TRANSACTIONS ON DATA PRIVACY 7 (2014) 27–50

IdentiDroid: Android can finally Wear its Anonymous Suit

Bilal Shebaro, Oyindamola Oluwatimi, Daniele Midi, Elisa Bertino

Computer Science, Cyber Center and CERIAS, Purdue University, West Lafayette, IN 47907, USA

Email: {bshebaro, ooluwati, dmidi, bertino}@purdue.edu

Abstract. Because privacy today is a major concern for mobile applications, network anonymizers are widely available on smartphones, such as Android. However despite the use of such anonymizers, in many cases applications are still able to identify the user and the device by different means than the IP address. The reason is that very often applications require device services and information that go beyond the capabilities of anonymous networks in protecting users' identity and privacy. In this paper, we propose two solutions that address this problem. The first solution is based on an approach that shadows user and application data, device information, and resources that can reveal the user identity. Data shadowing is executed when the smartphone switches to the "anonymous modality". Once the smartphone returns to work in the normal (i.e. non-anonymous) modality, application data, device information and resources are returned back to the state they had before the anonymous connection. The second solution is based on run-time modifications of Android application permissions. Permissions associated with sensitive information are dynamically revoked at run-time from applications when the smartphone is used under the anonymous modality. They are re-instated back when the smartphone returns to work in the normal modality. In addition, both solutions offer protection from applications that identify their users through traces left in the application's data storage or through exchanging identifying data messages.

We developed IdentiDroid, a customized Android operating system, to deploy these solutions and built IdentiDroid Profile Manager, a profile-based configuration tool that allows one to set different configurations for each installed Android application. With this tool, applications running within the same device are configured to be given different identifications and privileges to limit the uniqueness of device and user information. We analyzed 250 Android applications to determine what information, services, and permissions can identify users and devices. Our experiments show that when IdentiDroid is deployed and properly configured on Android devices, users' anonymity is better guaranteed by either of the proposed solutions with no significant impact on most device applications.

1 Introduction

The widespread use of mobile smartphones, such as Android devices, has raised privacy concerns as the use of these devices increases the privacy exposure of users by revealing locations, movements and habits of users. To address such concerns, several anonymous communication technologies, such as onion router mechanisms (e.g. Tor [1]) or secure VPN services (e.g. Hotspot Shield [2]), are today available on smartphones. However just using a network anonymization technique is not sufficient to assure privacy as the operating system together with the mobile applications invoked by users may still release information that may lead to re-identification of the user and/or the device. Even when users are careful and do not provide any identifying data to applications running on their smartphones during anonymous connections, applications can still leak identifying information

without user knowledge. For example, Tor has already warned users to not download files that are automatically opened by third-party applications, such as DOCs or PDFs, as these applications may escape the Tor network relay and access the Internet using a non-torified IP address [3]. As this is a common problem in mobile phones and ordinary computers, different security approaches are required on mobile systems as they introduced several new mechanisms for applications (such as permissions) to access system resources and identifying data.

As an example, suppose that an Android phone user connects to the Tor network and runs the popular Internet radio application "Pandora". Surprisingly, this application requires access to the user's *Contacts* list and *device ID*. When these privileges are given to this application, Tor becomes useless as the "Pandora" servers can easily identify the user through his/her *Contact* list or the *device ID* [4]. Another example is "Angry Birds", one of the top games in the Android market, which requires access to the user's *Location* and *device ID*, regardless of the anonymous network used. In fact, these are very common scenarios. A recent data collection and analysis by Lin *et al.* has shown that out of the top 100 Android applications, 56 use *device ID, Contact* lists and/or *Location services* [5].

Previous research regarding the management and protection of user personally identifying information on smartphones has mainly focused on privacy issues resulting from permissions granted to applications at installation. Approaches have been proposed for identifying privacy risks associated with smartphone applications and for reducing these risks [6, 7]. However, these approaches do not protect all the information that may lead to the user and/or device re-identification, such as user data in applications' data storage. Other research has focused on privacy risks resulting from over-privileged smartphone applications and on building tools that can identify applications not complying with the least privilege security principle [8, 9]. However, these tools are not sufficient as anonymity can still be broken even with minimally granted permissions, and in a few cases even when no permission is granted. The reason is either the lack of protection for all identifying data or the fact that applications store identifying information within their own data storage space.

The design of enhanced anonymity approaches is challenging, as such an approach should fulfill the following requirements:

1. Applications should not be able to bypass restrictions enforced by the solution, and any request for identifying data should comply with these restrictions before data is delivered to the application.

2. The approach should not require the application developers to modify source code, or impose any additional requirement on existing anonymity solutions.

3. Anonymity restrictions should be fully customizable per application to give users complete freedom and flexibility to grant/block access to information and resources.

4. The approach should not cause significant delays in the device functionality that could negatively impact the system performance.

In this paper, we propose two solutions that comply with the above requirements. Each solution offers a different technique that protects anonymous users from being identified when using applications during anonymous sessions. Our first solution is based on data shadowing and consists of choosing the identifying information that needs to be hidden from selected applications. Once one such application requests access to identifying data or resources, the call is intercepted and the returned data is randomized, thus concealing the actual information about the user/device. Our second solution is a sensitive permission manager that controls the access of applications to sensitive permissions at run-time. Such permissions are the ones that grant an application access to identifying information. By dynamically revoking these permissions at run-time, our approach

prevents applications from accessing identifying information during an anonymous session, while keeping them available during normal (non-anonymous) sessions. As part of both our solutions, we design two features; *Fresh Start* and *Intent filtering*. *Fresh Start* prevents applications from leaving any identifying information or traces within their own data storage. With such feature, an application is given the same "environment" that the application would have were the application running on a new device for the first time. This feature removes any opportunity for applications to use their own existing data to identify the device or user. On the other hand, *Intent filtering* prevents applications at run-time from exchanging messages during an anonymous session. With such feature, applications that have access to identifying data will not be able to send such data to applications that have been blocked access to identifying data. This feature removes any chances of designing cooperative applications that try to circumvent anonymity through message exchanges.

We implemented these two solutions on the Android operating system as part of a comprehensive anonymity system, called IdentiDroid. The system also includes a tool called IdentiDroid Profile Manager for administering configuration profiles, each consisting of a set of customized configurations for every installed Android application. These configurations apply once a user activates a profile, and should be used in conjunction when connected to an anonymous network. Users can configure these profiles using any of our proposed solutions and customize them per application.

Our solutions give users control over the applications' access to sensitive data. Under either solution, Android applications will not be able to leak any identifying information. Thus, even when applications are granted access to identifying data at installation time in order to permit installation, users can take away these privileges during their anonymous connections to avoid re-identification.

We deployed IdentiDroid on the latest Android Google Nexus 7 tablet and the Google Nexus 4 phone to compare the device anonymity behavior before and after using IdentiDroid. Our investigation involved 250 applications and several test cases. For anonymity networks, we used Tor [1] as an onion routing based network, and Hotspot Shield [2] as a secure VPN, two of the most popular anonymous communication techniques available for Android devices. Our experimental results show the effect of both solutions on identifiable information and IdentiDroid's impact on the overall system performance.

This paper is organized as follows. Section 2 introduces basic concepts and background information. Section 3 identifies the data and permissions we consider sensitive for the purpose of this work. Section 4 introduces the design of our proposed solutions followed by the implementation and technical details in Section 5. Section 6 reports results of experiments to assess the efficiency and accuracy of IdentiDroid and its impact on applications. Section 7 discusses our security analysis of IdentiDroid. Section 8 further discusses our analysis and results. Sections 9 and 10 discuss related work and outline future work, respectively.

2 Background

In this section, we cover related background information on Android operating system and how it manages its security policies, and some basics on anonymous networks.

2.1 Android

2.1.1 Operating System and API

The Android operating systems are derived from Linux based kernels and have enhanced support for security and privacy [10]. Android is designed with a multi-layered security infrastructure (Figure 1), which provides developers a secure architecture to design their applications.

