PmDroid: Permission Supervision for Android Advertising

Xing Gao[1,2], Dachuan Liu[1,2], Haining Wang[1], Kun Sun[2]
[1]University of Delaware
[2]College of William and Mary
{xgao,dachuan,hnw}@udel.edu, ksun@wm.edu

Abstract—**It is well-known that Android mobile advertising networks may abuse their host applications' permission to collect private information. Since the advertising library and host app are running in the same process, the current Android permission mechanism cannot prevent an ad network from collecting private data that is out of an ad network's permission range. In this paper, we propose PmDroid to protect the data that is not under the scope of the ad network's permission set. PmDroid can block the data from being sent to advertising servers at the occurrence of permission violation in ad networks. Moreover, we utilize PmDroid to assess how serious the permission violation problem is in the ad networks. We first implement 53 sample apps using a single ad network library. We grant all permissions of Android 4.3 to these apps and record the data sent to the Internet. Then, we further analyze 430 published market apps. In total, there are 76 ad networks identified in our experiments. We compare the permission of data received by these ad networks with their official documents. Our experimental results indicate that the permission violation is a real problem in existing ad network markets.**

I. Introduction

Mobile advertising has become a multi-billion dollar industry in the past few years [4]. It is now growing six times faster than desktop advertising with an average 50% increase every year [3]. The Android advertising market is blooming as most Android applications are free. To earn revenue, free app developers display advertisements in their apps. Normally, mobile advertising networks collect advertisements from the advertisers and publish Software Development Kit (SDK) libraries for mobile app developers to display the ads with APIs. To use the advertising service, developers simply incorporate those ad libraries into their apps. Once installed, both apps and advertisements are running in the same process, and users cannot kill or uninstall ads without affecting the original apps.

In Android, applications need to declare permissions to access the corresponding sensitive information or perform specific actions. Permissions are predefined in the app's *AndroidManifest.xml* file and need to be granted by the user during the app installation. In this paper, we call these *app permissions*. To utilize the resources and the functions of a mobile phone, ad libraries also need permissions. For example, the ad network needs the INTERNET permission to access networks. The permissions, which ad SDK libraries need in app development, are identified in the documents accompanying the library publishing. We call these permissions *ad permissions*. The app developers have to add the permissions requested by ad libraries into their AndroidManifest.xml, which means the XML file contains both ad permissions and app permissions. As a result, the ad libraries can also access the data and APIs granted by the app permissions.

The application usually contains different permissions than those requested by ad libraries. This could cause privacy problems since ad libraries can use app permissions to collect user private information. The information collected from mobile users can be mined to provide better advertisements and thus potentially increase ad income. For instance, an ad network could collect location information to provide location-based advertisements. Phone numbers and IMEI numbers may be collected to track users. Moreover, browser history or contact information can help provide customized advertisements based on a user's interest. Although ad networks can request as many permissions as they want to collect user information, users may not agree to install an app if it asks for a plethora of dangerous permissions. Since undue ad permissions could decrease app developers' willingness to use the SDK libraries, many ad networks are prone to requesting a minimum set of permissions.

However, ad networks can still misuse the app permissions. Ad libraries can simply probe the permissions of the app that are not mentioned in their documents [13]. If the app does not have such a permission, Android will throw a SecurityException. The ad networks' SDKs can just catch this exception without any other action. Once the ad libraries detect the permission, they can use the permission to collect sensitive information and send it to remote servers. It is reported that more than half of the current ad libraries try to probe permissions [13].

In this paper, we focus on the problem that ad networks attempt to probe their host apps' permissions and misuse them to leak private information. We call this *the permission violation problem*. The Android permission system will inform users of the risk of permissions that the apps request. Once users understand it and grant the permissions to the apps, the ad library can use the permissions declared in their documents to collect information corresponding to the permissions. However, the ad components should not be allowed to send sensitive information protected by other permissions into ad networks. In other words, our goal is to prevent sensitive information protected by app permissions from being collected by ad networks.

The basic idea of our work is to track the private data in mobile phones and block it from being sent to ad networks using app permissions. We utilize the TaintDroid [10], which is an extension of Android, to track the flow of sensitive data dynamically in the system. However, we want to make our system as general and comprehensive as possible. We design our system to work in an advertising scenario and protect any private information from being sent to the network. Also, we aim to shield as much data protected by Android permissions as possible. To achieve these two goals, we need to uncover the relationship between data and permissions of Android systems. Though there is no official document, we use PScout [6] to build a permission-API mapping in our work. Based on the analysis of the mapping and Android official documents, we reveal that more than 30 permissions relate to sensitive data. By attaching taint to those sensitive data, we built PmDroid to protect information from being stolen by ad networks.

We also use PmDroid to investigate how serious the permission violation problem is. Two sets of experiments are conducted to measure the data sent to advertising servers. In the first set of experiments, both in-house apps with an ad SDK developed by ourselves and sample apps provided by ad networks are tested. We give all the Android 4.3 permissions to those apps. In the second set of experiments, we test a number of real apps from Google Play. The experimental results show that the permission violation problem does exist among the commercial ad libraries. Although it has not yet been widely exploited, such a problem could become a serious risk to mobile users' privacy.

In this paper, we make the following major contributions:

- We propose PmDroid to protect Android apps from the permission violation problem. PmDroid can block the data delivery to ad servers if apps send sensitive information beyond ad permissions.
- We perform extensive experiments using PmDroid to show that the permission violation is a real problem in ad networks.
- We study the impact of such a problem and provide several insightful observations.

The remainder of the paper is organized as follows. We describe the background information and related work in Section II. In Section III, we detail the architecture and implementation of PmDroid. We present our experiments in Section IV and study the impact in Section V. We discuss the permission violation problems observed in the experiments in Section VI. Finally, we conclude the paper in Section VII.

