supported version of an application. Purchasing applications to remove advertisements, if not to save on advertising-generated traffic per se, will eliminate the

App	Cost/min	Monthly Usage Cost		
		10m/day	1h/day	4h/day
#1	$0.00004	$0.012	$0.075	$0.299
#2	$0.00004	$0.011	$0.068	$0.271
#3	$0.00013	$0.040	$0.239	$0.955
#4	$0.00001	$0.002	$0.015	$0.060
#5	$0.00010	$0.030	$0.177	$0.710

Table 5: DataPro plan users' cost breakdown for advertising traffic while running our subset of 5 randomly selected applications.

App	Cost/min	Monthly Usage Cost		
		10m/day	1h/day	4h/day
#1	$0.06804	$20.412	$122.471	$489.884
#2	$0.06152	$18.456	$110.734	$442.935
#3	$0.21693	$65.078	$390.470	$1561.879
#4	$0.01358	$4.074	$24.441	$97.764
#5	$0.16125	$48.376	$290.254	$1161.015

Table 6: International Pay-Per-Use (PPU) non-plan users' cost breakdown for advertising traffic while running our subset of 5 randomly selected applications.

possibility of intentional or accidental advertisement clicks that can incur unsuspecting, significant financial consequences. If traveling internationally, it is strongly recommended that users make the application purchase unless they can guarantee that the application will only run over local Wi-Fi networks.

2.8 Concerning Developers & Manufacturers

So far the majority of the presented discussion has been from the perspective of the consumer. Although the consumer has the power to control a portion of what Google and smartphone manufactures offer to customers, one must convince them that the desired changes will benefit the majority of the population. Let us briefly look at why application developers, advertising networks, Google, and smartphone manufacturers would welcome the proposed changes in this master's.

2.8.1 Enhanced Permissions & User Control

Upon implementing the necessary changes into the AOSP, 3PLs would neither have implicit access to privacy-sensitive user data nor inherit all the dangerous privileges granted to the host application. Of course it may be necessary for a 3PL to require an unusually long list of dangerous permissions, and in these cases the application developer can explicitly request them as needed. However, the majority of 3PLs will generally only require a minimal set of permissions, some of which developers can substitute for the new, lesser-privileged permissions. These mechanisms will eliminate, or mitigate, application developer and users' blind trust of 3PLs. It is still the case that 3PLs might run arbitrary code, but the lower-level implementation of the principle of least privilege minimizes the risks. Nonetheless, with full integration of the 3PL API into Android, application developers should generally not need to bundle 3PLs with the application to begin with. The enhanced permissions will help protect application developers and consumers from the activities hidden in opaque, binary-only distributable 3PLs. Consumers will be able to clearly identify the bundled 3PLs and have the controls at their disposition to mange what data they choose to share and with whom. Additionally, the required changes utilize the built-in enforcement mechanisms already found in the AOSP.

2.8.2 Ease of Transition

The majority of the burden for implementing a new system falls on Google; this is our intention. APIs for gaming networks and analytics engines can be similarly implemented. We purposefully make the changes in a way that it minimizes the burden to application developers, smartphone manufactures, and consumers. The advertising API as found in AdDroid was purposefully written to mimic the most popular advertising networks' SDKs as application developers are already familiar with integrating said 3PLs. We hope to keep the changes developers must make to their application of the order of a few tens of lines of code. Some of the required changes include the following: removing the previously-imported advertising SDK, importing the new library, specifying the desired advertising network from which to fetch advertisements (e.g., AdMob, Millennial Media), and updating the method call to request the advertisement. Essentially, developers will make the changes necessary to call the 3PL-enabled Android API rather than calling an advertising network's SDK.

2.8.3 Public Image

In this section I am referring to the public perception of one's brand and products and the need to protect that image. Negative press generated by the media due to security or privacy incidents can taint a brand's image and cause a loss of customer trust.

Last year Google made headlines for applications stealing financial data and accessing unnecessary data [9, 12]. Google was also in the news this year for a significant number of applications with malware found in the Android Market [12, 53]. In 2010 Pandora was in the news for leaking privacy-sensitive user data to third parties [55]. Earlier this year the government subpoenaed Pandora in an investigation of their Android and iPhone applications [18]. An independent investigation confirms the concern [49].

Google and third party application developers are not the only ones with brands and images to protect. It is common practice for phone manufactures to customize the Android build they ship with their devices. They generally add device-specific enhancements and bundle a customize UI to help differentiate their product offering from the competition. Users have had mixed feelings about the customized UIs. There is at least one instance where a phone manufacturer's customizations introduced new vulnerabilities that are not present in the stock AOSP build [47]. HTC included a suite of logging tools with their phones that collected massive amounts of private user and device data; one of these applications is HtcLoggers. This in and of itself does not present a privacy concern, however, the HtcLoggers application responded to all requests for information instead of HTC-only applications. The `INTERNET` permission is the only requirement to gain access to the logging application and all its data. Refer to Table 7 for a subset of data HtcLoggers collects and makes available to third-party applications.

I argue that if Google makes the changes proposed in this master's that they will minimize these types of occurrences, which in turn strengthens the Android platform. It is standard practice for phone manufactures, such as HTC, to provide customized versions of Android that ship with their phones. They will be able to fully take advantage of the improved system. Application developers will be able to better protect themselves and their applications from malicious or vulnerable 3PLs, or even be able to wholly eliminate

Data made available by the HtcLoggers application:

- the list of user accounts, including e-mail addresses and sync status for each
- last known network and GPS locations and a limited previous history of locations
- phone numbers from the phone log
- SMS data, including phone numbers and encoded text
- system logs (both kernel/dmesg and app/logcat)

Table 7: A sample list of data the HTC-provided application, HtcLoggers, collects and makes available to any application with the `INTERNET` permission. The author [47] notes that he is not yet sure if one can decode the SMS text, but it is very likely. He also notes that the system logs include everything that running applications do and is likely to include e-mail addresses, phone numbers, and other private information.

them. Consumers will enjoy the extra control of their sensitive data to protect it from unauthorized third parties.

2.8.4 Improving the Android Ecosystem

An integrated 3PL API also has the potential of strengthening the Android Market ecosystem. Installing a notion of 3PLs into the core Android Market would enable users to more easily identify 3PL-supported applications. For advertising, it would be possible to tie together advertising-supported and paid versions of an application at the Market level, allowing users to more easily make an informed decision. Google has an incentive to support such a system since it enables them to track advertisement impressions and clicks, and charge a nominal fee to advertisers; this is of course optional and only if Google decides to support this business model. Advertisers have an incentive to support such a system since it could presumably make integrating advertisements into applications easier, increasing their business.

These ecosystem advantages are in addition to the reduction of overprivilege, as described in § 2.6. Advertising networks and application developers would maintain or increase their ability to monetize applications. The number of installations of an AdDroid-enabled application would increase because security- and privacy-conscious users would be more likely to install them due to fewer or less-dangerous permissions. Additional users would therefore increase the advertising-revenue for the advertising networks and application developers. With Android in control of advertising, it is possible for the system itself to delegate control over whether or not an advertisement renders. It would be possible to create an infrastructure by which a user downloads a single version of an application, which is both the paid and free version. The free version of the application is advertising-supported. If the user wishes to get rid of advertisements they can perform an in-application purchase, which would in turn tell the operating system to stop requesting the advertisements.

Additionally, as discussed in § 2.7, advertisements cost users money (in terms of limited data plan usage). With the Data Usage application (or similar) and AdDroid cooperating, the system could track advertising-generated data usage. The application could then make this information readily available for users to help them avoid expensive data overage and roaming charges. With this information, users would be able to make educated decision about which applications to purchase to avoid advertising "costs." Also, we could modify AdDroid to detect when when a device is roaming internationally, for example, and not request advertisements until the device is no longer deemed on international travel or until the device establishes a Wi-Fi connection.

3 Related Work

The structure of the related work is as follows. First, I discuss the popularity of smartphones and the increasing concern of privacy-sensitive data disclosure. Disclosure can occur through malicious behaviors or by unintentional disclosure (i.e., user confusion or developer-provided defaults). Next, I introduce the topic of software bugs and vulnerabilities. I then present relevant research on methods to detect and prevent bugs and the unauthorized disclosure of sensitive data. Then I present studies that have shown that smartphones suffer from similar problems and recent studies on addressing these issues. Lastly, I explain how AdDroid differs from existing work in the same field of mobile research.

The increasing popularity and use of smartphones for personal, financial, social, information storage, and management have triggered concerns over the proper privileging of user-installable applications and the leaking of personal information. Installed applications can leak private information through these applications to the developers and arbitrary third parties without user consent. Part of the blame lies within the existing UIs that developers provide to the users of their systems. The UI plays a large role in the amount of control a user has over the system or application. Additionally, the UI design determines a large part of the users' understanding of the system and their ability to make informed decisions regarding the privacy of their personal information. Complicated UIs will overwhelm and confuse users to the point that they either do not know how to perform the action(s) necessary to enforce their intentions or they believe they are taking the correct action(s) but in reality their decision contradicts their intended purpose. Another contributing factor to the unintentional disclosure of personal information lies with the influence of default options or selections the developers provide to their users. Users are more likely to accept the defaults the developers provide than to change the selection to their intended action, especially in times of confusion or uncertainty. On this note, Facebook has been in the news [41] for providing overly relaxed defaults with respect to users' private information. On at least one occasion [33, 43] Facebook reset all users' privacy settings to the least restrictive option, effectively making all personal information globally visible. Sharma's master's project [48] looked at Facebook and the implications of developer-provided defaults with respect to personal information and social networks.

Software bugs and vulnerabilities originate in the operating system itself or from the running applications as a result of varying types of human error in: programming languages, protocol or application design, logic flows, or typos. Scientists have developed numerous techniques to aid in detecting and preventing vulnerabilities, privacy leaks, and unauthorized disclosure.