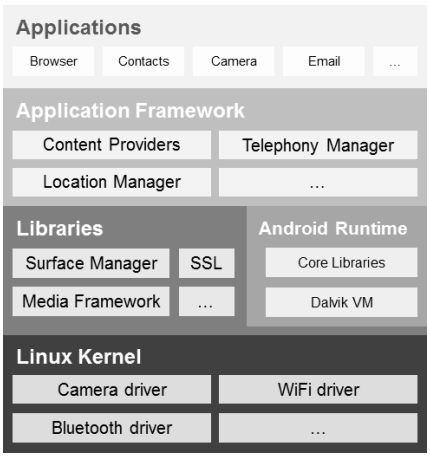


Figure 1: **Android Software Stack.**

Android applications are sandboxed and run inside the Dalvik Virtual Machine, which isolates each application's processes and data. Each application is assigned a user ID (UID) with assigned privileges, and is given a dedicated part of the file system for its own data. The processes of applications with different UIDs run separately and are isolated from each other and from the system. While applications have limited access to device resources, developers can explicitly request access for their applications through the Android permission system.

2.1.2 Permission System

The Android permission system controls which application has the privilege of accessing certain device resources and data. Application developers that need access to protected Android APIs need to specify the permissions they need in the *AndroidManifest.xml* file which, if inaccurately assigned, can increase the risks of exposing the users' data and increase the impact of a bug or vulnerability.

Each application declares the permissions listed in its *AndroidManifest.xml* file at the time of installation, and users have to either grant all the requested permissions to proceed with the installation, or cancel the installation. The Android permission system does not allow users to grant or deny only some of the requested permissions, which limits the user's control of application's accessibility.

As of Android API 17 version 4.2, there are currently 130 permissions that users can grant to applications, grouped into 30 permission groups as defined by the Android Manifest.permission_group class [11].

2.1.3 Android Application Components

Every Android application is composed of four essential components: Activities, Services, Content Providers, and Broadcast Receivers. Each component represents a different entry point for the system to interact with the application. For this reason, every application component must be declared in the application's *AndroidManifest.xml* file.

The Android operating system allows any application to start another application's component, thus allowing developers to use functionalities of other applications. For instance, developers can capture images in their own applications by calling the device's camera activity instead of building their own photo capturing activity. Application components (except content providers) can only be

activated by sending the Android system a message, called *Intent*, that specifies the intent of starting a particular component.

A content provider acts as a global instance in each application that manages a shared set of application data. Such data can be stored in the file system or any other persistent storage location that applications can access. Thus in order to secure such data, content providers are the only components that cannot be activated through *Intent* messages, and can only be activated through the *ContentResolver* object. In Section 5, we will show more details on how we implement the content provider component in developing IdentiDroid Profile Manager.

2.2 Anonymous Networks

The basic functionality of anonymous networks is to hide the user's IP address from all connected Web services and Internet traffic. What differentiates these networks from each other is the technique used to hide the source of such traffic. For instance, Tor uses onion routing to hide the source of TCP traffic based on layered encryption. Each Tor client chooses three Tor nodes, or onion routers (OR), which will route the client's Internet traffic. *Hotspot Shield* is a another popular VPN service that offers its users a secure and private Internet usage. As web servers can only see the IP address of the proxy server rather than the client IP address, *Hotspot Shield* secures its users from untrusted ISPs or public networks by encrypting all web transactions through HTTPS. This also allows one to bypass any censorship or firewalls. While VPN solutions are widely offered for network anonymization purposes, the users' privacy is in the hands of the proxy server.

We believe that sophisticated users who demand anonymity on their mobile phones and who are familiar with anonymity techniques and configurations will find it easy to set the proper IdentiDroid configurations. In addition, we provide through IdentiDroid some built-in profiles that are pre-configured for quick use, even though we still recommend users to customize their own profiles based on their personal requirements. Moreover, we can also refer to existing usability techniques [12–15] that can be used to offer regular users with preset and adapted IdentiDroid configurations.

3 Sensitive Data and Resources

For the purpose of this work, we did an extensive study to understand what data is used by Android applications that can identify users or devices and thus undermine their anonymity. Furthermore, we studied how applications may have access to such data and their methods of accessing it. In this section, we first define "sensitive data" as it pertains to anonymity and then classify it according to the methods applications use to access it. This is necessary due to the similarity of some access methods, which is reflected in the implementation of our solutions.

3.1 Sensitive Data Classification

We define "sensitive data" as any set of information that can be used to uniquely identify a device or the user of the device. Such data can either be in the form of identifiers (self-identification variables) or quasi-identifiers that can re-identify a user or device when combined with other data [16]. This includes, but is not limited to, information about the device itself, underlying hardware, services provided by the operating system, and of course personal data pertaining to the user. As such, protecting device and user anonymity is far from trivial.

We categorize sensitive data into four groups, as data in each group is accessed by a different method and thus needs a different protection mechanism as we discuss in Section 5.

Data	Method	Related Permission(s)
System Info		
Android_ID	Settings.System.getString(Android_ID)	-
IMEI or CDM or ESN	TelephonyManager.getDeviceId()	READ_PHONE_STATE
Current Cell Location	TelephonyManager.getCellLocation()	READ_PHONE_STATE
Phone Number	TelephonyManager.getLine1Number()	READ_PHONE_STATE
IP Address	WifiInfo.getIpAddress()	ACCESS_WIFI_STATE, ACCESS_NETWORK_STATE
User Data		
Contacts	ContentResolver.Query()	READ_CONTACTS, WRITE_CONTACTS
Photo Albums	ContentResolver.Query()	READ_EXTERNAL_STORAGE
SMS	ContentResolver.Query()	READ_SMS, RECEIVE_SMS, SEND_SMS, WRITE_SMS
Bookmarks/History	ContentResolver.Query()	READ_HISTORY_BOOKMARKS, WRITE_HISTORY_BOOKMARKS
Resources		
Camera	Camera.open()	CAMERA
Location	LocationManager.getLastLocation()	ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, ACCESS_LOCATION_EXTRA_COMMANDS, ACCESS_MOCK_LOCATION
WiFi MAC Address	WifiInfo.getMacAddress()	ACCESS_WIFI_STATE

Table 1: **Some of the sensitive data considered, with access method and related permissions.**

- **System Information:** It consists of all the information concerning the system state and identity. Such data is typically accessed by calling value-returning methods for each data entity.
- **User Data:** It consists of common data generated by the user, such as contacts and SMSs. Such data is typically accessed by calling methods that return a *Cursor* to requested databases.
- **Resources:** It consists of resources provided by the device, such as camera and GPS. Resources are typically accessed by calling their corresponding activity or service.
- **Application Data:** It consists of the data stored and managed autonomously by the applications. This data represents the application data storage and can be only accessed by the application itself.

Table 1 shows some of the sensitive data we consider in our work, following our categorization. We also define of what is considered "non-sensitive" data or resources. In general, non-sensitive data corresponds to common data that is not unique to a device or user of the device, which is generally duplicated in many devices. Examples of non-sensitive data could be the OS version, modem firmware, *etc.* Similarly, non-sensitive resources correspond to resources that do not generate sensitive data. Examples of non-sensitive resources could be the gyroscope, barometer, thermometer, or hygrometer (to measure humidity).

3.2 Sensitive Permissions

As mentioned earlier, in order for applications to execute functions involving the phone hardware, settings, or user data, specific permissions must be granted. In our work, we define sensitive permissions as permissions that, when granted, permit access to sensitive data and resources. Among the 130 available application permissions, we identify 41 permissions as sensitive. Such permissions belong to 14 permission groups out of the available 30 groups.

Permissions such as READ_PHONE_STATE are considered sensitive as they allow applications to access system information and easily identify the device through several unique identifiers such as

the IMEI for GSM phones. Furthermore, permissions associated with the Bookmarks, Calendar, Messages and Social_Info permission groups are also sensitive as they allow one to access to the user databases containing private and confidential information. Finally, resource permissions are considered sensitive when access to the associated resource reveals specific information about the device or user, such as its location and wifi state. Table 1 lists some of the sensitive permissions associated with the sensitive data groups defined in subsection 3.1.

After we categorized what permissions are to be considered sensitive, we tested 250 applications to investigate how many applications request such permissions. Table 2 shows the results of our investigation listing the top 20 most requested sensitive permissions, with appendix A listing the complete set of sensitive permissions.