II. Background and Related Work

A. Android and Android Permissions

Android is a privilege-isolated mobile phone operating system based on the Linux kernel. Applications in Android run with distinct system identity in separate processes. Thus, they are isolated from one another and the Android system framework. For the sake of security, each application runs in a sandbox and can only access limited resources by default. Android makes use of a permission mechanism to protect sensitive data and important functions. If an application wants to acquire sensitive information or APIs such as the location data or camera function, it must declare the corresponding permissions in the manifest file. All dangerous permissions claimed in the manifest file will be displayed on the screen during the app's installation. Only after the permissions are granted by a user, can the app then call protected APIs or access the sensitive data.

Android defines more than 100 permissions and classifies them with "normal," "dangerous," "signature" and "signatureOrSystem." If an application is signed with the same certificate as some others, it can acquire the permissions declared by those apps in category "signature." The permissions belonging to "signatureOrSystem" are similar to those of "signature." The difference is that they could also be used by the applications in the Android system image. Permissions from these two levels are not available to third-party applications; therefore, we focus on the regular permissions tagged as "normal" and "dangerous." Note that "normal" is the default value and contains low-risk permissions. Permissions in "dangerous" are more important and allow the applications to access private information.

The permissions needed by applications in Android are usually different from those requested by ad libraries. Once incorporating an ad library, the app will share the same permissions with the advertising SDK because they run in the same process. If the app requires no particular permission, the app may suffer from permission bloat problems [23], since the app must request the advertising SDK's permissions. On the other hand, if the application needs more permissions than the ad network, it can cause over-privileged advertisements [27]. Advertising SDKs could use app permissions, which are not required by the ad network, to collect sensitive data. In this paper, we focus on the over-privileged advertising SDKs. Once advertising SDKs collect sensitive data by utilizing the host's app permissions, we regard that they compromise user privacy.

B. Related Work

Android Security. Research has long shown that security remains a vital issue for Android platforms [7], [12], [29]. Both static [5], [8], [20] and dynamic techniques [9], [10], [26], [28] are used to detect malware or enhance security mechanisms. Since malware could leverage obfuscation techniques, such as Java reflection or bytecode encryption [22], to avoid being detected by static analysis, dynamic monitoring becomes essential and effective in the battle against malware.

TaintDroid [10] is built to track information flow in Android systems. TaintDroid provides a 32-bit taint tag for each variable by doubling the size of the stack frame. It uses dynamic taint analysis to efficiently monitor the data on smartphones with four granularities of taint propagation. Sensitive information is marked with a taint tag at the taint source and is then tracked inside the device. It will alert a user when sensitive information is sent to networks. While

TaintDroid does not track control flow, many other works consider implicit flows. SpanDex [9], based on TaintDroid, tracks both explicit and implicit flows. It relies on ScreenPass [19] to set taint tags for passwords and studies the password revealing problem.

Android Permissions & Mobile Advertising. The permission sharing problem between apps and advertising has been studied for years. Grace et al. [13] used static analysis methods to analyze ad libraries. They found that many ad libraries try to probe permissions. If the app does not include such a permission, it will throw a SecurityException, and the ad SDK simply catches the exception. Otherwise, the advertising SDK can utilize this permission. Stevens et al. [24] found a similar problem. They investigated 13 advertising SDKs by analyzing network traffic of a tier-1 wireless carrier and found that 3 of them would use permissions not recorded in the documents.

To solve such a problem, several research works try to isolate mobile advertising from applications. AdSplit [23] separates the original application and advertising into different processes so that each process has its own set of permissions. It uses transparency techniques to show advertisements under the main activity. AFrame [27] achieves both process and display separation. However, these works only provide limited advertisement styles and need modification of the app's source code. It may work well on displaying simple banner ads; however, it might not be enough for supporting other styles, such as ads in full screen. Users can simply kill the ad process, which is not the desired scenario for app developers or ad networks. AdDroid [21] proposes to include an ad library in the Android SDK, so app developers can use the API provided by Android to display advertisements. However, in this solution, the ad library can only be updated when Android is updated. Moreover, all of these works require code modification of apps, which would be annoying to app developers and inefficient for inexperienced developers.

While TaintDroid tracks 5 types of private data in mobile devices and alerts users when sensitive data is sent into networks, it does not take the permissions of ad networks into consideration. AppFence [16] uses TaintDroid to track 12 types of sensitive data from analysis of 11 dangerous permissions. It provides two mechanisms to protect this sensitive information. Users can choose to either replace sensitive data with shadowing data or block the data transmission that contains private data. Although it helps users protect their private information, this work does not focus on the permission violation problem. Users are still not aware of what information is collected by ad networks or the relation between data and permission.

Both Stowaway [11] and PScout [6] analyze the Android permission systems and provide a permission-API mapping. We use PScout in our work because PScout can be easily reused on the version of Android 4.3. Wei et al. [25] conducted a long-term study on the permission evolution of Android systems, and they found that both the number of "dangerous" permissions and the percentage of over-privileged apps increase over time.

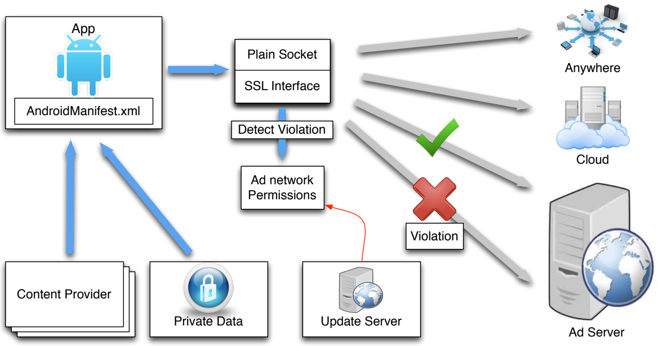


Fig. 1: Architecture of PmDroid.

