Investigating User Privacy in Android Ad Libraries

Ryan Stevens*, Clint Gibler*, Jon Crussell*†, Jeremy Erickson*† and Hao Chen*
*University of California, Davis
{rcstevens, cdgibler, jcrussell, jericks, chen}@ucdavis.edu
†Sandia National Labs, Livermore, CA
{jcrusse, jericks}@sandia.gov

***Abstract*—Recent years have witnessed incredible growth in the popularity and prevalence of smart phones. A flourishing mobile application market has evolved to provide users with additional functionality such as interacting with social networks, games, and more. Mobile applications may have a direct purchasing cost or be free but ad-supported. Unlike in-browser ads, the privacy implications of ads in Android applications has not been thoroughly explored. We start by comparing the similarities and differences of in-browser ads and in-app ads. We examine the effect on user privacy of thirteen popular Android ad providers by reviewing their use of permissions. Worryingly, several ad libraries checked for permissions beyond the required and optional ones listed in their documentation, including dangerous permissions like CAMERA, WRITE_CALENDAR and WRITE_CONTACTS. Further, we discover the insecure use of Android's JavaScript extension mechanism in several ad libraries. We identify fields in ad requests for private user information and confirm their presence in network data obtained from a tier-1 network provider. We also show that users can be tracked by a network sniffer across ad providers and by an ad provider across applications. Finally, we discuss several possible solutions to the privacy issues identified above.**

I. Introduction

Smart phones have rapidly increased in popularity in recent years, giving rise to a booming mobile application market. Many developers release their applications for free and generate revenue from ads. To make ads more relevant to the user, *ad providers*, who decide which ads a user will see, wish to *target* a user based on the information specific to the user. The more targeting information the ad provider can acquire, the more profitable the ads will be. When an ad library embedded in a mobile application on the user's device requests an advertisement, it sends information about the device and user to the ad server, thus raising concerns about user privacy.

With the torrent of privacy-related articles in the media, both public and private sectors have been working to alleviate growing concerns with how user privacy is handled on the Internet. Advertisers under the Digital Advertising Alliance have chosen to implement "Do Not Track" features into their services; however, the functionality is largely being implemented for browser-based advertisements and it is unclear exactly how the technology will operate [34]. A number of policy solutions are being considered as well, including requiring all mobile applications to include a privacy policy detailing the personal information the app collects [39] and a more general "Consumer Privacy Bill of Rights" that is supposed to outline the rights of online users to protect their privacy [35]. We need technical solutions to facilitate ad providers satisfying these policies and to allow mobile users to verify the conformance.

We consider two different entities who might infringe on an Android user's privacy. The first entity is an unscrupulous ad provider. She might attempt to acquire more of a user's private data than is necessary for serving ads or more than is declared in the documentation of the ad library. For example, we found that several ad libraries do not mention the use of certain permissions in their documentation. However, if these permissions are available to the application using these libraries, the libraries will take advantage of these permissions to read private data. We call these *undocumented* permissions, which might elude most application developers and users (Section V). The ad provider could also try to track the user across different applications by exfiltrating device IDs. The second entity that might infringe on user privacy is a network sniffer. None of the Android libraries that we analyzed use encryption to protect ad requests. Although there are reasons for not encrypting ad requests (Section VI), it exposes user's private data to network sniffers, ranging from open WiFi access points to ISPs.

In this paper we examine popular ad providers for Android applications. First, we analyze unique privacy concerns about Android ad providers compared to in-browser advertising. Then, we evaluate thirteen popular ad providers in detail.[1] We examine which permissions they require and whether they take advantage of undocumented permissions, what private data they exfiltrate, and how they can track the user across multiple applications. We also expose vulnerabilities in these ad libraries that would allow an attacker on the network to infringe on the user's privacy (Section V-C). Based on our study, we then recommend remedies to better protect user privacy.

II. Background

A. Android

Android is a Linux-based smart phone operating system. Every Android application runs as a separate Linux user, preventing one application from reading or manipulating another's private info or application code. Android requires

[1] We also studied the Flurry and Google Analytics analytics libraries as well as the AdWhirl ad aggregator. However, since they contain no unique privacy concerns not seen in the 13 ad libraries, we do not discuss them further.

applications to specify a list of *permissions* that govern how they may access sensitive user information or use sensitive functionality[2]. These permissions must be specified in the application's manifest file in order to use privileged functionality of the device. Upon application installation, users are presented with a list of required permissions. The application will only be installed if the user agrees to permanently grant the application all of the requested permissions. Though users know the capabilities of an application at a high level, how the application uses its permissions in practice is unclear.

The lack of insight into an application's use of sensitive functionality is especially relevant on Android because of the common development practice of including third-party libraries. Developers use libraries to parse a specific file format, communicate with a popular web application, or provide advertising functionality. Libraries in Android applications run in the same process space as the application code and thus have the same capabilities. This lack of differentiation gives libraries the same privileges as application code, allowing it to read application-specific private information or utilize functionality granted by any permission the application declares.

B. Unique Device Identifier (UDID)

Android provides a number of methods for developers to obtain a unique identifier for a device [5]:

- The ANDROID_ID, a random hex string that is generated during the first boot of a device. It does not require a permission to access and is now the recommended unique identifier for Android.
- The telephony device ID of the phone (IMEI for GSM, MEID or ESN for CDMA).
- The static field *android.os.Build.SERIAL*, which also does not require a permission to access. This has only been available since Android 2.3 and does not reset when the device is wiped.
- The MAC address for the phone's Wifi or Bluetooth adapter.