Permission	Req. apps #	Req. %
READ_EXTERNAL_STORAGE	160	63.40%
WRITE_EXTERNAL_STORAGE	159	63.13%
INTERNET	156	62.07%
READ_PHONE_STATE	124	49.07%
ACCESS_NETWORK_STATE	103	41.11%
ACCESS_WIFI_STATE	69	27.19%
WRITE_SETTINGS	51	20.03%
READ_CONTACTS	46	18.04%
ACCESS_FINE_LOCATION	41	16.18%
ACCESS_COARSE_LOCATION	36	14.46%
CHANGE_WIFI_STATE	35	14.06%
GET_ACCOUNTS	31	12.33%
CHANGE_NETWORK_STATE	29	11.27%
CALL_PHONE	26	10.34%
BLUETOOTH	24	9.42%
WRITE_CONTACTS	22	8.89%
READ_SMS	21	8.49%
CAMERA	18	7.69%
BLUETOOTH_ADMIN	18	7.29%
READ_SYNC_SETTINGS	17	7.16%

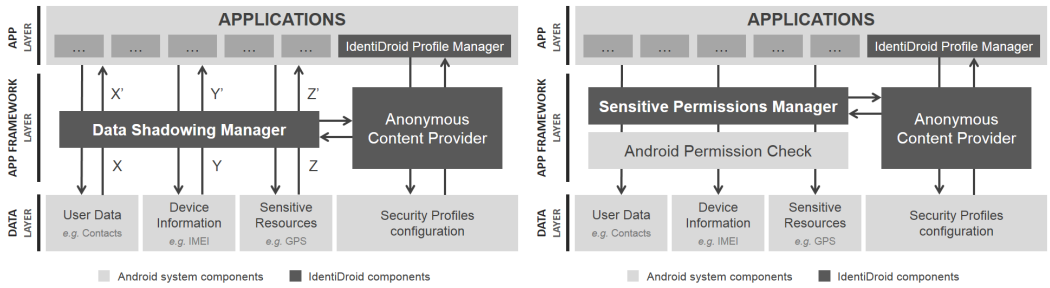
Table 2: **Top 20 requested sensitive permissions from the top 250 applications on Google Play Store.**

4 Overview of IdentiDroid

In this section, we outline the design details of IdentiDroid and the necessary components that we build our solutions upon. Referring to the Android software stack (see Figure 1), any interaction that occurs between the applications and other device components such as the Android kernel, libraries, and device hardware, must pass through the application framework layer first [17]. For this reason, we integrated our solutions within the Android application framework layer to guarantee that no applications can bypass the constraints applied by IdentiDroid (see Figure 2).

4.1 Data Shadowing Manager

Our first solution (see Figure 2(a)) is a data shadowing manager that enables the user to choose which information needs to be hidden from each application that might access it. The design of this solution requires to understand all possible ways and methods by which an application can access sensitive data. We achieved this by first referring to the Android API classes [18] as it is the main reference for developers who wish to call APIs in their applications. Second, we located all the API



(a) Data Shadowing Manager.

(b) Sensitive Permission Manager.

Figure 2: **Integration of the proposed solutions into the Android Application framework.**

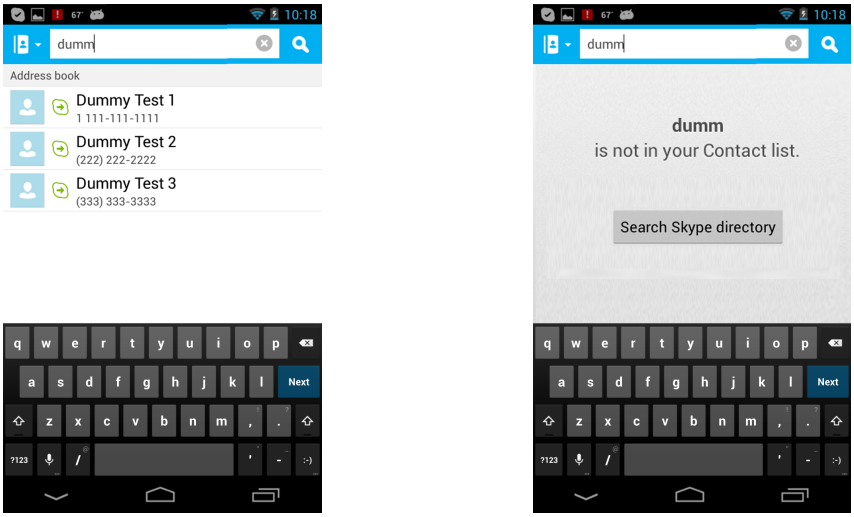
methods of our interest on the Android system source code and performed our changes. Finally, we tested these changes on applications that access these APIs to confirm its functionality. As an example of multiple methods accessing a resource, applications can use the *Camera* resource either though an *Intent* call or through the *Camera* class.

We noticed that several API methods share similar properties and thus can be shadowed in the same manner, as discussed in Section 3. We now show how we shadow each category according to its access method and functionality.

Device information. Shadowing device information returns a fake value for the requested information to any application configured to be shadowed. The returned value is randomized upon each data request such that different applications receive different device information. For example, if two applications use the *Device ID* to identify the same device they are running on, the applications will receive two different random *Device IDs*. Also if the same application is executed multiple times, it will receive each time a different random *Device ID*. The device information that we shadow includes *IMEI (GSM) or MEID or ESN (CDMA), Phone Number, Subscriber_ID, WiFi MAC Address, Android_ID, etc.*

User databases. Shadowing user databases means returning an empty list of database records to any application trying to access data that was previously entered and stored when the device was functioning under normal modality. For example, if an application like *PicsArt* tries to load images saved in the photo albums, the application will receive no images as if there were no pre-saved images. Accesses to phone contacts, SMS/MMS, calendar, music, videos, browser bookmarks/history and accounts are handled in the same way.

Resources. Shadowing device resources consist of either: (a) not providing access to the resource by intercepting the application request; or (b) returning modified or fake information, and sometimes no information. The techniques used for shadowing device resources defined in (a) and (b) are active at all times. What really decides which one is triggered depends on how an application is calling or using a resource, whether using an intent call to an activity with access to that resource or having its own methods in accessing the resource itself. Therefore, the choice of whether to use solution (a) or (b) is determined at run-time. For example, consider an application that is configured to be shadowed from using the *GPS* and *Camera* resources. If this application issues an intent call to the *Camera* stock application every time it needs to capture an image, then our system will dynamically choose solution (a) to intercept and drop the intent call to the *Camera* application. On the other hand, if this application has its own interface and methods to retrieve and load *GPS* data for location purposes, then our system will dynamically choose solution (b) to return fake location coordinates from the *GPS* services.



(a) Skype with "read contacts" permission granted

(b) Skype with "read contacts" permission revoked

Figure 3: **Sensitive permission manager's effect on Skype.**

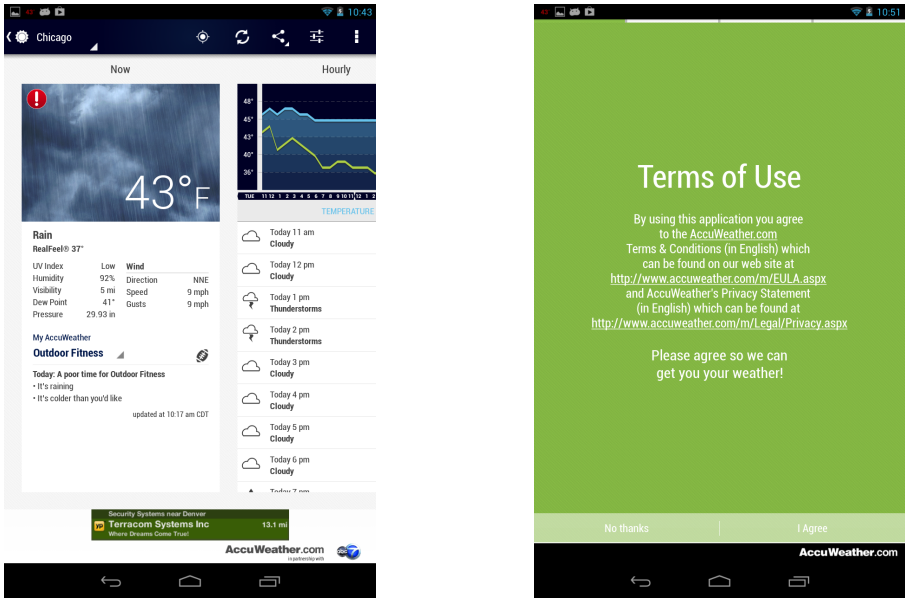
We carefully shadow all the above sensitive data and resources (discussed in details in Section 5) so that there is no significant impact on the application's functionality or on the system performance as we show in Section 6.