There are other works considering the privacy issue of mobile advertising. Leontiadis et al. [17] proposed a market-aware privacy protection framework that decouples the application from the advertising component so it can send private information separately to developers and ad networks. Another framework for which users can decide how much private information to share is proposed in [15]. Han et al. [14] compared the privileges used by the same version of advertising SDKs in Android and iOS, while Liu et al. [18] focused on characterizing ad frauds in mobile apps.

III. PmDroid

In this section, we introduce the architecture and implementation of PmDroid. We extend TaintDroid and AppFence to track sensitive data. However, it is impossible to distribute tags to every specific important type of data due to the limited taint bit provided by TaintDroid. Instead of calling attention to the data itself, we focus on the permissions required for accessing the data. For instance, both the phone number and the IMEI number require the same permission, READ_PHONE_STATE, thus we mark them with the same taint tag.

Figure 1 briefly illustrates the architecture of PmDroid. PmDroid taints sensitive private data based on their required permissions when apps try to access that data via APIs or Content Providers. The sensitive data is tracked in the mobile phone. When apps try to leak important data into networks through either plain socket or SSL interface, the taint tag of the data will be compared with the permission list maintained in a mobile phone. The list contains the information of ad networks and can be updated by a web server. If permission violation happens, PmDroid can block the data sent to ad servers.

A. Permission Taint Placement

To monitor the data flow in a smart phone, we first need to find the taint source to assign the taint tag. Android SDK provides app developers with many APIs to perform specific functions and get sensitive data. Some important data could be obtained via those APIs only if the apps contain specific permissions. Thus, we can taint the data during the API call. Also, a lot of sensitive data is stored in Content Provider. Third-party apps can use APIs to retrieve data from Content

Provider with a specific content URI[1] like a database. Some URIs are also protected by permissions. We can set the taint of those data when they are read from Content Provider based on their URIs.

Since we focus on the permissions, a permission-API mapping of Android is necessary for us to track all the sensitive private data starting from the API call. Unfortunately, Android does not provide a complete mapping between APIs and permissions. In the Android official document, it mentions part of their published APIs and the permissions they require. However, other APIs that also need permissions are not mentioned due to the documents being incomplete. Moreover, there are a lot of undocumented APIs existing in Android. Although Android does not recommend developers to use undocumented APIs because they might be deprecated in the future and cause apps crash, some still use them.

To generate a permission-API mapping, we run PScout on Android 4.3. Although PScout cannot ensure a totally comprehensive and correct mapping (which is also difficult to justify), it does provide a useful way to understand the connections between APIs and permissions. The PScout scan result shows that more than 30,000 APIs, including more than 1,000 published APIs, require permissions. It also demonstrates that more than 40 content URIs need permissions. Combining the result with Android official documents, we can find out which data is protected by permissions. Then, we instrument the API or taint the data source based on their permissions. Specially, we focus on the permissions that could be used to read data. We use methods below to taint data.

- For the permissions related to location, microphone, and camera information, which have been tainted by TaintDroid and AppFence, we keep them unchanged but modify the taint tag. Also, for the data stored in Content Provider, we add taint tags to the data files so that data will be tainted once it is read from Content Provider.
- For the permissions that have published APIs, we focus on the non-void published APIs. We either instrument those published APIs or hook the unpublished APIs. We regard that the system data read from APIs is clean, which means it only contains the permission's taint tag.
- For permissions that only have void published APIs or permissions that do not have published APIs, we hook several non-void undocumented APIs with permission tags.

We find that some content URIs or APIs are mentioned in official documents but do not appear in the result of PScout. Also, some APIs do not need the permissions found by PScout. For data that has been tainted by content URIs, we dismiss other APIs. These differences are taken into account when we implement the prototype of PmDroid. We tested the functions of all instrumented APIs to ensure that data contains only the configured taint bit.

B. *Taint Tag*

As we mentioned before, among more than 100 permissions in Android, only those permissions belonging to "normal" and "dangerous" could be used by third-party applications. Also, not all permissions, such as VIBRATE, are related to reading private data, which means apps cannot use the permissions to obtain some private information. Moreover, there are many permission pairs in Android, such as WRITE_CALENDAR and READ_CALENDAR. Normally, the app can read data if it possesses the write permission. Because of this, we treat them as one in PmDroid.

To set the taint tag, we first need to make sure that all permissions acquired by advertising SDKs are included. We collect the permission set of 112 ad networks from the lists of AppBrain [1] and other websites. In total, we obtain 35 permissions (including both WRITE and READ) that are mentioned at least once in their official documents. Figure 2 shows the number of permissions acquired by all 112 ad networks. As shown, there are required permissions and optional permissions in ad networks' documents. The required permissions indicate that app developers must add these permissions into the app's AndroidManifest.xml file in order to access the advertisements of ad networks. On the other hand, the ad component can still work in the app without the optional permissions. The optional permissions are mainly used to provide better services, such as location-based advertisements.

After combining the WRITE and READ permissions, there are 28 permissions. We observe that all ad networks request the INTERNET permission to access the Internet. Also, almost all ad networks require or strongly recommend ACCESS_NETWORK_STATE permission, which checks the network connection status. Since these two permissions are requested by most ad networks, we left them alone. Other frequently requested permissions include those about location, accounts, Wi-Fi states and phone states. Some ad networks will also ask for SMS, audio or radio, calendar and browser history information.

Since our main goal is to prohibit ad libraries from leaking private information through permission probes, we also need to take other permissions into consideration. After combining the WRITE and READ permissions, we have 33 permissions in total that are related to reading important data. Table I shows all 33 permissions. Among 28 permissions required by ad networks, 17 of them need to be tainted, through which we believe private data could be retrieved. The TaintDroid provides a 32 bits tag, which is long enough if we set all permissions not being used by ad networks as OTHER. In our experiments, in order to provide better understanding of what information is sent by ad networks, we combine some permissions related to settings and make use of all 32 bits to taint the 32 types of permissions.