In the rest of the paper we will use *unique device identifier* (or *UDID*) to represent any of the above identifiers that uniquely identify an Android device[3].

C. Online Advertising

We introduce online advertising by discussing common online advertising terms as they apply to Android advertising. For a more general definition of these terms, refer to the Internet Advertising Bureau's glossary of advertising terms [7].

- *Ad serving* is the on-demand process of choosing an ad to display when the user requests one.
- When the user's device requires an ad, it makes an *ad request* to an *ad server*, which responds with the advertisement that should be shown to the user.
- The ad server is owned by an *ad provider*, who has access to a pool of *advertisers* who pay the ad provider to disseminate their ads to certain types of users.
- A *publisher* is either a website or an application that is paid for displaying advertisements to users.
- *Targeting* is information associated with a particular user such as age, gender, or location. Targeting information is provided to the ad server in ad requests so that ads can be chosen that match the user's interests.

Web site advertisements are usually chosen dynamically when the browser renders the HTML of the web page. The web site contains an *ad tag* (typically an `<iframe>` or `<script>` tag) whose `src` field is set to the ad server. The ad tag is provided to the publisher by the ad provider to be included in the page. Once the ad tag is rendered, the browser makes a request to the ad server, which will respond with an image and a click-through URL that the user will be redirected to if they click the advertisement. The ad provider pays the publisher for displaying the ads to users, and the advertisers pay the ad provider for disseminating their ads. It is important to note that this is how ad serving is conducted when users are accessing the Internet through a web browser, both for mobile and desktop browsers. In contrast, we focus on in-application ads in this paper.

D. Advertising on Android

Advertisements for mobile applications are chosen much like they are for browsers: an ad server is queried using an ad request and the server responds with an ad to show to the user. To make the ad request, Android ad providers give developers a *Software Development Kit (SDK)* library, which provides an API for displaying advertisements that abstracts away the complexity of making an ad request, parsing the response, and displaying the resulting ad image. Further, the SDK keeps the developer from having to understand each ad provider's unique ad request format, allowing her to simply update the SDK when there is a new version, minimizing changes in application code. We will refer to the advertising SDKs as ad libraries throughout the rest of the paper.

III. Unique Privacy Concerns in Android Advertising

Here we consider three specific differences between in-app advertising using an ad library and in-browser advertising: mobile ad code is not subject to the same level of privilege separation as it is in browsers, in-app ad code has greater access to user information than in-browser ad code, and mobile devices have more consistent user identifiers.

A. Lack of Privilege Separation Between Application and Ad Code

Despite the benefits of using an ad library to facilitate ad serving in mobile applications, it gives ad providers the opportunity to run code on users' devices with the same permissions as the application that uses the library, allowing

[2]Examples include READ_CONTACTS, RECORD_AUDIO, and SEND_SMS.

[3]We define UDID for our own purposes, not to be confused with the UDID of Apple devices.

the ad providers to exfiltrate data from the device. In browser-based advertising, exfiltrating user information is difficult because of the *same origin policy* [40], which requires that code external to the user's machine cannot make requests to web pages nor access content beyond the domain from which it was served. JavaScript code provided by ad providers for a web site is isolated from accessing any information that belongs to the publisher's web site, thus any information about the user must be given to the ad provider explicitly (remember that for the JavaScript code to be able to make an ad request, it must have originated from the ad provider's domain through either an `<iframe>` or `<script>` tag). For example, if a browser-based application requests the user's GPS location and the user grants these permissions, ad code on the web page cannot access the GPS coordinates without the application passing the data to the ad code. However on Android, the ad libraries have the ability to gather specific user information on their own. If an Android application has the ACCESS_FINE_LOCATION permission, which gives the application the ability to access the current user's GPS coordinates, the ad library gets this permission automatically. As we will discuss later, many of the ad libraries we observed will check if this permission is granted and automatically add the user's GPS coordinates to the ad request when available. In many cases, the developer cannot turn this behavior off and may not be informed it is occurring in the first place. We will go over some of the implications of in-app ad code having the same privileges as application code in Section V.

B. Detailed User Data

Applications installed on the user's device may have more privileged access to the user's data, compared to applications running in a browser. Browsers assume external code is untrusted and sandboxes it from a great deal of the functionality of the underlying operating system. For example, JavaScript code is typically unable to communicate with applications beyond the browser, and does not know what other applications may be running on the user's machine. However, with the right permissions, Android applications are able to determine which applications are running and installed. Likewise, mobile applications can be given permission to write to the SD card, read contacts, or read SMS messages, among many others. Compounded with the lack of privilege separation between app and ad code, this allows in-app advertising code to run in a very privileged state compared with in-browser ad code. We observe the private data each ad library collects and sends in ad requests in Section VI.

C. Consistency of User Identifiers

Android applications are particularly lucrative to ad providers because they provide a more consistent way of tracking users compared with in-browser advertising. The UDID discussed in Section II can be sent with the ad request and allows the ad provider to track the user's behavior over a long period of time and between different applications which include the same ad library. There is no way to track users with this same level of consistency in a web browser; the common tracking mechanisms, IP addresses and cookies, change over time and may be reset by the user. However, resetting the UDID values on Android either requires a factory reset or root permission on the phone. We discuss implications in tracking users using their UDID in Section VII.

We have discussed a number of differences between in-browser advertising and in-app advertising that allow mobile ad providers to potentially have greater power in violating users' privacy. Our goal in this paper is to determine if it is common practice for ad libraries to abuse these privileges or if insecure design decisions were made in building the library that could expose users to greater privacy threats. In the rest of the paper we analyze the most popular Android ad providers for potential privacy vulnerabilities. Additionally, we propose guidelines that would prevent these vulnerabilities in the future.