4.2 Sensitive Permission Manager

Our second solution is a sensitive permission manager that controls the applications access to sensitive permissions at run-time (see Figure 2(b)). Even though Android applications are granted all their required permissions at installation time, our solution can dynamically block permissions when these permissions are required at run-time by applications. For example, if an application like "Skype" requests to load the contacts address book saved on the device, the Android operating system will check at run-time if this application has been granted the permission READ_CONTACTS. However, if "Skype" has been configured through IdentiDroid Profile Manager not to have access to the contacts list, the permission call is dynamically intercepted and denied at run-time, thus blocking the application access to such list. Figure 3 shows screenshots of the "Skype" application demonstrating its behavior upon permission blocking.

4.3 *Fresh Start* Feature

In addition to the previous solutions, our approach also includes a feature, called *Fresh Start*, which is combined with both the above solutions to protect users from being identified by applications that leave identifying information or traces within their own data storage. This feature makes user applications believe that they are running on a new device for the first time. This is extremely useful as applications will appear as if they do not have any pre-saved data on the device whenever either solution is used, while original saved data will be recovered once the user goes back to normal (non-anonymous) modality. The use of this feature prevent applications, that normally leverage their



(a) *Fresh Start* not activated (b) *Fresh Start* activated

Figure 4: **Screenshots from the "AccuWeather" application.**

own existing data, from being able to re-identify the device or user when the user is connected to an anonymous network.

For example, consider an application like "AccuWeather" that has been installed and configured to display the weather for *Chicago, IL*. Once the user connects to an anonymous network and activates any of our proposed solutions, the *Fresh Start* feature is activated and causes "AccuWeather" to appear as if it has been just installed. In this case, "Accuweather" would display the *Terms and Conditions Agreement* screen with no predefined preferred cities. Once *Fresh Start* is deactivated, "AccuWeather" will display its original configuration. Figure 4 displays screenshots from the "AccuWeather" application showing the effect of the use of the *Fresh Start* feature before and after being activated.

4.4 *Intent Filtering* Feature

The *Intent filtering* feature is also part of both solutions, which manages data and messages sent/received between applications. This component gives the user control over what intent messages can be exchanged between applications, as the user may wish to block the exchange of identifying data during an anonymous session. Without *Intent filtering*, an application that has been granted access to identifying data during an anonymous session may pass such data to other applications that are denied access to identifying data. As a result, such cooperating applications may break the anonymity of the user/device if at least one application is given access to identifying data. Therefore it is a necessity to include *Intent filtering* in both IdentDroid solutions to guarantee the anonymity of user/device during an anonymous session.

4.5 IdentiDroid Profile Manager

IdentiDroid Profile Manager is an Android application, also part of our approach, that hosts customizable configurations represented as profiles. Each profile has four main category settings: Permission Manager, Shadow Manager, Fresh Start Manager, and Intent Filtering Manager as shown in figure 5. Each of the aforementioned setting is unique per profile, giving users the ability to tailor different profiles for protecting various aspects of their identity against a subset of user-installed applications. The managers populate a list of currently-installed user applications and allow users to choose which applications they desire to deny access to sensitive data and resources. This benefits the user in terms of convenience, ease of use, and of course tailoring security configurations to protect various aspects of anonymity.

Even though the best practice of anonymity is to deny access to sensitive data from all applications, users can create multiple profiles and configure them differently according to their needs. A user must activate his/her profile in conjunction with the user's connection to an anonymous network. Once the profile is activated, the *Fresh Start* and *Intent filtering* features are activated and the user configured settings are applied per application according to the chosen solution, whether Data Shadowing Manager or Sensitive Permission Manager.

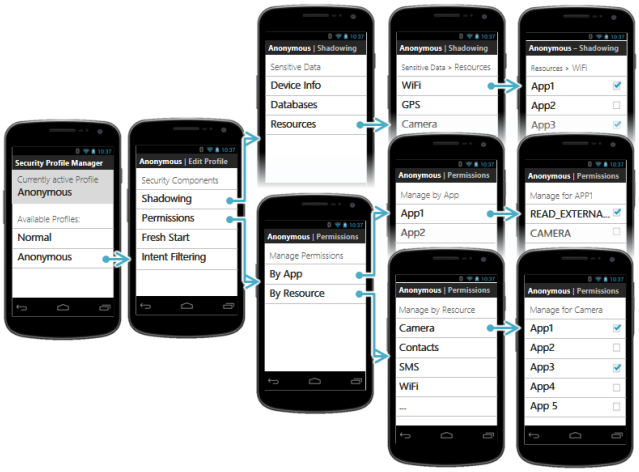


Figure 5: **GUI structure of the configuration profiles.**

5 Implementation

In this section, we introduce the technical details of the implementation of IdentiDroid and the components used to build the IdentiDroid Profile Manager.

5.1 Anonymous Content Provider

The IdentiDroid Profile Manager manages access control lists (ACL) which are accessed by the Data Shadowing and Sensitive Permission Managers. As we needed to modify the Android operating system to prevent applications from accessing certain data and resources at run-time, we had to make it possible for the system to read these ACLs. For this reason, we created

our custom *AnonymousContentProvider* Java class that allows the system framework to access IdentiDroid Profile Manager's ACLs in a controlled and secure manner. When IdentiDroid needs to decide whether or not to deny sensitive data or resource access to an application, it needs to send several parameters to our *Anonymous Content Provider* including whether shadowing or permission management is utilized. Parameters such as the accessed resource, the application accessing the resource, and the solution currently activated, are passed via the ContentProvider's *call (Uri uri, String method, String arg, Bundle extras)* method. We register the URI of our Anonymous Content Provider as "com.android.anonymous", which is declared in the *AndroidManifest.xml* file within IdentiDroid's "provider" tags.

5.2 Shadowing Sensitive Data

Data shadowing performs data obfuscation; it obscures user-identifying data from applications and replaces it with data that does not represent the user/device. We modified Android APIs that applications use to access sensitive data, according to the sensitive data classification discussed in Section 3.

Device information. We modified the methods in classes where device information may occur, specifically replacing their return values with random ones. Our modifications were mainly applied to the *TelephonyManager* class that contains the *device ID, phone number, etc...*, the *WifiManager* class that contains *wifi state, connection info, etc...*, and the *SettingsProvider* class containing the *Android ID*. For example, if any application attempts to retrieve the Android ID string through calling either *Settings.Secure.getString(ANDROID_ID)* or *Settings.System.getString(ANDROID_ID)*, each call will return a new randomized 10 byte hex alphanumeric string.

User databases. Access to complex resources provided by the Android framework usually involves access to a relational database. We modify the *query()* method of the *ContentResolver* class by intercepting application requests to the database and returning a *Null Cursor* object instead of the expected Cursor object. The *Null Cursor* represents an empty dataset, such as an empty list of contacts entries or SMS messages.

Resources. Resources are shadowed by either returning fake values or by ignoring the requests made by applications. As an example of the first behavior, we modify the *getLastKnownLocation()* method in the *LocationManager* class to return random longitude and latitude values that reflects a fake location of the device. As an example of the second behavior, an application can access the device's camera by calling the Activity class *startActivityForResult()* method passing a *camera* intent. Therefore, we modify the *startActivityForResult()* method to intercept the camera intent request and to ignore it if the user configured to shadow camera for this application, as if the intent was never sent. This approach can be applied to any blacklist of intents we choose and is thus extensible. As this will not cause the application to crash, we discuss the shadowing impact on applications in Section 6.

5.3 Managing Sensitive Permissions

IdentiDroid supplements the Android operating system with permission management: users have the ability to grant/revoke permissions per application post-installation.

Before to actually accessing a resource, the *ActivityManagerService* class verifies if the application has the right permission to access it. The verification process involves invoking the method *checkComponentPermission(String permission, int pid, int uid,...)* where the "permission" parameter is the permission associated with the resource that is requested and thus being checked. The "pid" and the "uid" parameters identify the process and/or application requesting the resource. We apply our modifications to this permission check method to implement our run-time restrictions. Since

the "uid" may be shared by applications signed by the same developer, we perform our permission management based on the package name rather the "uid" to avoid confusion with other applications. Figure 6 shows the interaction between the Sensitive Permission Manager with the application and data source, showing the steps applied upon each permission check. We retrieve the application name via the UID or name of the process in which the method is being invoked. The system then calls our Anonymous Content Provider's *checkResourceAccess()* method with the resource being accessed and the requesting application. If the boolean value returned is *false*, then the constant *PackageManager.DENIED* is returned by *checkComponentPermission(...)* allowing no access to that resource.