Among the 32 bits of taint tag in PmDroid, there are some permissions possessing more than one bit. For example, in Android, the ACCESS_COARSE_LOCATION only

[1] "A content URI is a URI that identifies data in a provider" [2].

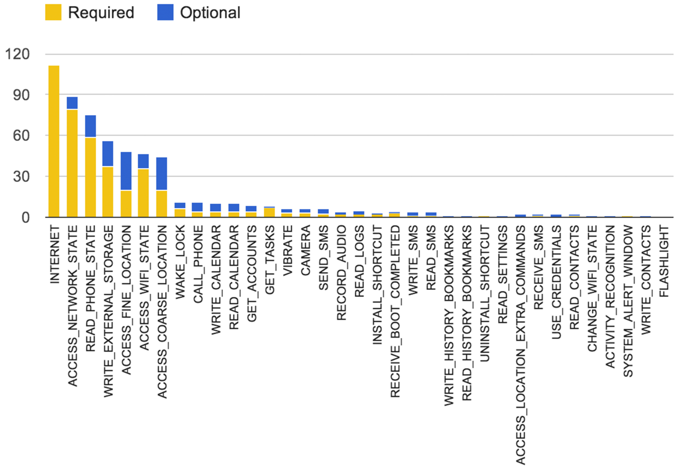


Fig. 2: Permissions used by ad network.

Ad network	Non ad network
ACCESS_COARSE_LOCATION	USE_SIP
ACCESS_FINE_LOCATION	BATTERY_STATS
WRITE_EXTERNAL_STORAGE	READ
READ_PHONE_STATE	READ_SYNC_STATS
RECORD_AUDIO	READ_SYNC_SETTINGS
GET_TASKS	GLOBAL_SEARCH
GET_ACCOUNTS	ACCESS_ALL_DOWNLOADS
READ_HISTORY_BOOKMARKS	READ_PROFILE
INSTALL_SHORTCUT	GET_PACKAGE_SIZE
READ_SETTINGS	READ_SOCIAL_STREAM
CAMERA	GALLERY_PROVIDER
READ_CALENDAR	ADD_VOICEMAIL
READ_SMS	READ_USER_DICTIONARY
ACCESS_WIFI_STATE	READ_CELL_BROADCASTS
USE_CREDENTIALS	READ_ATTACHMENT
READ_CONTACTS	ACCESS_PROVIDER
ACCESS_WIFI_STATE	

TABLE I: Permissions tracked in PmDroid.

contains the permission to receive the location update from the NETWORK_PROVIDER, which could be less accurate than the location update from the GPS_PROVIDER. However, if the user grants the ACCESS_FINE_LOCATION permission to an app, the app can access location information from both NETWORK_PROVIDER and GPS_PROVIDER. We need two bits to represent the GPS_PROVIDER and NETWORK_PROVIDER separately, so that the two permissions can be tracked.

C. *Violation Control*

To stop advertising SDKs from sending private information to networks when permission violation happens, we leverage the AppFence to block data transmission. Both plain socket and SSL interface are modified so that the taint tag of data will be compared before sending. If the app tries to send data, which does not belong to advertising SDKs, to ad networks, it will stop sending data. We regard the destination as an ad network if the domain name contains the name of an ad network or some specific keywords. Those destinations will be further explored in the following section.

To detect the permission violation problem based on the taint tag of data, PmDroid needs to be aware of the permissions required by each ad network. Previous work separates the permissions when parsing the manifest file. AFrame modifies the XML parser to separate permissions of apps and ad networks in the AndroidManifest.xml file. It might be intricate when the app uses more than one ad network. Also, inexperienced app developers may fail to set the correct permissions of ad networks, thus making the system useless.

Aiming to have less modification on current apps, we use a different method to handle such a situation. We maintain an ad network list in the mobile phone. The list contains the name of the destination and the permission set required by ad networks. We build a server so that the list can be updated at the server. The server could be maintained by either Android itself as a similar but much simpler approach as [21] or Service Providers or any credible proxy who is willing to provide a safer environment for mobile users. Only limited sets of permissions, say INTERNET, would be given to unknown ad networks, forcing them to share their permission sets. This mechanism does not require any modification on current apps, which is ideal for developers. Furthermore, it enables PmDroid to conveniently expand the number of ad networks, the permission sets of ad networks, and the destination addresses. Note that PmDroid can also work for third-party companies other than ad networks. For example, Grantoo publishes its SDK to provide online multi player and social features for mobile games. PmDroid can simply include the company information on the list through the remote server, thus protecting a user's private information from theft.

While it is a challenging task to cover all ad networks on the Internet in this list, as long as we are able to include the major ad networks in the list at the beginning and accumulatively update the list in a periodical fashion, PmDroid will reach the goal of protecting users' private data from malicious ad networks. Finally, our mechanism is also compatible with previous methods and can work jointly with them to defend Android users against the permission violation problem.

D. *Limitation*

Since PmDroid is a system based on TaintDroid and AppFence, most known limitations of TaintDroid and AppFence are still applicable to PmDroid. For instance, TaintDroid only tracks data flows. PmDroid also cannot work when malicious apps leak personal sensitive information through control flows. Also, TaintDroid could experience taint exploration, which would cause false positives because some clean data is treated as tainted. While these have been heavily studied, several techniques such as those used in SpanDex could be leveraged to address the limitation. AppFence stops data delivery to ad servers based on the name of the destinations. It would not work if the advertising SDKs sent private information to cloud servers. More advanced methods are needed, and we will explore this direction in our future works.

Another limitation is the absence of an official mapping between permission and data. Although the results of PScout cover most permissions and APIs, it cannot guarantee absolute correctness. Also, we do not hook all but instead choose those important and non-void APIs based on the results of PScout. Some paths to obtain data could be ignored in PmDroid.