IV. Methodology

Our objective was to examine the most popular ad providers to gain the best insight into current Android ad provider practices. However, there is no publicly available list that ranks the popularity of ad providers. We chose the set of ad providers reviewed in this paper by: (1) examining the prevalence of each ad provider in real network traffic and (2) extracting the libraries from the top 500 applications on the official Android Market. Therefore, we used data from a tier-1 network provider in the United States to determine the ad providers that are the most popular in terms of the overall amount of ad traffic. In practice, the amount of ad traffic is correlated with the number of users using applications supported by the ad provider, although factors like refresh rate and number of advertisements displayed per window may influence these results. Our final list combines these ad providers with the most prevalent ad providers in the top 500 applications on the official Android Market [19]. See Table I for a list of the ad providers we consider.

For each ad provider, we wish to analyze the behavior of their library: the permissions it checks for and the data it sends over the network. To achieve this, we signed up as a developer with each ad provider and downloaded a copy of the most recent ad library available (as of February 2012). For each library, we instrumented a sample application that made an ad request to the provider using the library and captured the data sent over the network. By observing the captures, we were able to determine the fields, such as age or gender, that may be present in the ad request. Additionally, we referenced the providers' documentation[4] to determine how the fields were populated, which were *required* for a successful ad request, and which were *optionally* set by the developer. To ensure these fields are present in live traffic, we manually verified that ad requests from each ad provider actually contained the relevant fields by observing them in the network stream of the tier-1 wireless carrier. Section VI presents the results of our

Ad Library (version)	INTERNET	ACCESS_NETWORK_STATE	READ_PHONE_STATE	ACCESS_LOCATION	CAMERA	CALL_PHONE	WRITE_EXTERNAL_STORAGE	READ_CALENDAR	WRITE_CALENDAR	READ_CONTACTS	WRITE_CONTACTS	SEND_SMS	RECEIVE_BOOT_COMPLETE	GET_ACCOUNTS	READ_LOGS	ACCESS_WIFI_STATE
adfonic (1.1.4)	R	R		R												
admob (4.3.1)	R	R														
airpush (2-2012)	R	R	R	O									R			
buzzcity (1.0.5)	R		R			R										
greystripe (1.6.1)	R	R	R													
inmobi (3.0.1)	R	O		O		O		X	X							
jumptap (2.3)	R	R	R	O												
millennialmedia (4.5.1)	R	R	R		O		R									
mobclix (3.2.0)	R	O	R	X	X			X	X	X	X			X		
mOcean (2.9.1)	R	R	R	O	O	O	O	O	O			O			O	
smaato (2.5.4)	R	R	R	O												
vdopia (2.0.1)	R	R														
youmi (3.05)	R	R	R	R			R									X

TABLE I: *Ad SDK Permission Usage* As specified in their online documentation, ad libraries may require a permission (R) or declare it optional, but use it if available (O). Worryingly, some ad libraries check for and use undocumented permissions (X).

network traffic observations.

V. Permissions

In addition to monitoring network traffic, we analyzed each ad library's code to determine the sensitive information it accesses.

A. Permission Classification

In its documentation, each ad library specifies the permissions it *requires* to operate. Often, the ad library will further specify a number of *optional* permissions that the ad library can take advantage of to deliver more targeted advertising. We find, in practice, there is a third category, *undocumented* permissions. If an application has these permissions then its ad library can take advantage of these permissions to covertly access sensitive data. To ensure these undocumented uses do not cause the application to crash, ad libraries can dynamically check if they have a permission or catch a thrown SecurityException.

Through review of the documentation of each ad library, we obtained a list of the required and optional permissions. We discovered the permissions each ad library may use in practice by running Stowaway [13] on a test application consisting of only the ad library. Stowaway detects which Android framework API methods the application accesses and, using an internal mapping between API methods and required permissions, shows the permissions the application may need to run. This set minus the required and optional permissions gives the undocumented permissions described above. We manually investigated each of the undocumented permissions to determine if they were indeed being used.

[4]For ad provider documentation reference [1, 2, 3, 8, 22, 26, 27, 37, 33, 36, 42, 45, 46]

B. Permission Misuse

We present the three sets of permissions in Table I. Some permissions that have low impact on a user's privacy, such as WAKE_LOCK and VIBRATE were excluded from the results. Also, we chose to collapse Android's two location permissions, COARSE and FINE, because COARSE location information may be specific enough to infringe on a user's privacy. As expected, most ad libraries require a similar core set of permissions (INTERNET, ACCESS_LOCATION, ACCESS_NETWORK_STATE and READ_PHONE_STATE). However, some ad libraries, such as Mobclix, have many more undocumented permissions. Many of these undocumented permissions seem unnecessary to display ads, such as SEND_SMS and READ_CALENDAR. We conjecture that some of these permissions may be used to create a more complete user profile by actively collecting personal information.

The Mobclix ad library is particularly noteworthy among the set we analyzed because it utilized seven undocumented permissions. These include four invasive permissions: READ_CALENDAR, WRITE_CALENDAR, READ_CONTACTS, and WRITE_CONTACTS. Through manual analysis, we have verified that Mobclix contains functionality to read from and write to a user's calendar and contacts databases, although it will prompt the user before doing so. The mOcean and Inmobi ad libraries contain functionality to start phone calls and add events to a user's calendar without user interaction. Additionally, mOcean can send SMS messages without user interaction. These largely undocumented "features" are quite alarming.