Prevention techniques help find bugs prior to compromise or disclosure and function in a variety of ways. Privilege separation is a technique in which developers divide programs into smaller parts that they then grant only the privilege(s) necessary to perform their specific function(s), effectively limiting vulnerable applications to a subset of permissions. One development, Privtrans [7], aids application developers in an automated fashion to achieve proper privilege separation and avoid overprivileging if given programmer annotations in the source code. Privtrans relies on standard *nix process isolation to enforce separation. Information flow techniques, discussed [15] as early as the 70's, rely on policies which define how data flows within the application and how certain types of data can leave the application. Flume [34] is a system that enforces policies at the granularity of processes and standard OS abstractions (e.g., sockets, pipes, and file descriptors) and uses the Linux security module framework. Rather than relying on existing OS primitives, Resin [59] defines three new concepts to write and enforce policies: *filter objects* define the data flow boundaries, *policy objects* define the sensitive data, and Resin performs *data tracking* as the sensitive data moves within the system. The security-by-contract (SxC) concept provides similar functionality on the .NET platform [16].

Dynamic taint analysis is a method used to track the flow of information (i.e., tainted data) within a system and determine which parts of the system the data of interest has influenced (i.e., tainted). TaintTrace [8] interacts with applications' binary instructions, hence it works on any application regardless of its originating programming language. Users supply a configuration file at runtime, the program monitor propagates the taint as the program runs, and then enforces the policy by rewriting an application's binary instructions. Dytan [10] offers a generic framework with enhanced flexibility and customizability with the ability to perform control-flow based tainting. Like TaintTrace, Dytan also works on x86 executable binaries and does so without specialized support from the runtime system. These papers have proven the usefulness of dynamic taint analysis in finding and preventing application vulnerabilities. Conversely, others [50] show that the most commonly cited use case of dynamic taint analysis, detecting privacy-breaching malware such as keyloggers, has significant limitations.

Mobile platforms, including the AOSP, are not without problems. A recent study [24] on Android application permissions shows that although the Android permission system is helpful at reducing the privilege level of Android applications, a significant number of applications are requesting unneeded permissions and that almost all applications request at least one dangerous permission. It is worth noting the authors' argument that a less severe warning for limited access to the Internet (e.g., communication only with known advertising domains) would be beneficial and that free applications regularly request the `INTERNET` permission for retrieving and displaying advertisements. Additionally, the authors claim that the `INTERNET` permission alone cannot leak personal information; the application must also receive a second dangerous permission such as `READ_CONTACTS`. AdDroid, presented later in this master's, addresses these issues directly; with AdDroid, these overprivileging occurrences will be meaningfully reduced by eliminating the advertising-only need for fully privileged Internet and location information. The authors' work is primarily an exploration of the problem space, and unlike here, they do not directly attempt to solve any of the problems they discovered. On the other hand, the Kirin security service for Android [21] aims to avoid these potentially dangerous permission combinations by analyzing the application permissions at install time and enforcing user-definable security rules.

AdJail [37] is a mechanism to protect desktop browsers from malicious advertisements. AdJail uses policies and sandboxing mechanisms to isolate users from advertisements and prevent the disclosure of sensitive user data. The authors intend AdJail for web site publishers who want to protect their visitors, much like we direct AdDroid's functionality to application developers to protect their customers.

Quire [17] is a proposed solution to the problem of provenance on Android. Currently in Android, the identity of the original caller can quickly become obfuscated. Quire approaches this problem with call chains and message signing. The authors proposed that they can use Quire to help build a new advertising ecosystem with separate user-level advertising applications. Quire's objective, with respect to advertising, is to prevent advertising fraud by certifying that applications properly display and users click advertisements. Our objective with AdDroid is to enhance privacy by preventing advertising libraries from disclosing private user data.

Previous work on Android overprivilege has not considered the impact of advertising. A recent study developed Stowaway [23], a tool used to explore whether Android applications request more permissions than they require to function. One major difference between our study and Stowaway is that in their study the authors considered advertising libraries to be a legitimate use of permissions, whereas we do not. Our measurement study focuses specifically on overprivilege that stems from only advertising.

Privacy on mobile platforms is a growing public and legal concern. Previous studies have suggested that a majority of users are uncomfortable sharing personal information such as location and browsing history with advertising networks [32,40,56], while others have shown that this occurs without user knowledge [39]. MobiAd [28] is a proposed solution to allow for targeted advertising without while maintaining user privacy and anonymity. The authors also note concerns over the trade-off users make for these interest- and location-based advertisements. They conclude by stating that there is insufficient data to know when and where users are willing to accept advertisements. Cleff [11] argues that legislation is slow and well behind technological advancements and the legal problems this raises in regards to protecting users' privacy. Cleff asserts that users need control over their private data and the types of advertising they receive.

Application developers and users developed their own mechanisms to overcome the lack of user control and fear of permission abuse. The first, LBE Privacy Guard [35, 36], is a user-installable application for unmodified, yet rooted manufacturer-supplied Android builds. Rooting an Android device is analogous to jailbreaking an iOS device; each process provides control over the system which is otherwise blocked. The second implementation, CyanogenMod [13] is a custom, pre-rooted firmware with permission revocation built in.

TaintDroid [20] and PiOS [19] are interesting studies on the two most prevalent smartphone platforms, Google's Android and Apple's iOS, respectively. These two studies utilize information flow and dynamic taint analysis techniques to show that malicious applications and advertising networks (i.e., 3PLs) are often exfiltrating sensitive personal information to application developers and other third parties. They also discovered that the applications and 3PLs are divulging unique device identifiers. Known advertising networks are among the top of the list of offending third parties. Journalists [55] conducted their own study of Android and iPhone applications with similar results. Perhaps partially in response to the work of PiOS, Apple strongly discourages developers from obtaining unique identifiers and has deprecated the applicable methods in the recent release of iOS 5. On the Android front, TaintDroid and the preceding work by the same authors are likely to be the most well-known privacy-related smartphone publications for drawing attention to this research space. AppFence [29] expands upon TaintDroid by tracking more types of user data along with providing a mechanism to detect the application-level impact of blocking the exfiltration of sensitive data by either dropping tainted data packets or by offering shadow (i.e., false) data in place of sensitive data to the application.

This work differs from TaintDroid and AppFence in several key aspects. First, AdDroid is purposefully designed to only work with applications modified to take advantage of new services and permissions (reasoning provided in § 4.3 and § 6.1.1); AdDroid required a trivial amount of rewriting for the application developer in order for their applications to run on AdDroid. Second, AdDroid's use is not intended to track, modify, substitute, or block user data from leaving the device. We designed AdDroid to allow the application, say from a trusted developer, to function normally with unfettered access provided by the user-approved permissions upon application installation. AdDroid does, however, attempt to prevent unauthorized data exfiltration via alternative mechanisms. Third, AdDroid does not work with existing Android permissions, but requires a modified AOSP build with additional, limited permissions designed for 3PLs. Finally, in regards to items two and three, AdDroid's goal is to prevent data exfiltration by decoupling third-party permissions from the host application. Additionally, AdDroid introduces new, limited permissions for use by 3PLs and by denying 3PLs direct access to sensitive, private data. More specifically, AdDroid is a proof-of-concept implementation that targets the 3PLs provided by advertising networks. This approach is perhaps sufficient for now, but it is only part of what should be the final solution. TaintDroid and AppFence's purpose is, by design, to prevent the exfiltration of sensitive data, yet they fall short in that they offer no enhancements to application developers who wish to protect their customers from untrusted 3PLs. This work addresses the latter problem.

Generally speaking, TaintDroid and AppFence attack the problem through problematic analysis to detect security issues and privacy-violating leakage of users' data. This type of approach is both expensive, error-prone, and user unfriendly. Though the authors state that the overhead their systems impose is acceptable, today's smartphones are rapidly becoming more and more power hungry where dual- and soon-to-be quad-core processor devices dominate the performance segment. The problem arises with the fact that battery technology lags far behind the increase in demand for power. Any extra demand on an already limited source of power will be further limiting. Modern taint tracking techniques are not perfect and the system will mark an unknown number of data packets as tainted when in fact they contain no privacy-violating data, and vice versa. In turn, "killed" data packets will negatively impact application functionality as developers will likely not have such systems in mind when coding their applications. Additionally, TaintDroid and AppFense do not propagate taint via logic flows. The general consumer user base of smartphones will find these systems difficult to use. One cannot assume that the average user, upon receiving a privacy-violating notification, will be able to determine what exact component of their system is trying to leak information. The system leaves it up to the user to resolve which application or 3PL attempted to leak information, what information it leaked, and to whom it intended the information. Conversely, Apple's iAd provides a more balanced approach of usability and user control. AdDroid is a new, programatic approach to the same problem that cleanly separates the host application permissions from 3PLs.

4 Design & Architecture

Currently, a developer wishing to integrate 3PLs into their Android mobile application must include opaque proprietary binary code into their codebase. Worse yet, due to the design of the Android permission system, developers must in essence overprivilege their applications by requesting dangerous permissions, such as full network access, solely for the functionality of the bundled 3PLs. The AOSP requires significant functional changes and minor code changes in order to overcome the deficiencies in the current system. This section covers the proposed design changes.

4.1 Concepts

The AOSP employs the principle of least privilege to mobile applications at the application-level. This is beneficial in that individual applications are only granted the permissions necessary for them to run; the permissions granted to one application have no effect over any of the other applications. Privilege separation at the application level for smartphone applications has proven helpful and in a number of instances, sufficient. Refer to the motivation behind this work in § 2 for reasons why Android's current permission system is inadequate.