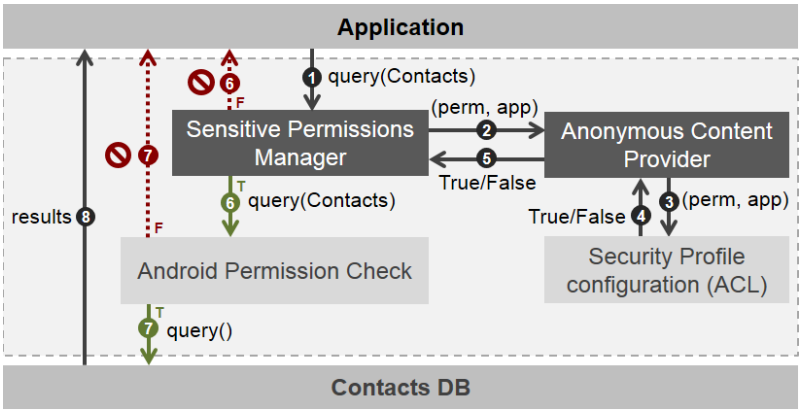


Figure 6: **Steps performed to manage sensitive permissions.**

5.4 Fresh Start

The goal of the *Fresh Start* feature is to restore an application to the state it was in when it was newly installed. *Fresh Start* temporarily relocates the data files of an application by moving its data files and directories to an unaccessible directory. Inaccessibility to such directory is achieved by giving no access permissions for this directory to any application on the device. The same process occurs in the reverse: when another profile is set or the current security profile is deactivated, each application's data files and directories will be restored to their original location. As some application data may still reside in the cached files, *Fresh Start* also deletes the application's cache data and terminates any related running process once a user activates or deactivates a profile. In addition, applications configured under *Fresh Start* are denied access to external storage during anonymous sessions, and are given back their original accessibility upon deactivating the anonymous profile.

5.5 Intent Filtering

The goal of *Intent filtering* is to block message exchanges at run-time between applications that may contain identifying data and may reveal the identity of the user/device. Intent messages are one of the common forms for inter- and intra-communication between application components, sent via three methods: startActivity(), sendBroadcast(), and startService(). Denying an application from sending intents is simply a matter of intercepting the intents when the aforementioned methods are called by applications. Intent filtering provides the user the ability to prevent an application component from broadcasting any possible sensitive information.

Data	Normal Connection	Anonymous Con. + Shadowing	Anonymous Con. + Permission Blocking
System Info			
IMEI/MEID/ESN	353850932165477	1234567890ABCDE	Access Denied
Phone Number	111-111-1111	123-456-7890	No Access to Phone Number
WiFi MAC Address	10:BF:48:F2:7E:EB	Null via null WiFiInfo	No Access to MAC Address
Android ID	A23BF6FD34	ABCDEF1234	A23BF6FD34
User Data			
Contacts	Access to Contacts	Empty List of Contacts	No Access to Contacts
Photo Albums	Access to Photo Albums	Empty List of Photos	No Access to Photos
SMS	Access to SMS	Empty List of SMS Messages	No Access to SMS
Calendar	Access to Calendar	Empty List of Events	No Access to Calendar
Account Manager	Access to User Accounts	Empty List of Accounts	No Access to Accounts
App Data			
Logs	Access to Log Entries	No Log Entries	Access denied to Logs
Application Storage Data	Access to application data	Appears as newly installed application	Appears as newly installed application
Resources			
Camera	Access to	Unable to open to camera via intent	No Access to Camera
Location	41.103807, -85.399449	37.428434, -122.072382	No Access to Location services
Microphone	Access to Record Audio	Unable to open Audio Recorder	No Access to Microphone

Table 3: **IdentiDroid's effect on some of the sensitive data.**

6 Experimental Results

We performed different sets of experiments to validate the functionality of IdentiDroid and its impact on the device applications. We designed the IdentiDroid Profile Manager in a way that allows us to configure and test each solution separately or combined. Our experiments were performed on the Android emulator and on real Android devices, the Google Nexus 7 tablet and the Nexus 4 phone. Our tested application set consists of the top free 250 applications downloaded in August 2013 from the Google Play Store listed under *All Categories*, representing the set of most common applications downloaded by users.

Our first experiment tests the effectiveness of each solution on the device applications to prove that no accessible data by applications can identify the user or device. Our second experiment tests the performance of our solutions by evaluating the timing overhead introduced by IdentiDroid compared to the Android stock operating system. The third experiment tests the impact of shadowing sensitive data and resources on user-installed applications. The fourth experiment tests the impact of permission revoking from applications that request a sensitive permission. Each experiment is conducted with the help of the Android Debug Bridge (ADB) utility by logging the application events using the command "adb logcat".

Experiment 1: Testing IdentiDroid on Applications - Expected Behavior Validation Based on the sensitive data classification we defined in Section 3, we tested each of our solutions separately on our set of the top 250 applications. Table 3 displays our experimental results showing the values returned to these applications and the actions taken by IdentiDroid. For demonstration purposes, the shadowed values displayed are of the general form $1234567890ABCDEF$, as they are supposed to return random numbers whenever an application accesses such data.

Experiment 2: Overhead Performance of IdentiDroid The goal of this experiment is to evaluate the performance of IdentiDroid against the stock Android operating system. We wanted to measure the overhead in execution times for the various methods that applications would call to access sensitive data. For each of these methods we measured the elapsed time from when the application requests the data until the data is retrieved. We first measured the elapsed time of the stock operating system methods and then the duration of the same methods when each of our solutions is activated. Each measurement was taken 8 times then averaged. Table 4 reports in milli-seconds (ms) the

Method	Before	After	Overhead
AM.checkComponentPermission()	0.0071	3.958	3.951
TelephonyManager.getDeviceId()	0.0544	5.648	5.594
Settings.getString(Android_id)	0.0093	0.371	0.362
WifiManager.getConnectionInfo()	0.0313	3.737	3.705
startActivityForResult()	0.0064	15.553	15.547
startActivity()	0.0159	11.908	11.892
startService()	0.0082	5.727	5.719
sendBroadcast()	0.0098	6.859	6.849

Table 4: **Time overhead (in ms) for some of the core Android methods that were modified in IdentiDroid.**

time overhead of these methods. As the displayed values show, the overall delay introduced by IdentiDroid is not perceivable by the end-user.

Experiment 3: Impact of Data Shadowing on Applications The goal of this experiment is to evaluate the impact of the shadowing solution on a real-world case scenario, that is, a device containing the top 100 applications from the Google Play store under *All Categories*. It is necessary to understand the overall effect of shadowing data and resources on user applications and the cost the user is paying for his/her identity protection.

The evaluation of each application is carried as follows: For every application, we make a checklist of its major functionalities based on what is listed in the application description in the *Google Play* market. For example, the major functionalities of the *Skype* application are video calls, voice calls, text messaging, phone calling, *etc.* Given the checklist of each application, we manually test each application in order to monitor, through the *ADB logcat* utility, all possible activities and services each application has on the Android device, as we need to record the effect of using IdentiDroid whenever they access identifying data or request sensitive permissions. For this purpose, we classify our evaluation into three categories:

1- **No Impact:** none of the application's major functionalities is affected.

2- **Partially Functional:** some major functionalities of the application are affected and cannot be used, with at least one major functionality that is still not affected. e.g. no camera access for the Skype application, but text messaging still works normally.

3- **Useless:** all of the applications major functionalities are broken, that is, the application becomes totally useless. e.g. no microphone access for the sound recorder application.

Figure 7 displays the impact on user applications using the most strict shadowing profile, which consists of shadowing all sensitive data and resources. On average, 92% of the applications had no impact, 6% of the applications became partially functional, and only 2% of the applications became useless when all resources have been shadowed. As these results are strictly related to the functionality of applications, we briefly discuss the shadowing effect on the usefulness of applications in Section 8.