C. JavaScript Interface

More disturbingly, we discovered that seven of the ad libraries we analyzed add a JavaScript interface to a WebView object, and in doing so, four of them open an avenue to attack a user's device. Android provides a mechanism allowing JavaScript code running in a WebView object to invoke a special set of designated application code through an interface. This can be useful for cross-platform applications written in JavaScript to remain cross-platform compatibility while incorporating platform-specific actions, such as Toast messages [20]. Additionally, this interface can be used to dynamically invoke other methods during runtime, similar to Java reflection. However, the Android documentation specifically warns against running untrusted JavaScript inside the WebView object, as the untrusted code can also access the interface.

1) Vulnerable Ad Libraries: Seven of the thirteen ad libraries we analyzed implement a JavaScript interface. We have confirmed that four of these seven will run external code within the WebView with a JavaScript interface. Functionality exposed by the interfaces for the different ad libraries is as follows:

- *Mobclix*: exfiltrate and/or modify the user's calendar and contacts, exfiltrate user's audio and image files, and turn on/off the camera LED.
- *Greystripe*: get and/or set user's cookies.
- *mOcean*: send SMS and email messages, start phone calls, add calendar entries, get location, make arbitrary network requests.
- *Inmobi*: send SMS and email messages, start phone calls, and modify the users calendar.

If an adversary were able to masquerade as one of these ad servers and supply malicious code, she would be able to perform any of the above functions on her target's device. In practice, an attacker can impersonate an ad server since the ad request and reply are sent in clear text. Then, if any user on the network were using an application containing one of the above four ad libraries, that user would be vulnerable to loss of personal data or other malicious actions. As these four ad libraries are in the top thirteen based on our metrics (Section IV), there are likely many devices vulnerable to this type of attack.

2) Proof-of-Concept: To demonstrate the significance of this vulnerability, we set up a test environment and attempted to exploit the Mobclix and mOcean ad libraries. Our test environment used a Samsung Galaxy Nexus as the victim device. To simulate the user's vulnerable application, we used our sample Mobclix and mOcean applications, which continuously request ads from the ad servers. We then impersonated the ad servers and supplied our malicious JavaScript in a response to an ad request.

In both cases, we were able to perform a successful end-to-end attack on our victim device. From the Mobclix ad library, we were able to obtain a stream of image data from an image that the user selected. From the mOcean ad library, we were able to initiate a phone call to an arbitrary number with no user interaction. An attacker could monetize this exploit by initiating phone calls to 0900 numbers on victims' devices. We were also able to obtain the device's location and start the email application with a pre-populated address, subject, and message from the mOcean ad library.

VI. Private Data on the Network

In this section we report what private user information each ad library is capable of sending over the network. As mentioned in Section IV, we observed each library's behavior both in an emulated environment as well as in live traffic, allowing us to determine which fields were present in ad requests. To determine how the fields were populated, we referenced the provider's documentation and noted which fields were required and had to be specified by the developer when calling the library. We determined which fields the library would set automatically by observing the permissions that the ad library used. The results of our analysis are outlined in Table II. We consider privacy leaks over the network to be particularly suceptible to observers on the network who are able to observe many ad requests from the same user.

We discovered a number of different behaviors regarding how ad providers handle private user information. The Admob library, for example, will always send the app package name over the network, however it will not attempt to collect location information on its own; instead, the developer must query for the GPS coordinates separately and explicitly pass these values into the library when requesting an advertisement. Libraries such as Airpush, however, will automatically include GPS coordinates in ad requests when either of the GPS permissions are available. Likewise, information that can be gathered from permissions such as connection type (cellular versus WiFi) and device ID are often gathered and sent by various ad providers. This is especially problematic when the developer is not informed of the behavior of the library and is not given the ability to turn off these features.

Targeting parameters such as the user's age, gender, or income typically cannot be gathered automatically by the libraries and must be explicitly set by the developer in order for these fields to appear in requests. However, if the developer cannot collect this information, she may statically set these fields in the hopes of targeting the application's user demographic, and therefore increasing her revenue. Thus, when observing targeting fields on the network, the values may not accurately correspond to users. An adversary may be interested in obtaining a targeted user profile for a particular device. A wily adversary could account for static data by building a long-term profile on the user based on their device or Android ID and aggregate targeting information across many apps and ad providers. Targeting information that is consistent across numerous independent sources would likely be correct. We discuss how tracking the user over these sources would be accomplished in Section VII.

As previously mentioned, ad providers do not typically encrypt ad requests because of the extra overhead that is

Ad Library (version)	App Package Name	GPS Coordinates	Connection Type	Device Make and Model	Wireless Carrier	Age	Gender	Income Level	Keywords
adfonic (1.1.4)		P		P		D	D		D
admob (4.3.1)	A	D				D	D		D
airpush (Feb 2012)	A	P		A	A				
buzzcity (1.0.5)									
greystripe (1.6.1)									
inmobi (3.0.1)	A	P	P	A		D	D		D
jumptap (2.3)		P				D	D	D	
millennialmedia (4.5.1)						D	D	D	D
mobclix (3.2.0)									D
mOcean (2.9.1)	A	P			D	D	D	D	D
smaato (2.5.4)	A	P	P		P	D	D	D	D
vdopia (2.0.1)			A	A					
youmi (3.05)	A								

TABLE II: *Private Data in Ad Requests* Some fields ad libraries will always populate in ad requests (A) while others it will populate only when the application has the appropriate permissions (P), both automatically. Alternatively, some ad libraries choose to only populate fields when the developer explicitly passes the value to the library (D).

required on the ad server to support large numbers of SSL connections. However, we did observe one ad provider, which attempts to obfuscate ad requests from an observer on the network. While Youmi doesn't encrypt the ad requests, it does do considerable work to make the requests difficult to read (interestingly, Youmi is the only Chinese ad provider we observed). Unlike other ad libraries, Youmi sends back the majority of its payload in a single HTTP GET field which appears to be a completely random string. Since the content of the network requests is a focus of this paper, we manually analyzed the Youmi ad library to discover how this field was created.