One of the principle ideas behind this work is to apply the principle of least privilege one layer deeper than presently implemented. Ideally, we would like the ability to assign a minimal set of permissions to a host application while granting an independent set of minimal permissions to each of any bundled 3PLs. Effectively we want to treat the host application and each of any 3PLs as separate entities. Android presently lacks the capability to separate the privileges granted to a host application from those assumed by 3PLs. Thus, if an advertising-supported host application requires read access to a user's contacts list, Android should not automatically grant the 3PL the same access to the user's privacy-sensitive data. If Android treated 3PLs and host applications as separate entities we would be able to pull out the functionality that

3PLs provide and incorporate them into the AOSP.

Given this restriction, one must place the 3PL components in a service separate from the application. One possible design involves running 3PL services as a separate user application, something 3PL developers themselves possibly supply. Existing permissions would then control access to the functionality these service-oriented applications provide. However, this design is deficient because it cannot guarantee how the services behave with user data. To ensure proper use of user data in Android, we need to integrate the services directly into the AOSP.

First, Google could incorporate a generic analytics engine into the AOSP and make it available to application developers. The developer could select the preferred third party to receive the statistics data relayed by the system. Second, following in Apple's footsteps with the release of iOS 4.1 and the introduction of Game Center [3], Google should take the initiative and build in their own social gaming network API. Third, Google should integrate an advertising API into the AOSP so that application developers can monetize their applications without the need to blindly-trust advertising-network-provided 3PLs. Apple's iAd [4] provides similar functionality for the iOS platform. Figure 7 shows a high level view of the AdDroid design.

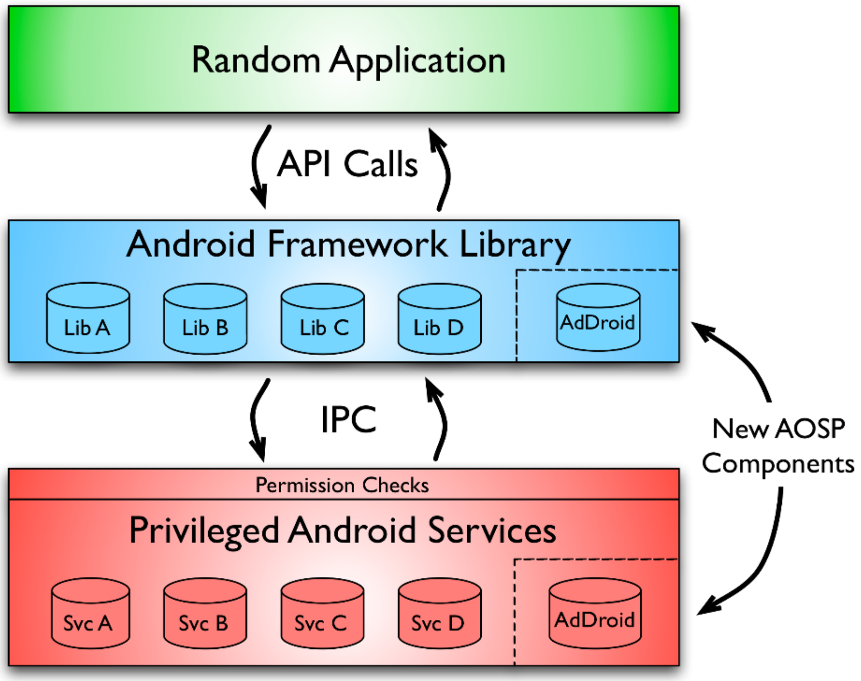


Figure 7: The new AOSP design showing the integration of AdDroid at the Framework and Service levels.

Each of these services need not only support Google-branded products, but they will also have built-in support for numerous other third parties. An application's request for an advertisement would specify to which third-party advertising network AdDroid should relay the request. Advertising networks could submit minimalist applications to the Market which contain configuration data for their particular servers. When a user launches an advertising-supported application for the first time, Android could detect if the named advertising network's configuration files are present and if not search for the appropriate application (by namespace) in the Market. Android already provides a similar functionality. For example, when a user first enables the text-to-speech functionality on a device the system automatically redirects the user to the SpeechSynthesis Data Installer (i.e., Pico TTS) application in the Market that enables said functionality [51].

Finally, the integration of 3PL APIs directly into the framework enables Google to introduce new permissions designed specifically for 3PLs. Applications should request the new 3PL permissions rather than the dangerous traditional permissions; Android would then act as an intermediary upon serving the application's requests. A prime example of this would be to have a lesser-privileged form of Internet access. Again looking at advertising networks as our example of 3PLs, when a user grants the new Internet permission to the 3PL, Android would step in and constrain it to only communications required to fulfill its role. In AdDroid, we achieved this property by having Android fulfilling the role of the advertising SDK by receiving the advertisement request from the application and returning only the advertisement image retrieved from the specified, pre-approved third-party advertising network. Another useful permission would be the ability to allow Android to relay to an approved advertising network the device's geolocation for appropriate targeted advertisements. The purpose of such permissions is, for instance, to allow for targeted or non-targeted advertisements in applications while isolating the application from privacy-sensitive user data.

4.2 Goals

The enhancements proposed in the previous section comprise only a subset of the necessary changes needed to realize the full potential of an ideal solution. This work is a large step in the right direction. The guiding principles of this work have helped to define this work and determine where we focused our work. Several of the key principles, or goals, include minimizing the burden to application developers and consumers, improving privacy controls, having independent controls for 3PLs, and reinventing the permission-system UI.

The decision to place the majority of the burden of change upon Google is important because this means that we can minimize the change for consumers and application developers. A smartphone platform without the support of application developers and acceptance of consumers has little purpose in the competitive smartphone business. The AOSP, even though it is open source, is virtually a Google-only platform they offer for use. Given this, along with the fact that they are the owner and primary contributor to the project, Google should find the incentive great enough to better their product and increase it is competitiveness in the smartphone space. With a minimal number of changes apparent to application developers, they should be able to make their application compatible with the system on the order of minutes; perhaps applications more tightly integrated with 3PLs will require on the order of hours. We converted a simple advertising-enabled demonstration application that we wrote from the current system (with bundled advertising SDK) to the new AdDroid system with less than ten lines of code (LoC). Similarly, most of the changes should remain transparent to the end-user and the learning curve of the new system should require a similarly short time adjustment period.

The primary goal on the user side is to improve controls over information sharing and privacy. Privacy controls in the AOSP are minimal and the level of control they impart to the user is likewise minimal. The archetype of user control would allow users to select, at a sufficiently fine-grained level, which parts of their data they will share with whom. As in one study [11], the author proposes that users should be able to control what data they share with others along with how often and the types of advertisements that they see. This comes at a cost to the user due to the added complexity. AdDroid is a step in the right direction, but more work is still required to realize this goal. The mechanisms should also improve the overall security of the platform by eliminating the need to bundle opaque, binary-only SDKs. Lastly, the revised system furnishes new less-dangerous permissions for any third-party functionality while brokering requests to prevent access to privacy-sensitive user data. AdDroid is able to broker requests from third parties because the 3PLs will not have direct access to user data, they must request it from the AdDroid service.

The primary architectural change is that 3PLs have user settable controls and permissions. Users would then be able to adjust settings to control which 3PLs have access to which resources. The new system should not limit these settings to a single global setting but should allow for autonomous settings, one per third party or group thereof. One possible scenario is a user wishing to share his or her geolocation with the Millennial Media advertising network while denying it to all others. The user would be able to grant geolocation access to Millennial Media and deny access to a single group comprised of all remaining advertising networks. Similarly, others may share their contacts list with the OpenFeint [44] mobile social gaming network to invite friends but share it with no one else. The improved controls would also allow users to create more advanced rules such as sharing fine location with one group of entities, no location information with another group, and coarse location with the remaining. Settings need not be permanent; users should be able to modify them as needed. AdDroid does not currently support this level of control as it still lacks a complete set of new permissions and 3PL APIs. We also have yet to implement the ability to modify permissions post-installation. There are, however, two implemented examples that we can model this feature after: LBE Privacy Guard and CyanogenMod 7. We hope to address temporary and post-installation permissions after we implement the foundational measures in AdDroid.

Adding to and extending the number of controls available comes at a burdensome cost to users. Users will end up lost if required to navigate in and out of extensive, multi-layered menus. Furthermore, copious lists, checkboxes, and a plethora of ~~piñatas~~ options can easily overwhelm and confuse users, causing more harm than good. Facebook has suffered from these UI problems with their privacy and sharing settings [48]. To support this work's goals, Google must reinvent the permission system UI. Android must present a clean, simple mechanism that the general user population can easily understand. Moreover, the new UI needs to allow users to see the growing number of options in a clean, compact space.

4.3 Non-Goal (!Goal)

In addition to the goals previously discussed in § 4.2 there is an auxiliary goal of something I purposefully do not attempt to accomplish in the final solution. Many previous and related studies have attempted to preserve backwards compatibility or designed to preserve the ability to run unmodified applications, but this is not a goal I share with others—hence !goal. Due to the importance of this matter I feel that it deserves its own section for special attention.

By not maintaining backwards compatibility (i.e., the ability to run unmodified applications) the end result is superior to those that do. Part of the reasoning behind this decision lies in the fact that the proposed changes to the AOSP require a major modification to the system, and this change must be visible to application developers. If it is visible and requires changes on their part, they must first understand the new system before modifying their applications. By requiring application developers' understanding before implementing any changes will increase the effectiveness of the new mechanisms. Additionally, if Google markets the new version of the AOSP as a more secure platform, as they should, users will assimilate this expectation. By allowing unmodified applications to run, users will believe they are operating in a more secure environment when they are not.