Experiment 4: Impact of Permission Revoking on Applications In this experiment, we performed a stress test on the sensitive permission solution and observed the impact of permission revoking on applications that requested sensitive permissions. For this test, we forced each application to use all its functionalities to ensure that the sensitive permission is requested. We performed our tests on 100 Android applications and used the same evaluation methodology considered in Experiment 3. Figure 8 displays the impact on user applications using the sensitive permission stress test. On average, 79% of the applications had no impact, 11% of the applications became partially functional, and 10% of the applications became useless.

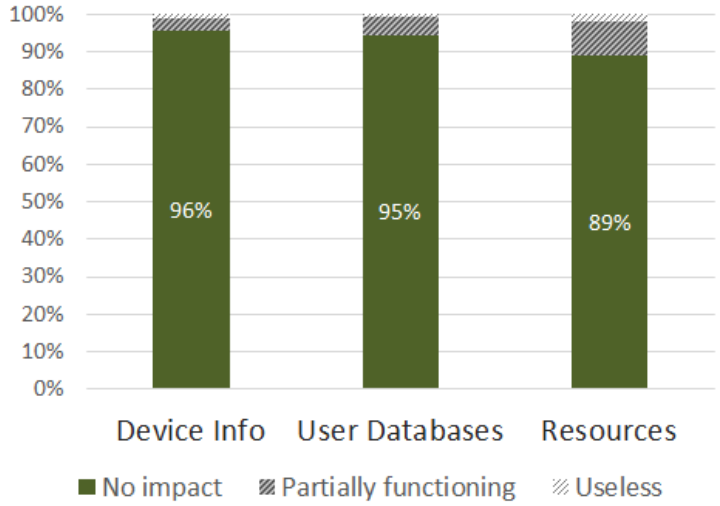


Figure 7: **Impact of data shadowing on applications.**

In the process of evaluating the impact of permission revoking on applications, some applications crashed while performing the permission stress test. Although the 100 Android applications that were evaluated are chosen to be among the ones that did not crash, we extend our evaluation to test a larger set of applications and study the impact of revoking sensitive permissions individually. Specifically, we are interested in whether or not an application crashes as a result of being denied a permission that was initially granted at installation time.

We performed our experiment on our set of 250 Android applications and use our Sensitive Permission Manager to deny the top 20 sensitive permissions individually. We used the ADB logging utility to view the permission being revoked when the *checkComponentPermission()* method is called.

Figure 9 shows the percentage of application crashes upon performing the stress test on each permission. We counted an application as crashing even if it crashed during the execution of one minor functionality. The main cause of these crashes is due to the developers' mishandling the denial of previously granted permissions. We noticed that the ACCESS_NETWORK_STATE permission is one of the most frequent permissions that is not exception-handled by the developers and causes 40% of the crashes, as observed in Figure 9. This is because most applications intend to communicate through Internet and developers always assume that such permission is available and granted to their applications.

Preventing applications from performing sensitive actions or process sensitive data is achieved by returning the PackageManager.PERMISSION_DENIED value at run-time. However, a security exception is thrown when such constant is returned and this might cause problems to some applications. The reason is that developers usually do not expect their applications to be revoked a permission they requested because users must grant all permissions in order to install the applications. Therefore, when a permission that was initially granted is revoked, applications have the likely chance of crashing, with the operating system subsequently force-closing it. Application crashing can be prevented if error-handling is added whenever an application attempts to access sensitive data or resources. In fact, throughout testing several application versions, we realized that the number of application crashes has been decreasing over time. This is because developers are

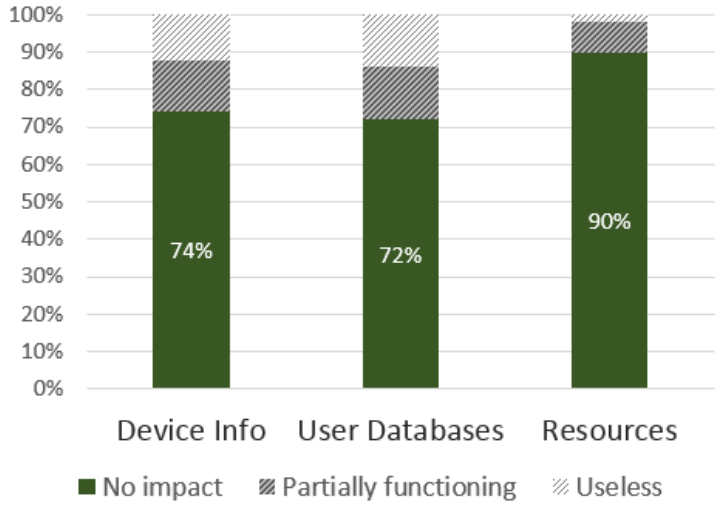


Figure 8: **Impact of permission revoking on applications.**

now aware that not having the permission error-mishandling script is causing several application crashes. A script is thus being added in their application updates especially with the evolvement of many permission restriction techniques. We observed examples of error-handling in several applications when an initially granted permission was denied, as shown in Figure 10.

Shadowing *vs.* Permission Revoking As both solutions protect the user/device identifying information, we also need to make sure that the device does not become useless. For this reason, we tested the overall behavior of each solution on a real-world case scenario with a device containing the top 100 applications from our application's set, and compared each solution's impact in protecting every sensitive data category.

We weighed the behavior of all the applications tested by assigning a score from according to the outcome, with a score of 1 for applications that crashes or become useless, 6 for applications that become partially functional, and 10 for no observed impact.

Figure 11 shows the cumulative score obtained for each of the two solutions with respect to the different categories of sensitive data. As the underlined score determines the better solution, the score are very close which denotes the strength of each solution separately. For users who demand the best tradeoff between privacy protection and user experience, the best solution is to combine the permission and shadowing solutions in order to better protect the users anonymity, an option already available within IdentiDroid Profile Manager application.

7 Security Analysis of IdentiDroid

In this section, we present a security analysis of the IdentiDroid system to analyze possible threats from malicious users or applications that can bypass our mechanisms for the protection of personally identifying information. The aim of our security analysis is to mitigate these threats and discuss how likely for these threats to occur and affect our customized Android system.

Colluding Applications. In Android, each device application is assigned a unique UserID (UID) that the system uses to refer to an application. However, if two applications are created and signed

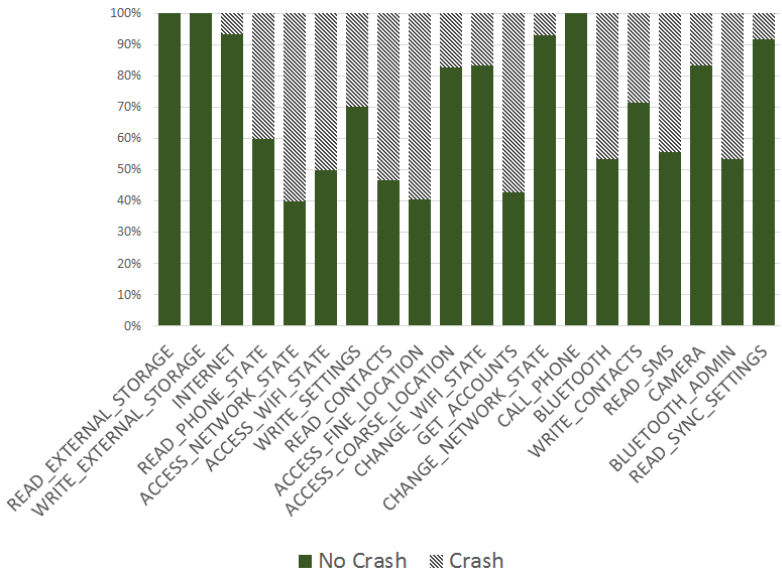
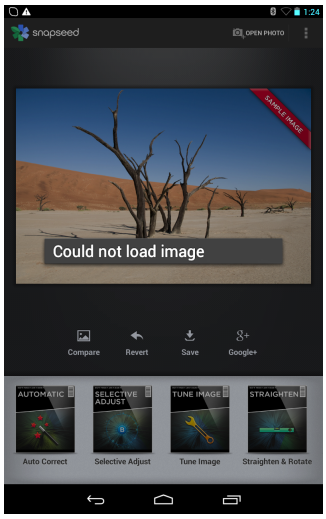
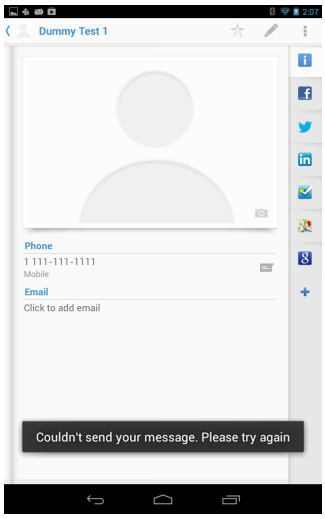


Figure 9: **Impact of permission revoking on applications.**



(a) Snapseed app. error handling when access to external storage is blocked.



