Scientific Research Publishing

Communications and Network, 2017, 9, 219-234
http://www.scirp.org/journal/cn
ISSN Online: 1947-3826
ISSN Print: 1949-2421

Risks behind Device Information Permissions in Android OS

Ali Alshehri[1,2], Anthony Hewins[1], Maria McCulley[3], Hani Alshahrani[1], Huirong Fu[1], Ye Zhu[4]

[1]Oakland University, Rochester, Michigan, USA
[2]University of Tabuk, Tabuk, KSA
[3]University of Maryland College Park, College Park, Maryland, USA
[4]Cleveland State University, Cleveland, Ohio, USA

Email: aaalshehri@oakland.edu, aahewins@oakland.edu, mmccull2@terpmail.umd.edu, hmalshahrani@oakland.edu, fu@oakland.edu, y.zhu61@csuohio.edu

How to cite this paper: Alshehri, A., Hewins, A., McCulley, M., Alshahrani, H., Fu, H.R. and Zhu, Y. (2017) Risks behind Device Information Permissions in Android OS. *Communications and Network*, **9**, 219-234.
https://doi.org/10.4236/cn.2017.94016

Received: September 14, 2017
Accepted: November 7, 2017
Published: November 10, 2017

Abstract

In the age of smartphones, people do most of their daily work using their smartphones due to significant improvement in smartphone technology. When comparing different platforms such as Windows, iOS, Android, and Blackberry, Android has captured the highest percentage of total market share [1]. Due to this tremendous growth, cybercriminals are encouraged to penetrate various mobile marketplaces with malicious applications. Most of these applications require device information permissions aiming to collect sensitive data without user's consent. This paper investigates each element of system information permissions and illustrates how cybercriminals can harm users' privacy. It presents some attack scenarios using READ_PHONE_STATE permission and the risks behind it. In addition, this paper refers to possible attacks that can be performed when additional permissions are combined with READ_PHONE_STATE permission. It also discusses a proposed solution to defeat these types of attacks.

Keywords

Android, Security, Privacy, Device Identifiers, Permissions

1. Introduction

Android has been increasing in popularity throughout the recent years, gaining a large majority of the smartphone market. Some of the factors that made it popular include its availability, lower cost, and open source platform. In addition, Google's hands-off approach with developers has facilitated a great deal of free-

dom in what applications can potentially do, allowing numerous categories of apps to grow within Google's main app market, Google Play.

The cost of this freedom is that malicious developers have more vulnerabilities to exploit. To mitigate attacks, Android screens all apps in the app store and uses a permission-based protection to limit and allow access to device features [2]. While this method of protection has been effective against viruses that directly break the rules of the Android system, it is unable to prevent Android malware that takes advantage of permissions.

Permissions restrict application access to user information (e.g. location) or sensitive system methods (e.g. run at startup). Applications that wish to bypass these restrictions need to declare what they wish to bypass in an XML file called the AndroidManifest.xml. Permissions in Android can be classified as either dangerous or normal. Dangerous permissions are considered to be those that could pose a security threat to the user, while normal permissions are generally benign on their own. Of the long list of permissions Android allows, 24 are categorized as dangerous [3]. Before Android's API 23, all required permissions for the app had to be accepted at install time or the app would not be downloaded. With Marshmallow, this is no longer true, as this version allows users to block permissions at runtime.

Many applications use permissions to take advantage of device identifiers. Device identifiers are strings that can directly identify the phone subscriber or identify their device. Three of the main device identifiers are the International Mobile Equipment Identity (IMEI), International Mobile Subscriber Identity (IMSI) and phone number. IMEI identifies the hardware of the device, *i.e.* the phone itself, so when a subscriber obtains a new phone, they also obtain a new IMEI number corresponding to that phone. MEID (Mobile Equipment Identifier) and ESN (Electronic Serial Number) numbers serve the same purpose as IMEI. IMSI, on the other hand, identifies the subscriber directly and is found in the removable SIM card. When a user obtains a new phone, the SIM card is removed from their old phone and placed in their new one, transferring their subscription to their carrier and their IMSI number.

Related works have shown that there is danger involved with device identifiers [4] [5] [6] [7], but little work has fully discussed specifically how this information can be used maliciously. Therefore, in this paper we discuss the significance of the READ_PHONE-STATE permission in Android devices. We analyze some of the known malware that leak device identifiers. We provide some examples of attack vectors using IMEI, IMSI, and phone number and describe them in details. Then, we propose a solution that can be used to evade these types of attacks.

The rest of this paper is organized as follows: Section 2 provides a brief introduction to Android. In Section 3, we introduce two permissions that are vital to misuse of device identifiers. After that, attack models are presented in Section 4. We present our solutions to protect device identifiers in Section 5. Lastly, we

compare our work with related works in Section 6 and then the conclusion of the paper and the future work is included in Section 7.