This introduces the challenge of verifying whether applications contain 3PLs or are utilizing the built-in API. One easy first step would be to check the namespaces of the code bundled with the application and verify that the application includes only the author's namespace. A system like Stowaway [23] would be useful after the necessary modifications. Another option would be to implement code analysis techniques and determine whether the system detects known third-party code in the application.

It is true that there will be an unknown number of developers who will not find incentive enough to make the necessary changes and some (legacy) applications may never make it to the new system. Nevertheless, many developers have the resources available to rewrite popular applications and make them available on the enhance platform. To their advantage, our first goal is to minimize the burden on developers and the amount of time required to port applications to the new platform. The increasing activation rate of Android-based devices should be large enough to incentivize the majority of application developers.

5 Implementation

Implementing the complete set of proposed changes that this master's covers would take considerably more time than available. For this purpose we determined what would be the most important first steps and decided to implement those in a proof of concept. We decided that it was best to first address the overprivileging associated with advertising network 3PLs we discovered in § 2.6. We present AdDroid, our proof-of-concept implementation. The AdDroid userspace library required 560 new source lines of code (SLOC), and the new system service required 255 SLOC.[3] Additionally, we modified or added 66 SLOC to the existing system to add support for the framework, system service, and new permissions.

5.1 Fixing Advertising-Network 3PLs with AdDroid

The primary purpose of AdDroid is to eliminate the need to overprivilege advertising-supported Android applications with three superfluous dangerous permissions: `ACCESS_COARSE_LOCATION`, `ACCESS_FINE_LOCATION`, and `READ_PHONE_STATE`. To accomplish this feat we added three new major components to the AOSP: (1) a service, (2) an API to utilize this service, and (3) new permissions to control access to the new API.

5.1.1 The New AdDroid Service

Any complete solution to the problem will involve isolating advertising components from the main application. The new component would then impose permission checks on access to the advertising service. To enforce the separation requirements, we needed to place the advertising component in a service separate from the application. We considered several possible designs, two of which are worth noting.

The first design involves implementing the advertising service as an independent user application with two possible sources: a generic application provided by Google that supports multiple third-party advertising networks, or applications provided by the advertising networks themselves. We believe this design to be sub-optimal since it does not enforce guarantees on the use of data by the application level component. The only way to ensure proper use of user data and enforce proper permission checks is to integrate the system component performing network connectivity and providing user components directly into the AOSP. The second design does exactly this: it implements the same functionality as the first but places the service in the AOSP, rather than an application.

Our decision was heavily influenced by the current AOSP design. Android developers implemented several important features of Android (e.g., the keyboard) in the AOSP as system services. For this reason, and for wanting to keep true to the AOSP design and architecture, we elected to built a new service directly into the AOSP. This enables Android to have better control over the components as it is running as a system service rather than a user-level application. It also provides us with greater functionality as we are not limited by user-level access controls and the virtual machine.

The new AdDroid service is responsible for relaying advertisement requests, and the related information, to the appropriate third-party advertising network. This service ensures that the application has the

[3]These totals include blank lines and comments.

correct advertising permissions and will perform all necessary network and location based operations. AdDroid will also forward location information with the advertisement request if the user grants the appropriate permission. In response, the advertising network supplies the AdDroid service with the requested advertisement content (e.g., advertisement image), which it then transfers to the application that originally issued the advertisement request. So applications can only request advertisements, targeted or generic, and give user attribute information while the advertising service can only give back bitmap images from predetermined advertising network services. We built AdDroid with support for multiple advertising networks instead of Google-only advertising services, as this would otherwise cause possible legal trouble for Google by having an unfair monopoly of advertising revenue on the Android platform.

5.1.2 The New AdDroid API

Applications access the advertising functionality provided by the new service through the AdDroid API. Applications interface with the AdDroid API which we integrated directly into the core Android framework. The API then communicates a request for advertisements to the new advertising service running as part of the core Android privileged services.

In order to build the AdDroid API we began with the most popular advertising network API, AdMob, per § 2.6. After implementing support for AdMob we looked at the remaining advertising networks and generalized our implementation to support the others. Specifically, we expanded our API to also support the Millennial Media advertising network, and are in progress of implementing support for InMobi. We purposefully skipped the second most popular advertising API, AdSense, since Google has deprecated the service and encourages their customers (i.e., application developers) to migrate to the newer AdMob service. The generalization step was straightforward and rather simple as independent advertising networks appear to function quite similarly or they happen to already model their API after Google's. Including applications currently using AdSense, as their developers should be moving to AdMob in the near future, our API supports 95.7% of the advertising networks used in the top free applications per our case study.

Our AdDroid API currently supports the core functionality of these APIs: supporting advertisements, listeners, layout behaviors, and statistical tracking data. AdDroid, however, does not currently support interstitial or video advertisements. Nevertheless, we do not believe there to be any technical limitations in supporting these advertisement types. As do all the studied advertising networks, the AdDroid API can send user attributes along with an advertisement request. The host application or 3PL supplies the user attributes through data they have access to or through manual user entry—in an exercise application a user might enter his or her age, gender, and weight to calculate burned calories, so the application can now use these attributes when requesting advertisements. Table 8 shows the user attributes AdDroid currently supports; we derived these from common attributes among the studied third-party advertising networks. For compatibility and to offer similar functionality, AdDroid provides built-in support for these attributes.

One major difference between the AdDroid API and existing advertising network's APIs is that when requesting an advertisement, application developers need to specify which advertising network they wish to use along with their unique ID for that particular advertising network. Advertising networks use unique IDs to determine which application developer is responsible for the advertisement impression or advertisement click and to pay them accordingly. One added benefit to implementing AdDroid as a generic advertising service is that developers can change their desired advertising network with modification to only one line of code. They can easily make the change without the need to learn a new API, bundle a new binary-only SDK, and modify multiple lines of advertising-network-specific API calls. Additionally, they can make the necessary changes with minimal debugging. Another benefit is, if application developers so desire, they may now use multiple advertising networks simultaneously.

• birthday	• income level	• sexual orientation
• ethnicity	• level of education	• political orientation
• gender	• location	• zip code
• has children	• martial status	• arbitrary attributes

Table 8: Common user attributes tracked by advertising networks that AdDroid supports. Note: Each advertising network we looked at had support for passing arbitrary data attributes and keywords, in addition to these built-in attributes.

We imagine advertising networks can submit applications containing configuration files to communicate with their servers to the Market. These applications would not request any permissions as they contain only the configuration files; the host application would be the one to request the appropriate advertising privileges. The installation process of the host application would then direct users to the appropriate package for download. Being able to update the configuration files avoids the inflexibility of integrating such data directly into the OS. Some might argue that the list may go out of date or that the advertising networks might change, but we argue that if the device has Internet access to download a new or updated application then they will have access to download the latest configuration files.

Figure 8 shows sample usage of the AdDroid API through a demo application that we wrote. This code provides a simple example of how application developers set AdDroid properties, instantiate advertisements, and send requests to the API. See Appendix A for a more detailed description of the AdDroid API.

5.1.3 The New Permissions

With the AdDroid service and API directly integrated into the AOSP we added two new advertising-based permissions to the existing set of Android permissions. By doing so we can leverage the existing capabilities of the AOSP for permissions checks and enforcement. The new permissions are `INTERNET_ADS` and `TRANSMIT_LOCATION_ADS`, one is non-dangerous and the other dangerous, respectively. An application wishing to integrate advertisements should request AdDroid's `INTERNET_ADS` or `TRANSMIT_LOCATION_ADS` privileges, and no longer needs to request network, location, or phone state directly. We chose to implement the `INTERNET_ADS` permission as non-dangerous because it is a limited form of Internet access: the application cannot create network sockets with this permission alone. Conversely, we chose to implement the `TRANSMIT_LOCATION_ADS` permission as dangerous because we feel that geolocation is privacy-sensitive; the user must grant this permission to the application in order to receive targeted advertisements, otherwise the AdDroid service will not sent the location information with advertisement requests. In AdDroid, the Android system itself mediates all security-sensitive operations, thus isolating the application and eliminating the overprivileging concern. Furthermore, the application can not make arbitrary network requests, nor can it gain access to location information. If utilized properly, this system could have the added advantage of training users that advertising-based applications do not require general Internet network privilege, thus raising awareness to overprivileging.

```java
public class AdDroidDemo extends Activity {
  // Called when the activity starts or restarts
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Specify the add type (e.g., banner, interstitial, video)
    AdProperties props = AdProperties.BANNER;

    // Specify the ad network (e.g., AdMob) and the associated unique ID
    AdNetwork network = new AdNetwork(AdNetwork.NetworkType.ADMOB, "ADVERTISER_ID");

    // Instantiate the view for the ad
    AdView adView = new AdView(this, props, network);

    // Lookup your LinearLayout assuming its been given the id "main_layout"
    LinearLayout layout = (LinearLayout)findViewById(R.id.main_layout);

    // Add the AdView to the layout
    layout.addView(adView);

    // Instantiate the ad request
    AdRequest adReq = new AdRequest();

    // Request and load the ad
    adView.loadAd(adReq);
  }
}
```

Figure 8: A complete AdDroidDemo activity showing sample usage of the AdDroid API.

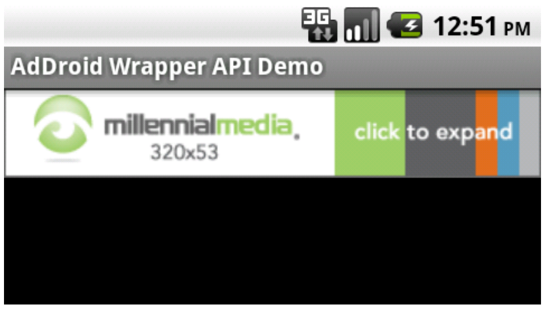


Figure 9: AdDroid wrapper API prototype demo fetching a demo advertisement from Millennial Media.