(b) Contacts+ app. error handling when sending SMS messages is blocked.

Figure 10: **Screenshots of application's error handling after permission revoking.**

by the same developer, the system will give both applications the same UID, which gives these applications the ability to share the same processes if needed. Because of that, applications with the same UID may collaborate together and can break the anonymity of the user or device if at least one of these applications is not restricted access to sensitive data. For that reason, our IdentiDroid system follows the same mechanism that the Android system uses in enforcing its security policies and uses the application UIDs to block all package names associated with a UID. For example, if

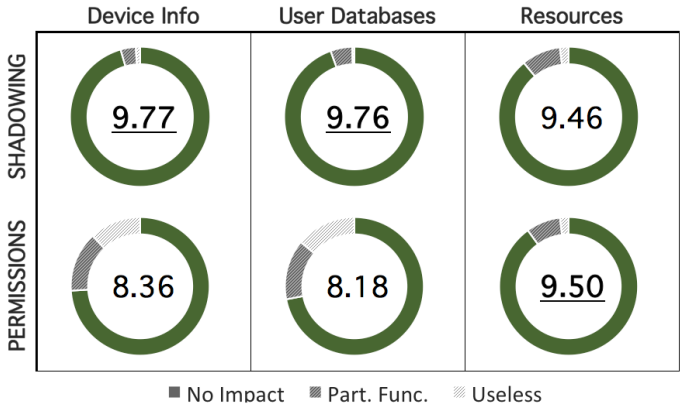


Figure 11: **Shadowing vs. Permission revoking score results.**

one application has been blocked access to the *Contacts* list, then all applications that share the same UID with the blocked application are also blocked access to the *Contacts* list.

Bypassing Permissions via NDK. Android's Native Development Kit (NDK) is a toolset that allows Android developers to build portions of their applications in native code or port existing libraries written in C and C++. Even though developers usually refer to Java code when calling higher-level functionalities in the Android system, they may prefer to write lower-level system calls using C or C++. In particular, by using an external standard C library, developers can call functions such as *connect(), read(), socket(), write(), etc.* However, if a malicious developer tries to use native code in order to bypass the IdentiDroid's permission configurations set by the user, such attempts will fail given that the Android operating system will always check at run-time for whether an application process has the appropriate permissions when accessing any resource. This is achieved through the Android kernel that checks if the application processes are in the right resource group [19, 20]. For example, an application written via NDK that needs to set up a socket connect will need to be granted the INTERNET permission, and therefore its process should have the Internet group ID (GID). If this application is blocked access to the INTERNET permission via IdentiDroid, then the *connect()* process will not have Internet GID and therefore will fail to setup the socket connection.

Data Forwarding. Many Android applications share and exchange data with each other via *Intent* messages. This can cause a rare but possible threat to IdentiDroid users as some malicious applications that has been configured not to have access to identifying data might request identifying data from other applications with access to such data. However, using our *Intent filtering* feature, we intercept and disable messages containing identifying data to be exchanged between applications. Another form of applications data sharing is when applications create world writable/readable files. However, our implementation is based on Android API level 17 in which the world writable/readable capability is deprecated [21].

Accessing ACL. The Android operating system protects content providers and the data they store by using both static and dynamic permissions. Information stored by all content providers is, by default, protected using static declarations of read and write permissions. These static declarations can be applied to specific file paths within the application data directory. The data that our anonymous content provider should protect is the ACLs managed by the IdentiDroid Profile Manager. We thus declare only *read* permissions to our data directory containing the ACLs, therefore preventing malicious applications from overwriting the ACL data by accessing it via our custom content provider.

8 Discussion

In our data shadowing solution, the randomized data that we return to the application may affect the usefulness of some application features even when the application is still fully functional. For example, randomizing location data might cause weather applications to return the forecast of a different city not relevant to the user. To address such issue, an approach could be for users to manually enter some input values that can result in the applications still returning some useful, perhaps less accurate, results. For example, in the case of the weather applications, the user could enter a logical "broad" location (such as a city or a county) instead of relaying on the location *auto detect* feature which returns a GPS location. Such an approach would make it possible for the applications to still return useful results to the users while at the same time not using privacy-sensitive information.

Another aspect of concern is the ability of the Internet service providers (ISPs) to identify their own smartphone users. It is important to point out that this a problem affecting current anonymous networks and VPNs as the ISP servers can still identify users through their IP or MAC addresses due to the fact that the Internet traffic originating at smartphones has to pass through the ISP servers. Thus as IdentiDroid works at application level, on top of anonymous networks or VPNs, addressing the problem of protecting the IP or MAC address from the ISPs is outside the scope of IdentiDroid. We emphasize that the goal of IdentiDroid is to assure user anonymity at application level. However it can be used in conjunction with tools assuring anonymity at the network level to provide comprehensive privacy.

9 Related Work

As we are aware that anonymity on Android devices is a known problem, we have not seen any complete work that solves this problem and prevents all possible identifying information from reaching device applications. Moreover, the solutions provided by anonymity network providers (such as Tor Orbot) are not sufficient where applications are still able to identify device and/or user. The closest works to achieve better anonymity on Androids were the privacy related approaches on smart mobile devices, and detection of over-privileged applications and malicious applications.

Past work on privacy for smart mobile devices has mainly focused on the applications that are over-privileged. Vidas *et al.* implemented the Permission Check Tool [22] to assist developers in utilizing the least privilege principle, so that applications would not be over-privileged. Enck *et al.* [23] categorized over-privileged applications based on their permission requirements before installation, and their potential threats in case such permissions are granted and abused. Their experiments identified malware signatures in applications upon installation, and tested applications against a set of security rules describing their potential threat. However, they did not examine the functionality of these applications or whether or not they are abusing their permissions. Most importantly, even with applications that are not over-privileged, user and device re-identification may still occur. Our solutions were built on the premise that even applications that are not categorized as over-privileged may still leak identifying information. Enck *et al.* also built the Kirin [23] that uses a formalized model of policy mechanisms to determine whether or not the privileges requested by applications match the desired functionality. If the matching fails, the application will not be allowed to install. However our solution allows all applications to be installed, but grant the user the ability to manage the resources the application is allowed to access.

In addition to risks arising from permissions granted to applications, a few researchers have tested for information leakage of applications in the Android market. Gibler *et al.* audited applications by using AndroidLeaks [9], a static analysis framework that finds data leaks in applications, and

reported over $2,000$ applications that leak sensitive information containing device information, sensor data, and users' information. However, their static analysis lacks in integration with the Android specific control flow, is thus not able to detect all possible data leaks. Felt *et al.* built Stowaway [8], a dynamic tool for determining the required permissions invoked by each API method. Mann and Starostin have described what sources and sinks of private data could Android APIs leak, and provided a framework for detecting privacy-violating information flows inside Android applications [24]. Although most of such work has been motivated by the privacy risks of Android applications, none of the previous approaches has provided any solution to limit information leakage.

There are few works that closely relate to our work but are still vulnerable and not sufficient for anonymity purposes. The first is by Hornyack *et al.* who developed AppFence [6] to enforce system-wide privacy controls on Android applications. Our work complements Appfence in protecting the user privacy but is more aimed towards protecting the identity of the user/device. Moreover, since AppFence was built on TaintDroid [7], it does not address all implicit flows and therefore cannot deal with malicious applications attempting to circumvent tracking.

Finally, AppFence does not prevent applications from accessing user traces left at the application data storage, which is the key goal of our *Fresh Start* feature. Similar work was done by Schreckling *et al.* who extend the TaintDroid framework to offer fine-grained security policies by supporting the tracking of access restrictions on single data items [25]. Zhou *et al.* developed TISSA [26] to introduce a privacy-mode implementation to the Android OS in order to limit the accessibility of user information to untrusted device applications. Other closely related work is by Nauman *et al.* [27] who modified the permission enforcement of the Android operating system by offering granularity in control permissions. Another similar work is MockDroid [28] that intercepts the permission requests for resources. However, these solutions are not sufficient for applications that can store identifying information within the data space, which is the main purpose of our *Fresh Start* feature.