IV. Experiments

Since TaintDroid has proven its low overhead (14% CPU overhead and 4.4% memory overhead [10]) and compatibility with current apps in the market, we do not repeat similar experiments. Instead, we conduct two sets of experiments using PmDroid to study the permission violation problems in current ad network markets. In the first experiment, we use example apps developed by ourselves and official sample apps provided by ad networks. In the second experiment, we download real apps from Google Play to perform the measurement.

A. Sample Apps

Free Android apps normally contain more than one ad network, which may require different permissions and send data to different destinations [17], [23]. To reduce the complexity of involving multiple ad networks, we first conduct experiments focusing on a single ad network. Most ad networks provide SDK downloads without the requirement of registration; however, they will not provide ad service until a user registers them with a published app. Therefore, we build a simple app and upload it to Google Play. Then, for all 112 ad networks in our list, we either try to register online or send them emails to request their Android SDK. Not all ad networks accepted our registration. There are also some foreign ad networks that require personal identification or a bank account in their countries for registration. We finally obtained 53 advertising SDKs from different ad networks.

We then use our sample app to apply for an app ID or something similar. The app ID is used to build an example app for each ad network by following their documents. For those ad networks' SDKs that contain example projects, we simply replace the app ID and use the projects provided by ad networks. We grant all permissions in Android 4.3 to each project so that they can collect all the data they want.

Again, some ad networks refuse to provide a new ID to our application. Some ad networks only provide fake ads or test ads. There are several ad networks that do not provide ads even after we successfully obtained a new ID. In total, 31 out of 53 ad networks provide us with real ads.

For each ad network's sample app, we first run it 3 to 20 times to guarantee at least 5 advertisements in the experiment. For those ad networks that do not provide advertisements, we run them 5 times. Then we run all apps for 3 minutes to make sure the ad networks have enough time to collect information. We record not only the taint tag they send into network, but also the destination's URL name and IP address.

B. Market Apps

In the first experiment, we only implement basic functions of ad networks. Most ad networks provide multiple ad styles or advanced solutions. These methods probably collect more private personal information to provide better advertisements. The real apps published in markets might implement those methods for the purpose of earning more money. Also, there are many ad networks that only deliver fake ads or no ads if they do not collect any sensitive information. Therefore, to make a comprehensive study of the permission violation problem, we use the real apps which have been published in the Android market in our second experiment.

We downloaded 430 apps from Google Play based on the list from AppBrain [1]. We utilized the URL name and IP address obtained from the first experiment to distinguish the ad network. We conducted similar experiments on those market apps as sample apps.

V. Results

A. Destination

In the first experiment, we treat all outgoing traffic as the data sent to the ad network since all apps have no other functions but acquiring advertisements. Only a few destinations of ad networks contain the name of their companies, and many ad networks send data to cloud servers like Amazon CloudFront. Some ad networks use IP addresses as their URL names.

In experiment two, we found that a large number of apps (201 out of 430) send data to cloud servers. For example, Amazon receives data from 152 apps. Since the addresses of cloud servers provide us with little knowledge about specific ad networks, except for a few ad networks we collected in the first experiment, we ignore all traffic sent to cloud servers when trying to distinguish the ad networks. We also found that many applications send data to destinations that are not related to ad networks. In our experiment, we considered the following destinations as *anywhere*: (1) the destination only has an IP address that is not in our list; (2) the company that develops or publishes the app; and (3) some other third-party companies that we believe do not belong to the ad network. In the end, 60 ad networks are found among 430 apps in the second experiment, 55 of which are covered in our list. We failed to obtain the permission information for 5 ad networks. From both experiments, we obtained 76 known ad networks in total.

Table II shows the data collected by the apps and the destinations to which the information is forwarded. NULL represents untainted data. We can see that ad networks possess a large part of the outgoing data of Android apps. Although the major proportion of the data is untainted, ad networks still receive a lot of personal information from mobile phone users, especially location and information about the device. Cloud servers are the destinations of many apps because many app companies of ad networks utilize their services. However, much less private data protected by permissions is collected by the cloud servers. We also found a number of destinations

Permission	Ad server	Cloud	Anywhere
NULL	188	166	111
READ_PHONE_STATE	89	29	68
ACCESS_WIFI_STATE	62	9	41
ACCESS_COARSE_LOCATION	15	3	19
ACCESS_FINE_LOCATION	15	2	13
WRITE_EXTERNAL_STORAGE	7	1	2
GET_ACCOUNTS	4	1	15
READ_SMS	1	1	5
READ_CONTACTS	0	1	4
READ_PROFILE	0	0	2
GET_TASKS	0	0	2
READ_CALL_LOG	0	0	1

TABLE II: Destination of Permissions.

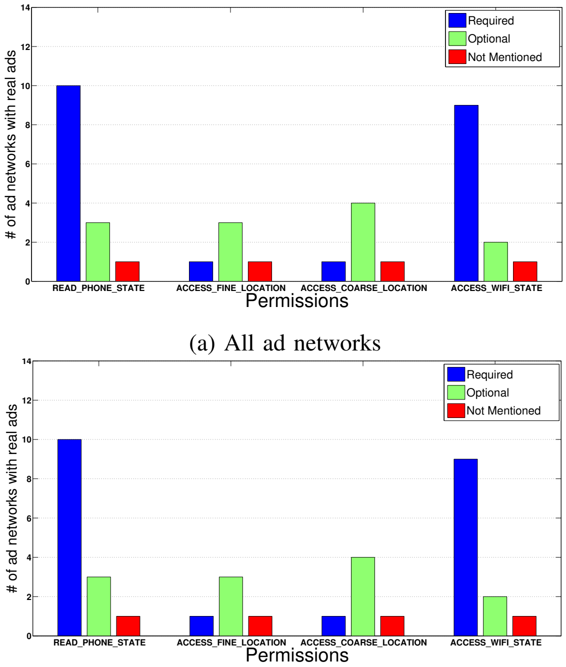
that were unclear as to whether they were ad networks. Some destinations marked as "anywhere" may contain certain ad networks. Moreover, we found that the data sent to anywhere covers more permissions than ad networks in our experiment. Most of these types of data are sent to the server of the app's company.