5.2 Two Versions

We have built two prototype implementations in order to demonstrate AdDroid. The first is a limited, user-level implementation that shows the flexibility of the API in supporting multiple advertising networks in an existing AOSP build. The second is a fully integrated AOSP branch showing how Google might implement an AdDroid functionality into their Android platform.

5.2.1 AdDroid Wrapper API Prototype

In our first proof of concept we built an AdDroid wrapper to interface with existing advertising network 3PLs. Also, to show the flexibility of the AdDroid API we built a user space prototype that implements the full AdDroid API. We took one of the popular advertising network's SDKs and compiled it with a modified version of an existing open source application. We modified the application to display banner advertisements and included our wrapper; rather than directly making the API calls to the advertising SDK we called our wrapper API which in turn made the appropriate calls.

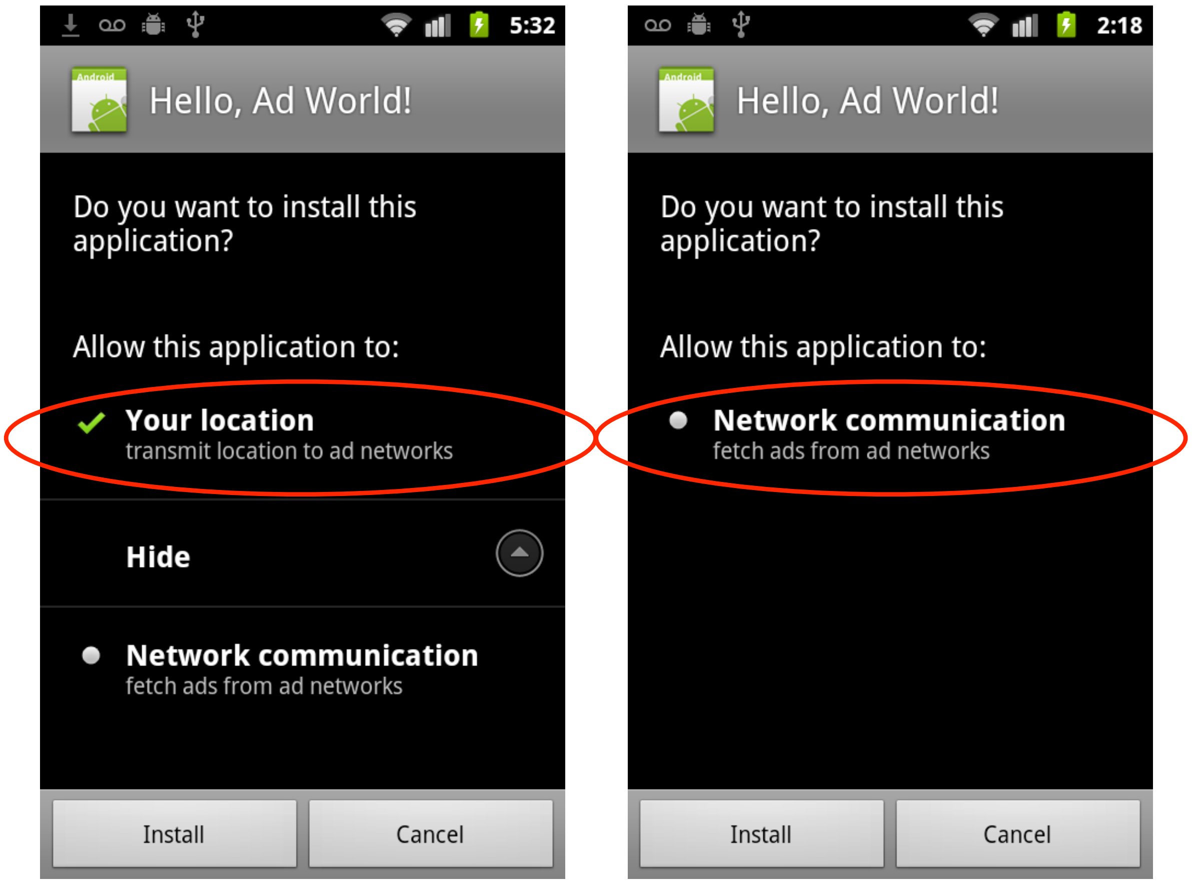
This API demo directly incorporates the advertising network SDK and runs the AdDroid library as an integrated user library. The demo does not enforce privilege separation and still suffers from overprivileging, since we integrated the advertising SDKs and AdDroid library directly into the user-level application. This behavior is necessary since in order to leverage the binary-only advertising network SDKs we had to integrate the code directly into the application. The source code for our AdDroid library and demo application consists of 450 and 64 SLOC, respectively.[4]

Figure 9 contains a screenshot of our demo application running, which has retrieved a demo advertisement from the Millennial Media advertising network. The screenshot looks quite uneventful, but our wrapper API generates the same output as does the standalone SDK. The AdDroid API abstracts away use of the advertising SDKs, and the demo application interacts only with AdDroid. This demo application is able to fetch advertisements and exercise the functionality of the AdMob and Millennial Media advertising networks.

5.2.2 AdDroid AOSP Integration Implementation

Our core prototype implementation is the fully integrated AdDroid AOSP prototype. The full AdDroid prototype takes the form of Figure 7. The core feature changes we implemented include the two new permissions, the extended framework API, and a new privileged advertising service integrated as a system

[4]Lines exclusively consisting of comments or a single closing curly brace were not counted in this number.


Hello, Ad World!

Do you want to install this application?

Allow this application to:

✓ **Your location**
transmit location to ad networks

Hide

● **Network communication**
fetch ads from ad networks

Hello, Ad World!

Do you want to install this application?

Allow this application to:

● **Network communication**
fetch ads from ad networks

Figure 10: Installation screens of two applications requesting the new AdDroid advertising permissions. On the left, the green check mark signifies a dangerous permission and any non-dangerous permissions, if at least one dangerous permission exists, reside under the Hide menu. On the right, the application only requests non-dangerous permissions.

process. This design supports our desired full privilege separation. Additionally, it removes overprivileging with respect to network access and location based information.

We made a custom build of the entire AOSP which we compiled and flashed onto a Nexus S development phone. To test the functionality of our system, we wrote a simple application called "Hello, Ad World!" and requested the new permissions. Figure 10 shows the new permissions as seen from the applications list in Android. As seen in the image on the right, if an application wants to display non-location specific advertisements the developers must request the non-dangerous `INTERNET_ADS` permission. If the developer would like to display location specific advertisements they must also request the dangerous `TRANSMIT_LOCATION_ADS`, as seen on the left.

Though AdDroid fully supports AdMob and Millennial Media, we chose to test our implementation with our own generic advertising service running on Amazon's EC2. We did this because we did not wish to reverse engineer the protocol or the binary-only advertising network SDKs due to possible intellectual property and copyright issues. Additionally, to provide proper privilege separation we would need to split the SDK to run partially as a privileged system service and partially as a user space library. We could not perform such a split without reverse engineering the binaries. Our generic advertising service has similar behavior to AdMob and Millennial Media, operates through the AdDroid API, provides location specific advertisements, and has similar functionality.

Permission assignment in a completed AdDroid system would appear similar to Figure 11. For example, the host application is a multiplayer boxing game. The application developer requests the `INTERNET` permission for the host application to enable simultaneous multiplayer gameplay. The application developer would also request the appropriate third-party permissions to enable additional functionality (advertising, a gaming network, and an analytics engine). In this example, the gaming network has access to the user's contacts and the ability to send SMS so the user can invite his or her friends to a match. It also has Internet access, limited only to the gaming network's servers, to report high scores and achievements. No other 3PL, not even the host application, has access to the user's contacts as AdDroid acts on behalf of the application. Of all the interested parties, only the advertising network can determine the user's location, for location-specific advertisement targeting.

In a traditional Android system, the application and all 3PLs would receive all the requested permissions. In turn, they all gain full access to the user's privacy-sensitive data (contacts, location, and SMS) with the ability to send the data to arbitrary destinations (via full Internet access). With AdDroid, the effective permissions for the application is simply the `INTERNET` permission. The principle of least privilege constrains each party to the minimal set of permissions required for its functionality.

5.3 Barriers to Adoption

Given that this is a work in progress a few of the goals have yet to be realized. In its current incarnation there are a few limitations. First is a matter of privilege separation. Our desire is to eliminate the need to blindly bundle 3PLs by providing new third-party APIs as a substitution. For the few cases where 3PLs are still needed, we will fully separate and treat 3PLs and host applications as separate entities with their own independent permission sets. Currently, if an advertising-supported host application like Skype, for example, requests the `READ_CONTACTS` and `INTERNET` permissions AdDroid cannot prevent the 3PLs from reading the contacts, creating a network socket, and then sending the data to an arbitrary host. AdDroid still requires rewriting the existing permissions checks to verify the caller and their permissions (e.g., host application versus 3PL). We can address this by a few simple modification options. Android can look at the calling namespace to determine the caller. Additionally, Google can make a simple modification to the compiler where calls to 3PLs (e.g., SDKs) receive a 3PL tag. Finally, a form of code introspection should be able to determine the difference between the host application and the 3PL.

The second limitation is that AdDroid does not sanitize or anonymize data sent to third parties. AdDroid is capable of doing so since the OS marshals all the data through the new AdDroid service by use of the API, but we have yet to implement this feature. However, by coupling AdDroid with TaintDroid or AppFence we can control the exfiltration of privacy-sensitive user data. Sanitization might be necessary because as do many advertising-networks' SDKs, we built in support to send arbitrary data with the advertisement request. This feature is theoretically used for advertisement targeting by specifying user attributes not currently supported via the API.