A. Youmi Encoding

First, we decompiled the ad library using *baksmali* [14]. We then manually reviewed the *smali* code and found that much of the encoding happens in a single class. They use an unknown encoding scheme which we believe may be proprietary; it transforms its input using bitmasks, bit shifts, scaling and other operations using a second string as a "key" to the encoding. We believe that a corresponding decoding algorithm could be created by reversing the encoding algorithm. However, to actually decode the URLs, we would need to determine what key was used. This may be difficult as it could be based on a shared secret embedded by Youmi in the ad library when a developer downloads it.

Since Youmi is the only ad provider that attempts to conceal the data being sent off the device, we wanted to determine if it was doing this to protect the user's privacy or because it had something to hide. As we aren't able to decode the outgoing requests, our only choice was to analyze the library. Rather than spend substantial time trying to understand the obfuscated library (perhaps output from Proguard [30]), we decided to inject our own code into the library to record the strings that were being encoded. We did this by decompiling the application using *apktool* [6], adding our custom logging class to the code base, and then adding hooks at the beginning and end of the encoding method to print the parameters and return values, respectively. We then recompiled the applications using *apktool* and installed it on an emulator. Our instrumented application allowed us to determine that Youmi does not send anything out of the norm.

Interestingly, in the same class that Youmi does the encoding, it has a method to do symmetric encryption. There was a second class of network requests that we were unable to decode and we hypothesize that these requests were generated using encryption rather than the encoding described above since these requests also appear to contain a unique identifier which the Youmi servers would need in order to lookup the correct decryption key. We hope to determine what the encryption's purpose and if it is possible for an attacker to figure out the symmetric key in future work.

Youmi's encoding scheme is a step towards protecting users' privacy, however, it may not be secure enough to prevent a determined attacker. We discuss how their encryption scheme could be better implemented in Section VIII-A.

VII. Tracking Users

Web ad providers have long tracked users across sites and several web ad providers may even collaborate to track users. However, the consistency of Android UDIDs allows for even more effective long-term user tracking, as they either never change or can only be changed with root privilege or flashing the phone. Figure III lists the UDID each ad library transmits and the hashing mechanism it uses, if applicable. We are concerned with two primary threats to mobile user privacy: an unscrupulous ad provider tracking users across several

installed applications and a network sniffer tracking users across several ad providers.

An ad provider may correlate a user's ad traffic across each application they've installed containing their ad library. This is because every ad provider consistently transmits the same UDID field (hashed or unhashed UDID value) regardless of the application in which it is included. For example, even though Admob sends the `md5` of the ANDROID_ID, every application on a user's phone containing Admob will transmit the same value, allowing Admob to correlate targeting information provided by all of the user's applications that use Admob.

Additionally, a further concern involves a network sniffer that may track users across several ad libraries. The key lies in that some ad libraries transmit device identifying information in clear text. As shown in Table III, some ad libraries transmit a user's UDID in clear text while others first hash it. As mentioned above, ad requests using the same library can be trivially correlated, as multiple applications using the same ad library will process and send the UDID identically. If every ad library hashed UDIDs differently then it would be impossible for a network sniffer to determine that two ad requests from different ad libraries are from the same user, assuming they use cryptographic hash functions. However, once a user's UDID is seen in the clear, the network sniffer can determine when ad requests from other libraries originate from that user by hashing the clear text UDID in the same manner as the various ad libraries. For instance, if an eavesdropper can capture ad requests to the Jumptap and Mobclix ad servers (therefore capturing both the DEVICE_ID and ANDROID_ID in clear text), then she can correlate subsequent requests to any ad provider from that user. Determining the hashing method used by ad libraries is straightforward, as the ad library can be reverse engineered.

This correlation across ad providers is not possible for an attacker sniffing in-browser ad traffic because web ads use cookies and other means to track users that are usually randomly generated. As the source of the tracking identifier is random and not based on an attribute unique to a user, one cannot build a user profile by correlating information received from different web ad provider requests.

VIII. Potential Solutions

In this section we propose potential solutions to the central problems we encountered when reviewing Android ad libraries: the failure to protect sensitive user data in transmission, mishandling of UDIDs, and the general issue of Android ad and application code having the same privileges. In proposing these solutions, we consider a threat model in which sensitive user information is collected either by malicious ad providers (who wish to be able to better target users) or a network sniffer who is able to observe ad requests coming from an Android device. We do not consider application developers to be malicious, but they may be uninformed or apathetic toward user privacy concerns when they choose to include an ad library in their application. Finally we do not consider the advertisers, whose ads are shown to the user, in the threat model because they generally are not in a privileged position to infringe on users' privacy.

A. Failure to Protect the Contents of Ad Requests

When it comes to protecting sensitive information during transmission, developers can use encryption to communicate without the fear of eavesdropping. Since ad requests contain sensitive information about the user, one would expect these transmissions to use some form of encryption; however, all the ad libraries we analyzed in this paper sent the information in the clear except for Youmi (Section VI-A).