B. Information Collected

In the first experiment, we focus on each single ad network. Only 4 types of data belonging to 4 permissions are collected by 53 ad networks. We illustrate the results in Figure 3. Figure 3a shows the permission data collected by all ad networks identified in experiment one. We also present the result of 31 ad networks that provide us with real ads in Figure 3b. While most ad networks send untainted data to their servers, we see that vast amounts of data protected by the permissions READ_PHONE_STATE and ACCESS_WIFI_STATE are collected. Most ad networks declare the READ_PHONE_STATE permission in their documents since this permission is used to get information about the mobile device, such as the phone number and IMEI number. The ACCESS_WIFI_STATE permission allows applications to access information about Wi-Fi networks. The most important and interesting data protected by this permission is the MAC address, which, similar to the phone state, is another piece of identity information of the mobile phone.

Location information can be used for providing location-based advertisement. Among 53 ad networks, 14 of them collect location information – 11 of them collect fine location while the others only collect coarse location. No other data is collected, even though some ad networks require permissions for the account, calendar, camera and so on. There are two major possible reasons. First, we only implement the basic function of the advertising SDK. Second, some ad networks only provide us with fake or test ads or even no ads.

Figure 4 shows the data collected by 60 ad networks we found in the 430 market apps. There are 5 ad networks marked as "unknown" since we cannot identify them. We find that the information about phone state, Wi-Fi state and location are also collected most frequently. Besides the four permissions we found in experiment one, several ad networks collect account information. WRITE_EXTERNAL_STORAGE permission is usually used to store advertisements. Some data is tainted with this permission. There is one ad network receiving tainted data with READ_SMS tag, and the permission is not mentioned in its document.



(a) All ad networks

(b) Ad networks with real ads

Fig. 3: Permissions collected by ad networks in Exp 1.

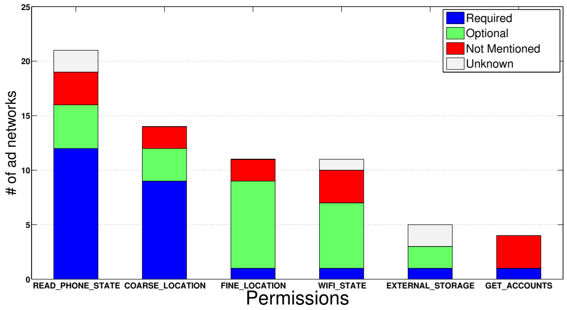


Fig. 4: Permissions collected by ad networks in Exp 2.

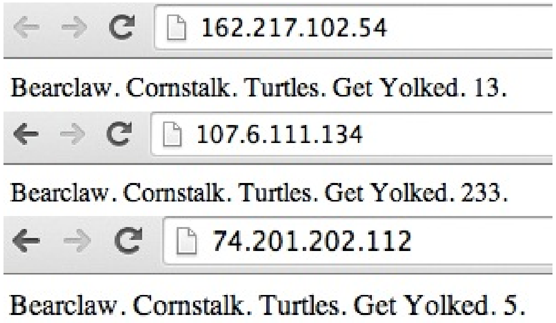
C. Permission Violation

We find that the permission violation problem does exist in current ad networks. Table III exhibits all ad networks that have this problem in our experiment. In experiment one, 4 ad networks use the permissions out of their documents to collect private information. One is Mobclix, which has been reported by [24]. Mobclix collects location information if the apps contains any location permissions. AppFlood collects both location, phone state and Wi-Fi state without mentioning any permissions. We find that both Mopub and Domob acquire coarse location permission in their documents. However, they will also collect fine location.

In our experiment, Mobclix only sends data into Amazon's cloud server. After running the demo several times, we figured

Ad networks	INTERNET	ACCESS_NETWORK_STATE	ACCESS_COARSE_LOCATION	ACCESS_FINE_LOCATION	ACCESS_WIFI_STATE	WRITE_EXTERNAL_STORAGE	READ_PHONE_STATE	GET_ACCOUNTS	VIBRATE	WRITE_CALENDAR	READ_CALENDAR	CALL_PHONE	USE_CREDENTIALS	WRITE_CONTACTS	READ_CONTACTS	SEND_SMS	RECEIVE_SMS	WRIETE_SMS	READ_SMS	RECORD_AUDIO
AppFlood	X			-	-	X	-													
Mobclix	X	X	-	*			X													
Mopub	X	X	X	-		O	+													
Domob	X	X	X	*	X	X	X		X											
Admob	X	O						+												
Flurry	X	X		+	+															
StartApp	X				X	+	X													
Vserv	X		O	+	O	O	O			O	O	O	O	O	O	O				
Papaya	X	X	O	O	+		X	+									O	O	O	
Millennial	X	X	O	O		X													+	O
Admarvel	X	X	O	O	+	X	+			O	O									

TABLE III: Permission violation problems found in our experiments. (X) represents the required permissions of ad networks. (O) stands for the optional permissions. (*) shows the permissions collected in experiment one but are not mentioned in their documents. (+) denotes the unmentioned permissions in experiment two. We use (-) to show the unmentioned permissions appearing in both experiments.


162.217.102.54

Bearclaw. Cornstalk. Turtles. Get Yolked. 13.

107.6.111.134

Bearclaw. Cornstalk. Turtles. Get Yolked. 233.

74.201.202.112

Bearclaw. Cornstalk. Turtles. Get Yolked. 5.