Third, in the new system we would like to ensure that application developers are not purposefully or unknowingly including unnecessary 3PLs into their applications. By doing so they compromise the integrity of the application, and possibly the system. In these situations we can rely on analysis techniques (e.g., Kirin [21]) or advanced runtime mechanisms (e.g., TaintDroid [20], AppFence [29]).

Fourth, our current implementation exhibits limited 3PL functionality. AdDroid currently only works for advertising networks for which we built in support. We are limited by the fact that advertising networks supply their SDKs in closed source, binary-only form. However, the final solution will incorporate a stan-

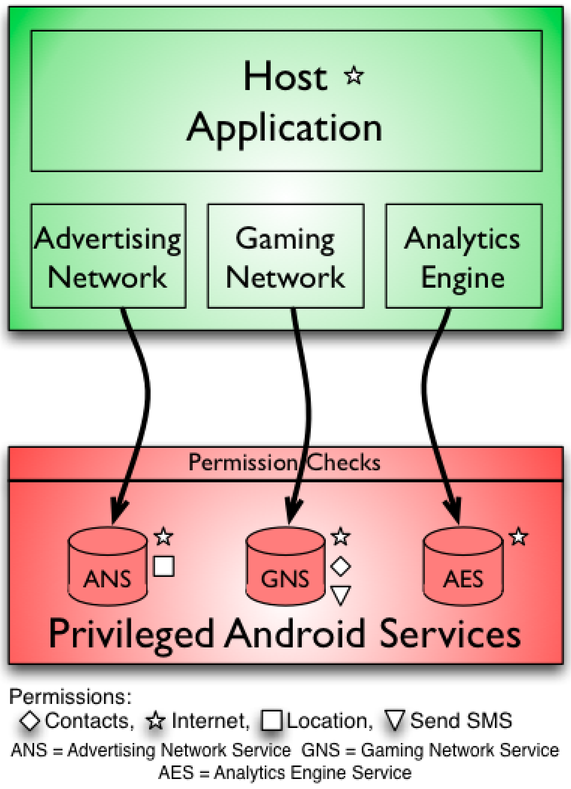

Host ☆
Application
Advertising Network
Gaming Network
Analytics Engine
Permission Checks
ANS ☆ □
GNS ☆ ◇ ▽
AES ☆
Privileged Android Services
Permissions:
◇ Contacts, ☆ Internet, □ Location, ▽ Send SMS
ANS = Advertising Network Service GNS = Gaming Network Service
AES = Analytics Engine Service

Figure 11: Diagram showing how permissions are assigned in the new system. The host application requires no permissions, but requests the appropriate third-party permissions to support the advertisements, the gaming network, and the analytics engine. The host application nor any of the libraries have direct access to the

dardized API, one with which advertising networks will make their servers compatible, and with support for the dynamic configuration file. Additionally, only newly compiled applications that utilize the new permissions and API benefit from the enhanced features that our AdDroid build offers to the system. Furthermore, AdDroid currently only supports one of the three major 3PLs we aim to target, yet we see no complications with adding support for gaming networks and analytics engines. We accept these limitations as AdDroid is a proof of concept.

And lastly, we hope to make the source code and further details available in the next few months.

6 Discussion & Challenges

In this section I present the two most likely deployment scenarios for the completed AdDroid system. Also, I discuss some adoption challenges that either of the two scenarios might face.

6.1 Deployment Plan

Throughout this master's I have assumed that Google would be the one to implement the new AdDroid system. Nevertheless, in an alternate scenario a third party could build their own enhanced version of Android since the AOSP is in fact open source.[5] Thus, the two scenarios are one in which Google implements the changes and the other where a third party takes the initiative.

6.1.1 With Google

This scenario would yield the best outcome for two primary reasons: (1) Google knows Android better than anyone else, and (2) all Android users receive the benefits of the overhauled system. Even with Google implementing the enhancements to the AOSP there are varying degrees of changes that Google may choose to pursue.

The first possibility incorporates the largest number of changes and requires a major rewrite of the AOSP and Android permission system. Due to the drastic changes and a new permission system Google would need to deploy this as a major OS update, say Jelly Bean (as Android is currently at Ice Cream Sandwich). In this update Google would abandon the do-or-die, all-or-nothing approach with respect to application permissions. Since this version introduces a possibly large number of new permissions, some for host applications and others for 3PLs, I strongly recommend a revised UI to prevent user overload and confusion. As noted earlier, Google should drop backwards compatibility as this may create a false sense of security by allowing unmodified applications to run without utilizing the improved privacy and security mechanisms. Additionally, unmodified applications would not request the new, finer-grained, less dangerous permissions and remain overprivileged. Moreover, the host application and 3PLs would not have proper privilege separation. A major overhaul would prevent this situation from happening. Google should also incorporate the ability for users to change permissions post-installation, if not for the host application then at least for the 3PLs. At a minimum, users should have the ability to opt-out of interest-based and location-specific advertisements and receive generic advertisements instead. Consumers should be in control of their privacy and have the ability to choose between generic or targeted (i.e., location-specific or interest-based) advertisements [11]. In the current model the application developer carries the power to make this decision, which may conflict with the consumers' desires.

[5] A few Google-branded applications, including the Market, remain closed source. Google awards these applications to OEMs only after the OEM meets certain criteria.

Second, a less involved implementation might only implement new permissions for 3PLs. This scenario would still require a large number of changes but maintains the current permission system that users are already familiar with, but extended to 3PLs as well. The disadvantage of this approach, as opposed to the first, is that application developers remain in control over privacy-sensitive permission choices. If utilized properly, however, users still benefit from privilege separation. This approach would still require a new major version release and a revised permissions UI.

The third option involves a minimal number of changes. Since Android would treat 3PLs no differently than the host application, it is possible for Google to release this revision as a minor version update. Google should introduce new, lesser-privileged permissions to help eliminate the abundance of overprivileged applications. A possible new permission would be a limited form of Internet access that application developers can use to specify a list of hosts with whom their application can communicate. By electing this option Google gives up the isolation benefits AdDroid offers to the platform.

6.1.2 Without Google

If Google does not change their Android platform, a third party could make an improved version for their own product, or set of products. Amazon, with their recently released Android-based Kindle Fire, would be a prime example where a third party initiative would be practical and economical. To make the effort economically worthwhile, the developer would have to expect wide adoption and to sell a large number of devices. Preliminary reports have surfaced that Amazon sold 250,000 Kindle Fires in the first five days of pre-orders with 2.5 million projected sales by release day on November 15, 2011 [6]. Initial sales forecasts also highly favor the Kindle Fire with 5 million units sold in the fourth quarter [22] and projected sales of 12 and 20 million in the years 2012 and 2013, respectively [58].

Besides sales numbers, another major advantage Amazon has over other companies is that they have exclusive control over the hardware and the Kindle platform, much like Apple does with its products. Additionally, both companies' products target their own closed ecosystem of content and applications. This enables Amazon to create their own customized kernel and branch of the AOSP, complete with 3PL privilege separation and new permissions, without creating a division among the rest of the Android platform. On the contrary, they can enhance the Android ecosystem for generic Android devices as well as their Android-based Kindle Fire.

One way Amazon is already doing this, again with the Kindle Fire, is through the Amazon Appstore for Android [1]. The Amazon Appstore is an optional third-party store for applications available to all Android devices that competes directly with Google's Android Market [2]. On the Kindle Fire, Amazon ships the device only with their Appstore, which is the only approved method to obtain applications. The Amazon Appstore differs from the Android Market is two key aspects: (1) it offers applications designed exclusively for the Kindle Fire, and (2) Amazon vets applications and application updates before they make them available for download [54]. To vet an application means to certify that the application behaves as intended and does not introduce security and privacy risks to the user. This shows that developers are already willing to port existing applications over to the Kindle Fire. The second feature improves the security of the entire platform. Amazon and Apple provide this feature, however, Google does not.

Amazon's advertising model would work well with the enhancements AdDroid provides to the Android platform—in particular, following their model of Kindles available with "Special Offers." First, Amazon is already in the business of selling and approving advertisement content for the Kindle platform. Second, AdDroid's ability to update which advertising networks it approves allows Amazon to remain in control of advertising on their Kindle platform.

Google only makes some of its Google-branded applications, such as the Android Market, available to third-parties if they follow certain guidelines. It is possible that Google would not allow Amazon to run the Android Market on their Kindle Fire for competition reasons. The Kindle Fire's success, however, is not dependent on the inclusion of the Android Market on their Android-based devices, as the Amazon Appstore is growing in popularity and has attracted enough attention to incentivize application developers to release their application(s) exclusively on the Appstore. The Appstore might end up with a subset of applications available in the Android Market, but the more popular applications have been and will continue to appear in the Appstore as adoption continues to grow. Amazon would likely not worry about offering a smaller collection of applications because their primary focus is on available content (e.g., books, movies, music) rather than applications.

The biggest disadvantage to Amazon would be the cost of incorporating AOSP updates into their customized build. Users might not carry the same expectations with the Kindle platform, in terms of updates, as they do with Android-based smartphones and tablets. To their advantage, Amazon is not targeting the Kindle Fire as a general-purpose device. The device is specifically targeted for content consumption and shopping, something that does not change much from one release to another. Perhaps Amazon will utilize the latest build available at the time of release for each new device.

6.2 Challenges

Throughout this master's I have presented several challenges one would face in implementing and introducing an enhanced version of Android in the market. In this section I re-emphasize key challenges as well as introduce some previously unmentioned ones.