Given the ease with which developers can encrypt their communications, it may seem surprising that they don't. We hypothesize that this is due to scalability issues with SSL. SSL is designed for long-term, persistent connections, and ad requests are short and must be handled in a timely manner so the user experience is not hindered waiting for ads. When communication is encrypted with SSL there is additional setup overhead which translates to more concurrent connections being open on the server at a given time. Considering the sheer volume of ad requests, even a moderate increase in the number of concurrent connections can require the ad provider to increase their hardware to ensure that they can still handle peak loads. Since the primary goal of ad providers, as with all businesses, is profit and there are little to no ramifications to the ad provider for not protecting the user's privacy, they will probably never use encryption unless a scheme can be designed which addresses the scalability issues.

In order to encourage adoption, an encryption scheme suitable for ad requests should have:

1) Low overhead: no session should have to be initiated in order to reduce the number of concurrent connections.
2) Fast encryption/decryption: symmetric key encryption is preferable over asymmetric encryption.

In the current ad-serving environment, it is not critical for our encryption scheme to be computationally infeasible to brute-force for a single ad request. This is because a single ad request contains little personal information on its own – we have shown the real danger to privacy occurs when ad requests can be observed over a long period of time to build a long-term profile on a user. Thus, our goal is for the encryption scheme to be at least computationally infeasible to brute-force a large number of ad requests so that building a user profile is no longer profitable to the adversary. We leave the design of such a scheme for future work.

B. Mishandling of UDID

Currently, each ad library handles UDIDs differently — some use a cryptographic hash and others send it off in the clear. A user's privacy must be protected against a network sniffer attempting to build a user profile across several ad providers and from an unscrupulous ad provider attempting to track a user across several applications. Recall from Section VII that it only takes one ad library sending clear text UDIDs to allow a network sniffer to correlate users across ad

Ad Library (version)	UDID Generation Scheme
adfonic (1.1.4)	sha1(ANDROID_ID)
	ANDROID_ID
admob (4.3.1)	md5(ANDROID_ID)
airpush (Feb 2012)	md5(DEVICE_ID)
buzzcity (1.0.5)	DEVICE_ID
greystripe (1.6.1)	DEVICE_ID
	ANDROID_ID
inmobi (3.0.1)	md5(ANDROID_ID)
	ANDROID_ID
jumptap (2.3)	ANDROID_ID
millennialmedia (4.5.1)	sha1(ANDROID_ID)
	md5(ANDROID_ID)
	ANDROID_ID
mobclix (3.2.0)	DEVICE_ID
	ANDROID_ID
mOcean (2.9.1)	md5(DEVICE_ID)
	md5(ANDROID_ID)
smaato (2.5.4)	DEVICE_ID
	ANDROID_ID
vdopia (2.0.1)	ANDROID_ID
youmi (3.05)	encode(DEVICE_ID)

TABLE III: *UDID's Per Ad Provider* Here we show how each ad provider populates its UDID field and thus how it is possible to recognize an ad request is from the same user across multiple ad providers. Ad libraries which have multiple encoding schemes will attempt to use the first one and only send subsequent ones in case of failure (ie. not having permissions or the appropriate hash function).

providers. Thus ad libraries should transmit

$$hash(\text{ad provider} + \text{UDID})$$

so the original UDID is never leaked and even if multiple ad providers use the same UDID the transmitted fields will not be identical.

However, this hash scheme will not prevent a single ad provider from tracking a user across several applications, as the hashed value will be the same for each ad provider on the same phone. This attack can be prevented by adding to the hash something unique to each application. For example,

$$hash(\text{ad provider} + \text{package name} + \text{UDID})$$

will be different for every ad provider for a given application and different across applications for a single ad provider, preventing tracking from both a network sniffer and an unscrupulous ad provider.

C. Lack of Ad and Application Privilege Separation

Ad libraries used by Android applications have access to all of the sensitive information and functionality as their containing application. This is both a violation of the principle of least privilege as well as a real world issue, as we found several popular ad libraries take advantage of this access (Section V). On the web, the same origin policy restricts ad code from reading or manipulating the publisher's content. We believe there is a need for a *mobile application same origin policy*, as the current situation does not provide adequate measures to protect users from unscrupulous ad libraries. Several important requirements of an effective mobile application same origin policy include:

- Application and included third-party code should not execute with the same privileges.
- Third-party code should not be allowed to access application-specific data nor phone user data unless specifically allowed by the user.
- If a permission is not explicitly granted to ad code then any request that requires it is denied.

Two potential methods to implement a mobile application same origin policy are to (1) separate ad libraries into their own applications or (2) restrict sensitive functionality based on the caller. If each ad provider had their own standalone application that would provide applications with ads via IPC, the permission separation problem would be solved; ad applications would only have the set of permissions they were granted at install time, easily reviewed by users. However, this complicates developers being compensated as users may refuse to install ad serving applications and developers may need to add checks in their code to ensure the availability of an ad provider, increasing code complexity. This solution also does not address the overall problem of privilege separation in third-party code — libraries to communicate with web applications and other functionality are not restricted.