Fig. 5: Destinations of Mopub.

out that the destinations are always the same two addresses. In the next step, we considered that one app contains the SDK provided by Mobclix if it sends data to the two addresses identified in experiment one.

Mopub sends the collected private data to several destinations with different IP addresses. The URL name is just the IP address. When we enter the IP address, all the webpages are similar, as illustrated by Figure 5. If the app sends data to the exact same addresses, we believe they send the data to the Mopub ad network. A similar situation happens on several other ad networks, including Flurry and Millennial.

We also find the permission violation problem existing in Mobclix, Mopub and AppFlood in the second experiment. Besides, among the 55 ad networks covered by 430 apps of experiment two, the problem is found in another 7 ad networks among 7 apps. Table IV lists the applications, ad networks, and permissions that protect collected data. Flurry, which only acquires INTERNET and ACCESS_INTERNET_STATE permissions, receives fine location from two apps. The Wi-Fi state is also sent to Flurry in one app. Millennial receives data tainted with an SMS tag in one app. The location protected by ACCESS_FINE_LOCATION is collected by Vserv. We also find that both Admob and Papaya collect account information in different apps.

Through our experiments, we also notice that the permission violation problem in ad networks is still in its early stages and has not yet been widely exploited. More than 100 apps send untainted data to Admob in our experiment, but we only found the problem in one app. Similarly, two apps indicate that Flurry suffers the permission violation problem while another 60 apps do not. In fact, all 9 ad networks with this problem appear more than once in our second experiment.

We notice that false positives might happen in TaintDroid since all data in a parcel share the same taint tag. It is possible that the tainted data received by the ad networks is not collected by the ad networks themselves. As illustrated by Table IV, in most cases, whenever the permission violation happens, this type of data is either collected by another ad network that requires or optionally requests the permission or is sent to another place that is not an ad server. Take the app "1Weather: Widget Forecast Radar" as an example; we find that Flurry receives location information. However, fine location data is also collected by Admarvel with optional permissions. Moreover, the app will send data protected by ACCESS_WIFI_STATE to both Admarvel and Flurry. While these two ad networks do not acquire the permission, the app will send Wi-Fi state information to other places. In this app, all the data that violates the permission of the ad network is sent legally to other ad networks or other destinations. Since PmDroid is extended on TaintDroid, it may not prove that advertising SDKs contain codes for stealing user's private

App	Destination	Permission	
Treasure Looter	Admob	GET_ACCOUNTS	(-)
	Other	NULL	
Lucktastic	Flurry	ACCESS_FINE_LOCATION	(-)
	Kiip	ACCESS_FINE_LOCATION	(O)
1Weather: Widget Forecast Radar	Flurry	ACCESS_FINE_LOCATION	(-)
		ACCESS_WIFI_STATE	(-)
	Admarvel	ACCESS_FINE_LOCATION	(O)
		ACCESS_WIFI_STATE	(-)
	Other	ACCESS_WIFI_STATE	
chomp SMS	Millennial	READ_SMS	(-)
	Other	READ_SMS	
The Hunter batman	Papaya	GET_ACCOUNTS	(-)
		ACCESS_WIFI_STATE	(-)
		GET_PHONE_STATE	(X)
	AppFlood	GET_PHONE_STATE	(-)
		ACCESS_WIFI_STATE	(-)
	Tapjoy	GET_PHONE_STATE	(X)
KiKORiKi Free	Vserv	GET_PHONE_STATE	(O)
		ACCESS_WIFI_STATE	(O)
		ACCESS_FINE_LOCATION	(-)
	Other	GET_PHONE_STATE	
		ACCESS_WIFI_STATE	
		ACCESS_FINE_LOCATION	
theCHIVE	Mopub	ACCESS_FINE_LOCATION	(-)
		GET_PHONE_STATE	(-)
	Inmobi	ACCESS_FINE_LOCATION	(O)
TweetCaster for Twitter	Admarvel	ACCESS_WIFI_STATE	(-)
		GET_PHONE_STATE	(-)
	Other	ACCESS_WIFI_STATE	
		GET_PHONE_STATE	
Mp3 Music Download	Startapp	ACCESS_WIFI_STATE	(-)
		WRITE_EXTERNAL_STORAGE	(-)
	Other	ACCESS_WIFI_STATE	
		WRITE_EXTERNAL_STORAGE	
		ACCESS_COARSE_LOCATION	
Tiny Tower	Mobclix	GET_PHONE_STATE	(X)
		ACCESS_COARSE_LOCATION	(-)
	Other	GET_PHONE_STATE	
		GET_ACCOUNTS	
		ACCESS_COARSE_LOCATION	

TABLE IV: Apps that suffered from the permission violation problem. (X) means the ad network requiring that permission. Permission with (O) is the optional permission. (-) represents ad networks that do not mention the permission in document.

information. However, from the perspective of protecting user privacy, it will not cause any damage for PmDroid to block all traffic to ad networks that suffers the permission violation problem.

D. Optional Permission

Another interesting finding is that many ad networks collect private information with the optional permission. As previously mentioned, Android mobile users might refuse to install some apps that acquire excessive permissions. As a result, app developers will refuse to use some ad networks that require a plethora of Android permissions. To avoid such situations, many ad networks choose to provide optional permissions that will not affect the function of the advertising SDK. As we can see in Figure 2, a number of ad networks provide optional permissions for the app developers. Optional permissions are used to provide better advertisements so that it can increase the likelihood that users will click on it. Since ad networks do not have the knowledge about the permissions that apps declare, they must probe the permission first to avoid any crashes. Once the advertising SDKs are aware of the permissions, they can use them to collect data.