6.2.1 Application Developers

Without the support of application developers, modern smartphone platforms would be boring and largely dissatisfying. The success of any modern platform depends on the easy of development and the popularity amongst application developers. AdDroid allows application developers to easily switch between or simultaneously support multiple advertising networks. A primary goal of AdDroid is to minimize the required changes and learning curve for application developers. We introduce no new expectations as developers are already proficient working with third-party SDKs and APIs. Additionally, we modeled the API after popular 3PLs with which application developers are already familiar. Furthermore, the majority of applications would require modification to only a few lines of code, specifically where the application makes the API calls. Finally, the strong sales numbers and forecasts for a device, such as Amazon's Kindle Fire, provides incentive enough for developers to learn the new, yet similar platform.

6.2.2 OS Updates

In order for a device to benefit from an improved version of Android it must be running the latest version of the software. However, the current state of OS updates on the Android platform is quite limited. Manufacturers recently introduced some devices into the market with Android 2.2 (Froyo), despite the fact that Google released version 2.3 (Gingerbread) available to the public on December 6, 2010. Google has since moved through version 3.x (Honeycomb) and is currently on version 4.x (Ice Cream Sandwich). The problem is that manufacturers and carriers are slow to update devices, if at all. This means that a significant majority of devices would never receive the enhanced version of Android if Google were to release it. In other words, all current and previous-generation devices remain vulnerable and obsolete. This problem is complex since the rate of Android adoption is growing and most devices have a two year lifetime (i.e., smartphones are tied to two year carrier contracts). It could take years before such benefits reach the majority of Android users.

6.2.3 Inherent Limitations

Regardless of the direction Android moves, there are inherent limitations in the platform that enable certain problems to perpetuate. First, malicious applications might try to obfuscate information and code making detection of said applications difficult. Second, it is only a matter of time after the release of a new system that adversaries find ways to run malicious code or bypass security features. AdDroid addresses these concerns by improving the privilege separation mechanisms and introducing less-privileged permissions. Finally, despite how easy a developer makes a platform to use, there will always be a group of users overwhelmed or confused by the settings at their disposal. Confused or overwhelmed users can easily nullify any privacy features built into the platform.

6.2.4 A New User Interface

An enhanced platform will only benefit users if they can utilize the augmented controls properly. If users already find the current system, with limited controls, difficult to understand then it will only become worse by adding more controls without improving the UI. Additionally, asking the general user to jump in and out of multi-level menus or read long lists of options is an unreasonable expectation. However, the ideal solution is to have that level of control for advanced and willing users while keeping it simplified for those who so desire. Android needs a new permissions UI so that its users can see a large amount of information, easily understand it, and readily determine how permissions interact one with another.

Then comes the question of defaults. It is impossible to satisfy the entire user base with any predefined set of defaults, but again it is unreasonable to expect users to make decisions on every possible level regardless of their expertise. Where developers provide defaults, it may be difficult to determine which features should be opt-out versus opt-in. For example, does a phone manufacturer enable targeted advertisements by default for an enhanced user experience or do they err on the side of caution for possible user-privacy violations and disable the feature. Some users will never bother changing defaults, so the phone manufacturer will lock their customers into a decision without them knowing.

7 Future Work

There are a number of directions this work can head, but here are a few I would like to see. First, I would like to extend AdDroid from advertising 3PLs to support the two remaining primary categories of 3PLs, namely game networks and analytics engines. These are the three most prevalent 3PLs I have encountered through my use of the Android platform. On the game network side, I would start with the OpenFeint API and begin a generalization of that as it is one of the more popular gaming networks. Additionally, OpenFeint is cross-platform, supporting both Android and iOS devices, which is helpful for extending this work to non-Android-based devices. For analytics, I would begin with Google's [26] or Flurry's [25] API and create a generalized API from that. The completed set would create a well balanced, more universal platform making the adoption of the enhanced platform more likely. I would also like to investigate how one could incorporate game and graphics engines into the platform as well.

An important concept that I would like to see explored further is the combination of AdDroid with the works of TaintDroid and AppFence. These approaches share a common goal, but attempt to solve the problem through different mechanisms. What is interesting is that the approaches are not mutually exclusive.

It is very reasonable to believe that one can unify these works to create a more effective approach to user control over privacy-sensitive data. I would also like to extend this work by examining a similar approach on other platforms. However, this could prove more difficult as Android is unique in being an open source platform.

Another interesting concept would be to study how advertising networks can preserve user privacy while still supporting targeted or interest-based advertisements. When an application or web page requests an advertisement, user attributes are regularly sent along with the request so that the advertising networks can select the most appropriate advertisement. This protocol, however, divulges user data that may violate one's privacy. To overcome this drawback we need a protocol that allows the advertisement selection process to occur on the user-controlled device rather than the remote server. One possible approach would be for the client to retrieve multiple advertisements at once and then select the most appropriate advertisement from the available set. Tracking advertisement clicks and information sharing among third parties brings in a whole other set of problems, some of which Mayer of Stanford wrote about in his blog [39].

Finally, one puts users at risk of confusion by including additional controls to have better user control over their sensitive data. In order to not overwhelm users and to better the general-user understanding of permissions I would like to develop a new revision to the permissions UI. I only briefly introduced the problem in § 6.2.4. The idea behind a new UI is to make a large number of options readily visible to users while making it easy to understand how the permissions interact at different levels (e.g., global versus application-specific, or host-application versus 3PL permissions).

8 Conclusion

Developers regularly supplement their mobile applications with functionality or advertising revenue offered by 3PLs. When developers overprivilege their applications to support 3PLs, they leave their users vulnerable to a range of risks, including the potential unauthorized disclosure of privacy-sensitive user data. We can apply the concepts we present here to eliminate overprivileging problems with advertising networks, gaming networks, analytics engines, game engines, and even to other platforms. Smartphones have become general purpose computing devices where sensitive information is readily available: contact lists, phone numbers, geolocation, financial and social data. Android does not provide users with sufficient control over their data and privacy. Our goal is to offer users proper control over their data while minimizing the burden of change to application developers and consumers. This work presents *AdDroid*, a proof-of-concept implementation that applies the principle of least privilege to mobile applications and advertising 3PLs. AdDroid uses privilege separation to isolate advertising-network functionality from host applications. We built two prototype implementations showing both the flexibility of the API as well as how Google might integrate such a system into the AOSP. AdDroid eliminates overprivileging in 44% of advertising-supported free applications through a new service, an API to utilize this service, and permissions to control access to the new API. We also discussed possible deployment plans and their associated challenges.

9 Acknowledgements

I would first like to thank Paul Pierce for the initial AdDroid concept, the overprivileging case study, and largely contributing to the implementations. I thank Anthony D. Joseph for being my research advisor and Eric Brewer for his time as second reader. I would like to acknowledge Paul's research advisors, Vern Paxson and David Wagner, for their input on the initial class project. Also, I would like to thank Adrienne

Porter Felt for helping with general and permission-related questions about Android. Finally, I would like to thank and acknowledge the support I receive from my wife, kids, and family.

References

[1] Amazon Appstore for Android. `http://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011`, 2011.

[2] Android Market. `https://market.android.com/`, 2011.

[3] Apple - Game Center - Social gaming on iPod touch, iPhone, and iPad. `http://www.apple.com/game-center/`, Nov. 2011.

[4] Apple - iAd. `http://advertising.apple.com/`, Nov. 2011.

[5] AT&T Wireless Customer Agreement. `http://www.wireless.att.com/learn/articles-resources/wireless-terms.jsp`, 2011.

[6] BROWNLEE, J. Leaked Sales Numbers Suggest Amazon Kindle Fire On Track To Outsell iPad [Exclusive]. `http://cultofandroid.com/257/leaked-sales-numbers-suggest-amazon-kindle-fire-on-track-to-outsell-ipad-exclusive/`, Oct. 2011.

[7] BRUMLEY, D., AND SONG, D. Privtrans: Automatically Partitioning Programs for Privilege Separation. In *USENIX Security Symposium (SSYM)* (2004).

[8] CHENG, W., ZHAO, Q., YU, B., AND HIROSHIGE, S. TaintTrace: Efficient Flow Tracing with Dynamic Binary Rewriting. In *IEEE Symposium on Computers and Communications* (2006).

[9] CLABURN, T. About 1% Of Google Android Apps Bad. `http://www.informationweek.com/news/security/vulnerabilities/222300435`, Jan. 2010.

[10] CLAUSE, J., LI, W., AND ORSO, A. Dytan: A Generic Dynamic Taint Analysis Framework. In *International Symposium on Software Testing and Analysis* (2007).

[11] CLEFF, E. B. Privacy issues in mobile advertising. *Int. Rev. Law Comput. Technol.* (2007).

[12] CLULEY, G. Android malware steals info from one million phone owners. `http://nakedsecurity.sophos.com/2010/07/29/android-malware-steals-info-million-phone-owners/`, July 2010.

[13] Cyanogenmod - About. `http://www.cyanogenmod.com/about`.

[14] DELEVETT, P., AND OWENS, J. C. Groupon's Big IPO Could Speed More Silicon Valley Tech Firms to the Stock Market. `http://www.mercurynews.com/top-stories/ci_19263793`, Nov. 2011.

[15] DENNING, D. E. A Lattice Model of Secure Information Flow. *Communications of the ACM 19* (May 1976).

[16] DESMET, L., JOOSEN, W., MASSACCI, F., PHILIPPAERTS, P., PIESSENS, F., SIAHAAN, I., AND VANOVERBERGHE, D. Security-by-contract on the .NET platform. *Information Security Technical Report 13* (2008).

[17] DIETZ, M., SHEKHAR, S., PISETSKY, Y., SHU, A., AND WALLACH, D. S. Quire: Lightweight provenance for smart phone operating systems. In *USENIX Security Symposium (SSYM)* (2011).

[18] EFRATI, A., THURM, S., AND SEARCEY, D. Mobile-App Makers Face U.S. Privacy Investigation. `http://online.wsj.com/article/SB10001424052748703806304576242923804770968.html`, Apr. 2011.