Our modifications to the Android system were applied at the application level of the operating system, however several researchers have made lower level modifications that are targeted at the kernel and the Android run-time levels for the purpose of giving users security and privacy controls over their devices. Russello *et al.* created security profiles that are dynamically switched by their MOSES Hypervisor to enforce software isolation of applications and data on the Android platform [29]. Xu *at al.* [30] created security and privacy policies that are enforced after repackaging applications with sandboxing code. Finally, Tiwari *et al.* provided users with Bubbles [31], an almost virtualization abstraction, where only data contained in a Bubble can be shared with applications.

Other researchers have implemented detection mechanisms for malicious applications [32–34], some of which are responsible for leaking identifying information. Chen *et al.* investigated the differences between malicious and benign applications and characterized them based on the temporal order of the used APIs and permissions [35]. They built the Pegasus tool that detects applications performing sensitive operations without the user's consent. Moreover, Zhou and Jiang identified two vulnerabilities in Androids resulting from applications with unprotected content provider components [36]. We benefit from such work in understanding how these applications leak phone identifiers and user data.

In all the aforementioned related work, we have not seen any work that offers complete solution for the anonymity problem on Android systems, while IdentiDroid offers two solutions and two major features (*Fresh Start* and *Intent filtering*) that strongly compliment existing anonymity networks in protecting the identity of their users.

10 Conclusion and Future Work

Anonymous networks on smartphone devices such as Android are not sufficient to guarantee the anonymity of users due to several identifying information that may be accessible to applications. In this paper, we introduced a customized Android operating system, called IdentiDroid, that deploys two solutions designed to protect users from being re-identified. Our first solution is based on shadowing sensitive data and the second solution is based on blocking at runtime permissions that lead access to identifying data. We also developed IdentiDroidProfile Manager as a profile-based tool that hosts the customized configuration of our solutions for each installed application. Our experimental results show the effectiveness of these solutions on the sensitive data returned to applications and that it can no longer identify the user or device.

We plan to extend our system to investigate its effectiveness against applications that use native libraries, as they might be able to circumvent both shadowing and permission checking. We also plan to introduce guidelines on how to build applications that can function anonymously. Finally, we plan to study vulnerabilities in applications that have unprotected APIs where malicious applications may use for user/device identification.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *In Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 303–320.

[2] I. AnchorFree, "Hotspot shield," http://www.anchorfree.com.

[3] T. T. Project, "Want tor to really work?" https://www.torproject.org/download/download-easy.html.en.

[4] S. T. Amir Efrati and D. Searcey. (2011, Apr.) Mobile-app makers face u.s. privacy investigation. [Online]. Available: http://online.wsj.com/article/SB10001424052748703806304576242923804770968.html

[5] J. Lin, N. Sadeh, S. Amini, J. Lindqvist, J. I. Hong, and J. Zhang, "Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ser. UbiComp '12, NY, USA. [Online]. Available: http://doi.acm.org/10.1145/2370216.2370290

[6] 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*, ser. CCS '11, NY, USA, 2011.

[7] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10, Berkeley, CA, USA, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[8] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *Proceedings of the 2nd USENIX conference on Web application development*, ser. WebApps'11, Berkeley, CA, USA, 2011, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002168.2002175

[9] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in *Proceedings of the 5th international conference on Trust and Trustworthy Computing*, ser. TRUST'12, Berlin, Heidelberg, pp. 291–307. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30921-2_17

[10] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *Security Privacy, IEEE*, vol. 7, no. 1, pp. 50–57, 2009.

[11] A. D. Guide. Android permissions. [Online]. Available: http://developer.android.com/guide/topics/security/permissions.html

[12] M. Shehab, G. Cheek, H. Touati, A. Squicciarini, and P.-C. Cheng, "User centric policy management in online social networks," in *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, 2010, pp. 9–13.

[13] R. W. Reeder, L. Bauer, L. F. Cranor, M. K. Reiter, and K. Vaniea, "More than skin deep: measuring effects of the underlying model on access-control system usability," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2065–2074. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979243

[14] L. Cranor and S. Garfinkel, *Security and Usability*. O'Reilly Media, Inc., 2005.

[15] K. Fisler and S. Krishnamurthi, "A model of triangulating environments for policy authoring," in *Proceedings of the 15th ACM symposium on Access control models and technologies*, ser. SACMAT '10. New York, NY, USA: ACM, 2010, pp. 3–12. [Online]. Available: http://doi.acm.org/10.1145/1809842.1809847

[16] A. Hundepool, J. Domingo-Ferrer, L. Franconi, S. Giessing, E. S. Nordholt, K. Spicer, and P.-P. de Wolf, *Statistical Disclosure Control (Wiley Series in Survey Methodology)*. Wiley, 2012.

[17] A. O. S. Project. Android security overview. [Online]. Available: http://source.android.com/tech/security/

[18] A. D. Guide. Android apis. [Online]. Available: http://developer.android.com/reference/packages.html

[19] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid android: versatile protection for smartphones," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 347–356. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920313

[20] H. Hao, V. Singh, and W. Du, "On the effectiveness of api-level access control using bytecode rewriting in android," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ser. ASIA CCS '13. New York, NY, USA: ACM, 2013, pp. 25–36. [Online]. Available: http://doi.acm.org/10.1145/2484313.2484317

[21] A. D. Guide. Android interface to global information. [Online]. Available: http://developer.android.com/reference/android/content/Context.html

[22] T. Vidas, N. Christin, and L. Cranor, "Curbing Android permission creep," in *Proceedings of the Web 2.0 Security and Privacy 2011 workshop (W2SP 2011)*, Oakland, CA, May 2011.

[23] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 235–245. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653691

[24] C. Mann and A. Starostin, "A framework for static detection of privacy leaks in android applications," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12, New York, NY, USA, 2012, pp. 1457–1462.

[25] D. Schreckling, J. Posegga, J. Köstler, and M. Schaff, "Kynoid: real-time enforcement of fine-grained, user-defined, and data-centric security policies for android," in *Proceedings of the 6th IFIP WG 11.2 international conference on Information Security Theory and Practice*, ser. WISTP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 208–223. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30955-7_18

[26] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *Proceedings of the 4th international conference on Trust and trustworthy computing*, ser. TRUST'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 93–107.

[27] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10. New York, NY, USA: ACM, 2010, pp. 328–332. [Online]. Available: http://doi.acm.org/10.1145/1755688.1755732

[28] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "Mockdroid: trading privacy for application functionality on smartphones," in *Proceedings of the 12th Workshop on Mobile Computing Systems and*

Applications, ser. HotMobile '11. New York, NY, USA: ACM, 2011, pp. 49–54. [Online]. Available: http://doi.acm.org/10.1145/2184489.2184500

[29] G. Russello, M. Conti, B. Crispo, and E. Fernandes, "Moses: supporting operation modes on smartphones," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, ser. SACMAT '12. New York, NY, USA: ACM, 2012, pp. 3–12. [Online]. Available: http://doi.acm.org/10.1145/2295136.2295140

[30] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: practical policy enforcement for android applications," in *Proceedings of the 21st USENIX conference on Security symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 27–27. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362820

[31] M. Tiwari, P. Mohan, A. Osheroff, H. Alkaff, E. Shi, E. Love, D. Song, and K. Asanović, "Context-centric security," in *Proceedings of the 7th USENIX conference on Hot Topics in Security*, ser. HotSec'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2372387.2372396

[32] 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*, ser. SPSM '11, NY, USA, 2011, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2046614.2046618

[33] W. Enck, "Defending users against smartphone apps: techniques and future directions," in *Proceedings of the 7th international conference on Information Systems Security*, ser. ICISS'11. Springer-Verlag, 2011, pp. 49–70.

[34] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proceedings of the 20th USENIX conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028088

[35] K. Chen, N. Johnson, V. D'Silva, S. Dai, T. Magrino, K. Macnamarra, E. Wu, M. Rinard, and D. Song, "Contextual policy enforcement in android programs with permission event graphs," in *Proc. of the Conference on Networked and Distributed System Security*.

[36] X. J. Yajin Zhou, "Detecting passive content leaks and pollution in android applications," in *Proceedings of the 20th Network and Distributed System Security Symposium*, ser. NDSS'13. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/pubs/NDSS13_CONTENTSCOPE.pdf