We believe a better solution to the privilege separation problem is to restrict access to sensitive functionality by the source of the caller. This would increase user privacy while allowing nearly all of the current in-app ad distribution system to remain unchanged; ad providers would distribute libraries that developers could use to display ads. Quire [9] could be used to outfit Android with verifiable call chains of IPCs, allowing a modified Android framework to respond differently to sensitive functionality requests based on if it was originating from application code or third-party code. One potential way to distinguish application from library code is by package name, as every application already specifies its package in its manifest file. An additional entry type could be created for

the Android manifest file that explicitly grants third-party code permissions. For example,

```
<uses-permission package="com.admob"
  android:name="INTERNET" />
```

This approach informs users of the information an ad library may collect and grants developers the ability to control the capabilities of third-party code.

IX. Related Work

Previous work on online advertising privacy has focused primarily on in-browser advertising. Researchers have considered the dichotomy that exists between the ad providers, who want to target users to serve them the best ads, and the users, who do not want private information divulged [11, 17]. Approaches to mitigating privacy leaks in browser-based ad requests have focused on keeping private data from needing to be accessed by ad providers themselves [23], as well as regulations that keep ad providers from collecting private information in the first place [18]. A great deal of work concerning ad fraud, which hurts the advertisers who are paying for the ads, has been conducted [15, 43], but this work is not directly related to user privacy.

Privacy-conscious advertising models have been considered as a possible solution to user privacy concerns in in-app advertising [12, 24, 44]. Location privacy is of particular concern to mobile users, and users are generally not willing to share their location with advertisers when given the choice [28]. Solutions that allow location-dependent services to function without disclosing user location have been proposed [32], and such schemes could be adopted for ad serving if they were to be implemented.

User privacy on Android has been explored extensively in the literature. Previous work has focused on educating users to privacy vulnerabilities [29], as well as detecting and mitigating unwanted privacy leaks on the Android platform, both statically [16] and in real-time [10, 9]. However, none of these approaches have yet to be widely adopted and privacy concerns are still an open topic in the mobile sphere. Android permission security and permission-hungry applications have been studied in terms of privacy vulnerabilities [4, 13], and casual analysis of permission requirements of Android libraries has been previously considered [25].

In concurrent work, Grace et al. [21] analyze 100 ad libraries for potential risks to security or privacy. Using their automated tool, AdRisk, they detect which permissions the ad library uses based on approaches developed in [13] with findings similar to Table I. However, AdRisk did not investigate whether these permissions were required, optional or undocumented. In addition, they look for other code patterns of interest such as dynamic code loading, permission probing and JavaScript linkages within the ad libraries. In contrast, we focus on investigating privacy concerns such as how UDIDs are handled and what user data is sent in ad requests. Several other recent works have acknowledged the dangers of the lack of privilege separation between Android application and and ad code and propose methods of separating them [38, 41].

Additional concurrent work by Luo et al. [31] investigates security concerns in exposed JavaScript interfaces in WebView, much like Section V-C. Whereas our work looks at exposed interfaces in popular advertising libraries, they look at WebView more generally and point out a number of different attacks that can be performed that leverage WebView's functionality.

X. Conclusion

We have considered a number of privacy vulnerabilities in the most popular Android advertising libraries. Almost all of the libraries have functionality that allows for sensitive user data to be sent to the ad provider, although we consider the cases where the library automatically extracts and sends information when permissions are available to pose the greatest privacy threats. Additionally, we observed a number of ad libraries that check for and leverage permissions beyond what is specified in their documentation. Although no single ad provider may provide a complete private user profile, we identified that the UDID field present in nearly all in-app ad requests allows someone observing the network to correlate user information between different ad providers. Because the UDID fields are populated by persistent values, this allows the observer to build a long-term user profile including GPS locations and targeting information. We note there are no equivalent persistent UDID fields when viewing ads through a browser, and thus these privacy vulnerabilities are unique to Android in-app advertising. Finally, we proposed potential solutions to several common ad library privacy vulnerabilities, including the failure to protect user data in ad requests, mishandling of UDIDs, and the lack of privilege separation between application and ad code on Android.

XI. Acknowledgements

The authors would like to thank the anonymous reviewers for their insightful feedback. This paper is partially based upon work supported by the National Science Foundation (NSF) under Grant No. 0644450 and 1018964. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] *Adfonic*. Feb. 2012. URL: adfonic.com.
[2] *Admob*. Feb. 2012. URL: admob.com.
[3] *Airpush*. Feb. 2012. URL: airpush.com.
[4] David Barrera, P. C. Van Oorschot, H. Gnes Kayack, and Anil Somayaji. "A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android". In: (2010).

[5] Tim Bray. *Identifying App Installations*. Feb. 2012. URL: http://android-developers.blogspot.com/2011/03/identifying-app-installations.html.

[6] Brut.alll. *Android-Apktool*. URL: http://code.google.com/p/android-apktool.

[7] Internet Advertising Bureau. *IAB Interactive Advertising Wiki*. Feb. 2012. URL: http://www.iab.net/wiki/index.php/Category:Glossary.

[8] *Buzzcity*. Feb. 2012. URL: buzzcity.com.

[9] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D.S. Wallach. "Quire: lightweight provenance for smart phone operating systems". In: *USENIX Security*. 2011.

[10] William Enck, Landon P. Cox, and Jaeyeon Jung. "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones". In: (2010).

[11] David S. Evans. "The Online Advertising Industry: Economics, Evolution, and Privacy". In: *Journal of Economic Perspectives* (2009).

[12] A. Fawaz, A. Hojaij, H. Kobeissi, and H. Artail. "An on-demand mobile advertising system that protects source privacy using interest aggregation". In: *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*. Oct. 2011, pp. 127 –134. DOI: 10.1109/WiMOB.2011.6085414.

[13] A.P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. "Android permissions demystified". In: *Proceedings of the 18th ACM conference on Computer and communications security*. ACM. 2011, pp. 627–638.