[19] EGELE, M., KRUEGEL, C., KIRDA, E., AND VIGNA, G. PiOS: Detecting Privacy Leaks in iOS Applications. In *Network and Distributed System Security Symposium* (2011).

[20] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *USENIX Symposium on Operating Systems Design and Implementation* (2010).

[21] ENCK, W., ONGTANG, M., AND MCDANIEL, P. On Lightweight Mobile Phone Application Certification. In *ACM Conference on Computer and Communications Security* (2009).

[22] EPSTEIN, Z. Amazon may move 12 million Kindle Fire tablets in 2012. `http://www.bgr.com/2011/11/23/amazon-may-move-12-million-kindle-fire-tablets-in-2012/`, Nov. 2011.

[23] FELT, A. P., CHIN, E., HANNA, S., SONG, D., AND WAGNER, D. Android permissions demystified. In *ACM Conference on Computer and Communication Security (CCS)* (2011).

[24] FELT, A. P., GREENWOOD, K., AND WAGNER, D. The Effectiveness of Application Permissions. In *USENIX Conference on Web Application Development* (2011).

[25] Flurry — Mobile Application Analytics — Traffic Acquisition — Monetization — iOS, Android, Blackberry, Windows Phone, J2ME. `http://www.flurry.com/`, Aug. 2011.

[26] Google Analytics SDK for Android. `http://code.google.com/mobile/analytics/docs/android/`, Nov. 2011.

[27] Google Chrome Extensions - Optional Permissions. `http://code.google.com/chrome/extensions/permissions.html`.

[28] HADDADI, H., HUI, P., HENDERSON, T., AND BROWN, I. Targeted advertising on the handset: Privacy and security challenges. In *Pervasive Advertising*. 2011.

[29] HORNYACK, P., HAN, S., JUNG, J., SCHECHTER, S., AND WETHERALL, D. These Aren't the Droids You're Looking For: Retrofitting Android to Protect Data from Imperious Applications. In *ACM Conference on Computer and Communication Security (CCS)* (2011).

[30] How to opt out of interest-based ads from the iAd network. `http://support.apple.com/kb/HT4228`, Apr. 2011.

[31] Introduction to privacy and interest-based ads. `http://support.google.com/adwords/bin/answer.py?hl=en&answer=176795`, 2011.

[32] KELLEY, P. G., BENISCH, M., CRANOR, L., AND SADEH, N. When are users comfortable sharing their locations with advertisers? In *Conference on Human Factors in Computing (CHI)* (2011).

[33] KREBS, B. Check your Facebook 'privacy' settings now. `http://voices.washingtonpost.com/securityfix/2009/12/check_your_facebook_privacy_se.html`, Dec. 2009.

[34] KROHN, M., YIP, A., BRODSKY, M., CLIFFER, N., KAASHOEK, M. F., KOHLER, E., AND MORRIS, R. Information Flow Control for Standard OS Abstractions. In *ACM Symposium on Operating Systems Principles* (2007).

[35] LBE Privacy Guard - Android Market. `https://market.android.com/details?id=com.lbe.security.lite`.

[36] LBE Privacy Guard - Homepage. `http://www.lbesec.com/lite/en/`.

[37] LOUW, M. T., GANESH, K. T., AND VENKATAKRISHNAN, V. AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements. In *USENIX Security Symposium (SSYM)* (2010).

[38] Manifest Permissions — Android Developers. `http://developer.android.com/reference/android/Manifest.permission.html`, Sept. 2011.

[39] MAYER, J. Tracking the Trackers: Where Everybody Knows Your Username. `http://cyberlaw.stanford.edu/node/6740`, Oct. 2011.

[40] MCDONALD, A. M., AND CRANOR, L. F. Americans' attitudes about internet behavioral advertising practices. In *Workshop on Privacy in the Electronic Society (WPES)* (2010).

[41] MCKEON, M. The Evolution of Privacy on Facebook. `http://mattmckeon.com/facebook-privacy/`, May 2010.

[42] MILLER, B., PEARCE, P., GRIER, C., KREIBICH, C., AND PAXSON, V. What's clicking what? techniques and innovations of today's clickbots. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)* (2011).

[43] NEEDLEMAN, R. How to fix Facebook's new privacy settings. `http://news.cnet.com/8301-19882_3-10413317-250.html`, Dec. 2009.

[44] OpenFeint - Mobile Social Gaming Network for iPhone, iPad, Android, iOS, iPod Touch. `http://www.openfeint.com/`, Nov. 2011.

[45] Permission — Android Developers. `http://developer.android.com/guide/topics/manifest/permission-element.html`, Nov. 2011.

[46] PRINCE, B. Users Hit by Malicious Ad. `http://securitywatch.eweek.com/malware/nytimescom_users_hit_by_malicious_ad.html`, Sept. 2009.

[47] RUSSAKOVSKII, A. Massive Security Vulnerability In HTC Android Devices (EVO 3D, 4G, Thunderbolt, Others) Exposes Phone Numbers, GPS, SMS, Emails Addresses, Much More. `http://www.androidpolice.com/2011/10/01/massive-security-vulnerability-in-htc-android-devices-evo-3d-4g-thunderbolt-others-exposes-phone-numbers-gps-sms-emails-addresses-much-more/`, Oct. 2011.

[48] SHARMA, H. Facebook privacy. Master's thesis, UC Berkeley, 2010.

[49] SHIELDS, T. Mobile Apps Invading Your Privacy. `http://www.veracode.com/blog/2011/04/mobile-apps-invading-your-privacy/`, Apr. 2011.

[50] SLOWINSKA, A., AND BOS, H. Pointless Tainting? Evaluating the Practicality of Pointer Tainting. In *ACM European conference on Computer systems* (2009).

[51] SpeechSynthesis Data Installer. `https://market.android.com/details?id=com.svox.langpack.installer`.

[52] STERLING, G. Google Intros New Privacy Controls For Mobile Consumers. `http://searchengineland.com/google-intros-new-privacy-controls-for-mobile-consumers-73156`, Apr. 2011.

[53] STRAZZERE, T. Update - Android Malware DroidDream - How it Works. `http://blog.mylookout.com/2011/03/android-malware-droiddream-how-it-works/`, Mar. 2011.

[54] STROHMEYER, R. Why I Get Apps From Amazon, Not Google. `https://www.pcworld.com/businesscenter/article/239270/why_i_get_apps_from_amazon_not_google.html`, Aug. 2011.

[55] THURM, S., AND KANE, Y. I. Your Apps Are Watching You. `http://online.wsj.com/article/SB10001424052748704694004576020083703574602.html`, Dec. 2010.

[56] TRUSTE. 2009 Study: Consumer Attitudes About Behavioral Targeting. `http://www.truste.com/pdf/TRUSTe_TNS_2009_BT_Study_Summary.pdf`, 2009.

[57] What Applets Can and Cannot Do. `http://docs.oracle.com/javase/tutorial/deployment/applet/security.html`.

[58] YAROW, J. Amazon Will Sell 12 Million Kindle Fires Next Year – Citi. `http://www.businessinsider.com/amazon-will-sell-12-million-kindle-fires-in-2012-taking-15-of-the-tablet-market-says-citi-2011-11`, Nov. 2011.

[59] YIP, A., WANG, X., ZELDOVICH, N., AND KAASHOEK, M. F. Improving Application Security with Data Flow Assertions. In *ACM Symposium on Operating Systems Principles* (2009).

Appendices

A The AdDroid API

Listing 1 shows our AdDroid API as found in package com.ads.addroid, abbreviated for space.

```
Interface AdListener
  void onCacheReceive(Ad ad)
  void onDismissScreen(Ad ad)
  void onFailedToRecieveAd(Ad ad, AdRequest.ErrorCode error)
  void onForcedPresentScreen(Ad ad)
  void onLeaveApplication(Ad ad)
  void onPresentScreen(Ad ad)
  void onReceiveAd(Ad ad)

Class AdNetwork
  public AdNetwork(AdNetwork.NetworkType network, java.lang.String applicationId)
  public java.lang.String getId()
  public AdNetwork.NetworkType getNetwork()
```

```
static enum AdNetwork.NetworkType
  ADMOB   MILLMEDIA   MOBCLIX

Class AdProperties
  public AdProperties(int width, int height, AdProperties.Align alignment, int refresh)
  public AdProperties.Align getAlignment()
  public java.lang.String toString()
  public static AdProperties BANNER
  public static AdProperties BANNER_BOTTOM
  public static AdProperties BANNER_TOP
  public static AdProperties IAB_BANNER
  public static AdProperties IAB_LEADERBOARD
  public static AdProperties IAB_MRECT
  public static int NO_REFRESH

static enum AdProperties.Align
  BOTTOM   TOP   UNSET

Class AdRequest
  public AdRequest()
  public void addExtra(java.lang.String key, java.lang.Object value)
  public void addKeyword(java.lang.String keyword)
  public java.util.Map<java.lang.String,java.lang.Object> getRequestMap()
  public void setBirthday(java.lang.String birthday)
  public void setExtras(java.util.Map<java.lang.String,java.lang.Object> extras)
  public void setGender(AdRequest.Gender gender)
  public void setKeywords(java.util.Set<java.lang.String> keywords)
  public void setLocation(boolean location)
  public void setTesting(boolean testing)

static enum AdRequest.ErrorCode
  INTERNAL_ERROR   INVALID_REQUEST   NETWORK_ERROR   NO_FILL   UNKNOWN

Class AdView extends RelativeLayout implements Ad
  public AdView(Activity activity, AdProperties props, AdNetwork network)
  public boolean isReady()
  public boolean isRefreshing()
  public void loadAd(AdRequest request)
  public void setAdListener(AdListener listener)
  public boolean stopLoading()
```
Listing 1: The abbreviated AdDroid API available in package com.ads.addroid.