Figures 3 and 4 show the data collected by optional permissions of ad networks. We see that the optional permissions act as a vital role for the Android ad network to collect information. In our experiment, almost one-third of the phone state information is collected via optional permissions, and more than half of the location information is collected by optional permissions. Moreover, some ad networks use optional permissions to collect Wi-Fi state. Compared with permission violation, the amount of data collected by optional permissions is much larger than the permissions not mentioned in ad networks' documents. Since the advertising SDK also needs to probe the permissions before using them, we infer that the large amount of permission probing reported in [13] is due in large part to the optional permissions.

VI. Discussion

Based on the results described in Table III, we find that the most serious problem is about location information leakage. Since location-based advertising becomes more and more prevalent, it is not surprising that ad networks attempt to collect location information. An interesting finding is that some ad networks will collect fine location information while only asking app developers to add the permission for coarse location. We believe that this poses a serious threat to user privacy and violates the permissions. One possible reason is that the advertising SDKs obtain a incorrect location provider. For instance, function getBestProvider() will return the provider that best meets a given criteria. Besides location, the permissions about a unique device identifier such as the phone number or MAC address are also stealthily collected by several ad networks. Account information is also of interest to certain ad networks since accounts can safely associate the user with ad networks, especially when users change devices.

Plenty of ad networks prefer to collect sensitive private data with optional permissions. Optional permissions are more flexible since they do not play a "must-grant" role on the running of advertising SDKs. It is interesting to compare the optional permissions with the permissions not mentioned in the documents. No matter what permissions an ad network declares, advertising SDKs can collect private information only if the app contains these permissions. However, the possibility that an app contains an unmentioned permission is lower than an optional permission since app developers request permissions based on the needs of the applications. On the other hand, app developers may prefer ad networks with fewer permissions in case users refuse to install their apps due to over-claimed permissions. App developers can make their own decisions regarding the tradeoff between having better advertisements and minimizing the number of permissions needed for their apps.

VII. Conclusion

The Android permission mechanism provides advertising networks with a chance to probe and misuse their host application's permissions to steal sensitive private information. We propose PmDroid to detect and prohibit ad networks from using undocumented permissions in their official documents to obtain private information. PmDroid does not require any modification on either the apps or the advertising SDKs, and can effectively protect various types of data belonging to 33 permissions from being secretly assessed by third-party ad network companies. We conduct two sets of experiments to measure the permission violation problem using PmDroid on both ad networks' published SDKs and real Android market apps. Our experiment covers 76 different ad network companies, and the results show that the permission violation problem indeed exists in the ad network market. App developers should be careful when choosing the ad network's SDK.

VIII. Acknowledgements

We would like to thank the anonymous reviewers for their detailed and insightful comments. This work was partially supported by ARO grant W911NF-15-1-0287 and ONR grant N00014-15-1-2012. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Ad networks. http://www.appbrain.com/stats/libraries/ad.

[2] Content provider basics. http://developer.android.com/guide/topics/providers/content-provider-basics.html#ContentURIs.

[3] Global advertising expenditures to grow by 5.5% in 2014. http://www.portada-online.com/2014/04/08/global-advertising-spend-to-grow-by-5-5-in-2014/.

[4] Mobile advertising history - from 2.5 pound "brick" to multi-billion dollar industry. http://partners.gamehouse.com/mobile-advertising-history-2-5-pound-brick-multi-billion-dollar-industry/.

[5] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014.

[6] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. PScout: Analyzing the Android permission specification. In *Proceedings of the 19th ACM Conference on Computer and Communications security*, 2012.

[7] M. Becher, F. C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf. Mobile security catching up? revealing the nuts and bolts of the security of mobile devices. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, 2011.

[8] K. Z. Chen, N. M. Johnson, V. D'Silva, S. Dai, K. MacNamara, T. R. Magrino, E. X. Wu, M. Rinard, and D. X. Song. Contextual policy enforcement in Android applications with permission event graphs. In *Proceedings of the 20th Network and Distributed System Security Symposium*, 2013.

[9] L. P. Cox, P. Gilbert, G. Lawler, V. Pistol, A. Razeen, B. Wu, and S. Cheemalapati. SpanDex: Secure password tracking for Android. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[11] 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*, 2011.

[12] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011.

[13] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2012.

[14] J. Han, Q. Yan, D. Gao, J. Zhou, and R. H. Deng. Comparing Mobile Privacy Protection through Cross-Platform Applications. In *Proceedings of the 20th Network and Distributed System Security Symposium*, 2013.

[15] M. Hardt and S. Nath. Privacy-aware personalization for mobile advertising. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, 2012.

[16] 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*, 2011.

[17] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo. Don't kill my ads!: Balancing privacy in an ad-supported mobile application market. In *Proceedings of the 12th Workshop on Mobile Computing Systems & Applications*, 2012.

[18] B. Liu, S. Nath, R. Govindan, and J. Liu. Decaf: Detecting and characterizing ad fraud in mobile apps. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation*, 2014.

[19] D. Liu, E. Cuervo, V. Pistol, R. Scudellari, and L. P. Cox. Screenpass: Secure password entry on touchscreen devices. In *Proceedings of the 11th ACM International Conference on Mobile Systems, Applications, and Services*, 2013.

[20] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. Chex: Statically vetting Android apps for component hijacking vulnerabilities. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, 2012.

[21] 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*, 2012.

[22] V. Rastogi, Y. Chen, and X. Jiang. Droidchameleon: Evaluating Android anti-malware against transformation attacks. In *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security*, 2013.

[23] S. Shekhar, M. Dietz, and D. S. Wallach. AdSplit: Separating smartphone advertising from applications. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[24] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in Android ad libraries. In *Workshop on Mobile Security Technologies*, 2012.

[25] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Permission evolution in the Android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012.

[26] L.-K. Yan and H. Yin. DroidScope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic Android malware analysis. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[27] X. Zhang, A. Ahlawat, and W. Du. AFrame: Isolating advertisements from mobile applications in Android. In *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013.

[28] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang. Vetting undesirable behaviors in Android apps with permission use analysis. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, 2013.

[29] Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.