[14] Jesus Freke. *Smali/Baksmali*. URL: http://code.google.com/p/smali.

[15] Mona Gandhi, Markus Jakobsson, and Jacob Ratkiewicz. "Badvertisements: Stealthy click-fraud with unwitting accessories". In: *Online Fraud, Part I Journal of Digital Forensic Practice, Volume 1, Special Issue 2*. 2006.

[16] C. Gibler, J. Crussell, J. Erickson, and H. Chen. "AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale". In: *Trust and Trustworthy Computing* (June 2012).

[17] A. Goldfarb and C. Tucker. "Online display advertising: Targeting and obtrusiveness". In: *Marketing Science* (2010).

[18] A. Goldfarb and C. Tucker. "Privacy regulation and online advertising". In: (2010).

[19] Google. *Android Market*. URL: http://market.android.com.

[20] Google. *Building Web Apps in WebView*. URL: http://developer.android.com/guide/webapps/webview.html.

[21] M. Grace, W. Zhou, X. Jiang, and A.R. Sadeghi. "Unsafe Exposure Analysis of Mobile In-App Advertisements". In: *Conference on Security and Privacy in Wireless and Mobile Networks (WiSEC)*. 2012.

[22] *Greystripe*. Feb. 2012. URL: greystripe.com.

[23] S. Guha, B. Cheng, A. Reznichenko, H. Haddadi, and P. Francis. "Privad: Rearchitecting online advertising for privacy". In: *Proceedings of Hot Topics in Networking (HotNets)* (2009).

[24] Hamed Haddadi, Pan Hui, and Ian Brown. "MobiAd: private and scalable mobile advertising". In: *Proceedings of the fifth ACM international workshop on Mobility in the evolving internet architecture*. MobiArch '10. Chicago, Illinois, USA: ACM, 2010, pp. 33–38. ISBN: 978-1-4503-0143-5. DOI: http://doi.acm.org/10.1145/1859983.1859993. URL: http://doi.acm.org/10.1145/1859983.1859993.

[25] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications". In: *Proceedings of the 18th ACM conference on Computer and communications security*. ACM. 2011, pp. 639–652.

[26] *InMobi*. Feb. 2012. URL: inmobi.com.

[27] *Jumptap*. Feb. 2012. URL: jumptap.com.

[28] Patrick Gage Kelley, Michael Benisch, Lorrie Faith Cranor, and Norman Sadeh. "When are users comfortable sharing locations with advertisers?" In: *Proceedings of the 2011 annual conference on Human factors in computing systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pp. 2449–2452. ISBN: 978-1-4503-0228-9. DOI: http://doi.acm.org/10.1145/1978942.1979299. URL: http://doi.acm.org/10.1145/1978942.1979299.

[29] J. King, A. Lampinen, and A. Smolen. "Privacy: Is there an app for that". In: *Proceedings of the 7th Symposium On Usable Privacy and Security (SOUPS)*. 2011.

[30] Eric Lafortune. *Proguard*. URL: http://proguard.sourceforge.net.

[31] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin. "Attacks on WebView in the Android system". In: *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM. 2011, pp. 343–352.

[32] Masanori Mano and Yoshiharu Ishikawa. "Anonymizing user location and profile information for privacy-aware mobile services". In: *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*. LBSN '10. San Jose, California: ACM, 2010, pp. 68–75. ISBN: 978-1-4503-0434-4. DOI: http://doi.acm.org/10.1145/1867699.1867712. URL: http://doi.acm.org/10.1145/1867699.1867712.

[33] *Millennial Media*. Feb. 2012. URL: millennialmedia.com.

[34] Elinor Mills. *Firms embrace Do Not Track for targeted ads only*. Feb. 2012. URL: http://news.cnet.com/8301-1009_3-57384193-83/firms-embrace-do-not-track-for-targeted-ads-only/.

[35] Elinor Mills. *Obama unveils Consumer Privacy Bill of Rights*. Feb. 2012. URL: http://news.cnet.com/8301-1009_3-57383300-83/obama-unveils-consumer-privacy-bill-of-rights/.

[36] *Mobclix*. Feb. 2012. URL: mobclix.com.

[37] *Mojiva*. Feb. 2012. URL: mojiva.com.

[38] P. Pearce, A.P. Felt, G. Nunez, and D. Wagner. "AdDroid: Privilege Separation for Applications and Advertisers in Android". In: *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. ACM. 2012.

[39] Nicole Perlroth and Nick Bilton. *Apple, Google and Others in Agreement on App Privacy*. Feb. 2012. URL: http://bits.blogs.nytimes.com/2012/02/22/california-attorney-general-reaches-deal-on-app-privacy/.

[40] *Same Origin Policy*. Feb. 2012. URL: http://www.w3.org/Security/wiki/Same_Origin_Policy.

[41] S. Shekhar, M. Dietz, and D.S. Wallach. "AdSplit: Separating smartphone advertising from applications". In: *Arxiv preprint arXiv:1202.4030* (2012).

[42] *Smaato*. Feb. 2012. URL: smaato.com.

[43] B. Stone-Gross et al. "Understanding Fraudulent Activities in Online Ad Exchanges". In: (2011).

[44] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. "Adnostic: Privacy preserving targeted advertising". In: *17th Network and Distributed System Security Symposium*. 2010.

[45] *Vdopia*. Feb. 2012. URL: mobile.vdopia.com.

[46] *Youmi*. Feb. 2012. URL: youmi.net.