2. Android Overview

In order to fully explain how device identifiers are at risk. We will illustrate Android's security features. We will then explain some of the flaws and vulnerabilities in the way Android approaches applications, developers, and advertisers. These vulnerabilities will be important to keep in mind when we present out attacker vectors.

2.1. Android Security Features

2.1.1. Sandbox

Android applications are run in a sandboxed environment by assigning each application a Linux User Identification number (UID) and a group ID (GID). For the operating system, this creates the appearance that applications are actually separate people using the device [8]. The UID number is specific to the developer of the app to prevent impersonation of another developer in the Google Play store. Applications from the same developer have the option to combine permissions into a single user if they request to do so [9] [10].

2.1.2. IPC

Although apps are sandboxed, applications can still communicate with other apps and the Android system if the correct permissions are in place. Both apps need to consent otherwise the requesting app cannot communicate with other apps. This is called Inter-Process Communication (IPC) [10]. Having apps communicate with the same UID can be a security risk that allows privilege escalation, as will be discussed later.

2.1.3. Broadcasts

Besides IPC, apps can be allowed to both send and receive messages to all applications on the device. This is called broadcasting and broadcast receiving, respectively [11]. Broadcasts can be messages such as "5% Battery", which displays for a brief moment. Broadcast receiving is the process of receiving these broadcasts in order to perform some action. For example, an app may want to know when the "5% battery" broadcast is received in order to optimize battery life.

2.1.4. Intents

In Android, events are driven by Intents. Intents are objects that communicate the desire to perform an action [9], such as open up the user's contacts. Intents can be packaged with extra data, such as opening up a specific contact inside the user's contacts. Intents can be Explicit or Implicit. Explicit Intents signal exactly what app will execute the Intent, while Implicit Intents will ask all able applications on the device to perform the desired action. Only apps that can handle the request will be notified [10]. For example, when a user has two Internet browsers

and wants to open a link, both browsers could be used, so the system will ask the user to choose which one he/she prefers.

2.1.5. Permissions

Permissions, as explained in the introduction, allow and deny developers access to sensitive user data or phone functions. If an application requests to use dangerous permissions, the user must accept the request before the app can gain access. This security measure allows users to make the important security decisions.

2.2. Android Vulnerabilities

2.2.1. Unprotected Broadcasts

Implicit Intent broadcasts can lead to data leakage. Implicit Intent broadcasts are for all eligible applications to hear, and in the broadcast can be sensitive data that is not protected. Malicious programs that have the permission required to listen will eavesdrop and receive information bundled with the Intent object. In addition to eavesdropping, malicious programs have been known to intervene into Intent conversation by injecting Intent objects that siphon information from broadcast receivers that are not well protected [5].

2.2.2. Collusion

Collusion enables a developer to add permissions together [11]. Apps signed with the same UID can combine permissions, allowing any one of the apps under the UID to access all the permissions signed with the UID [12]. For example, one app using device identifiers and another using internet under the same UID can combine their permissions, making it so that app can use both internet and device identifiers. A developer could then send these personal identifiers over the internet to do harm to a user.

2.2.3. AD Libraries

Advertisement libraries get packaged with an app and receive the same permissions as that app. Malicious libraries can then abuse the permissions they are given on behalf of the application and use any number of the malicious attack strategies discussed further on [13].

3. User Privacy and Android Permissions

Permissions in Android give developers access to user information such as device identifiers. In the following sections, we will introduce the READ_PHONE_STATE permission, which allows access to device identifiers, and READ/RECEIVE SMS, which we will use in our explanation of our attack vectors.

3.1. Device ID Information

READ_PHONE_STATE is one of the Android permissions categorized as dangerous. This is because it "allows read only access to phone state, including the

phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any Phone Accounts registered on the device" [2]. This is an important permission for messaging, calling, and other applications that replace the built-in phone application; however, it is often requested and misused by other types of applications that do not have any reason for needing it, including malware. In fact, **Figure 1** shows READ_PHONE_STATE is one of the most common permissions requested by malicious apps, with 1179 out of 1260 malicious apps requesting it and only 34 percent of benign apps requesting it [6].

This permission is desirable to malware because it allows access to device and user identifiers that can be utilized to harm the user.

READ_PHONE_STATE (Device ID information) enables the developer to get access to the Telephony Manager class, a class built into Android that has several methods that reveal device information. Some of the important information it can reveal includes the user's phone number and Device ID. **Table 1** lists the important methods [14] discussed in this paper.

To further prove malicious applications collect and leak device identifiers, we analyze a set of malicious applications [15]. Through our analysis, we found a malware application looks as normal kitchen timer application, however, it transmits a user's sensitive data such as device ID and phone number to an outside URL without the user consent as illustrated in **Figure 2**. Out of the many applications we observed, this example showed the clearest misuse of device identifiers. The example clearly shows the device identifiers, IMEI (Device ID) and phone number (Line 1 Number), collected and then transmitted to the URL of the malicious site. This example clearly shows that it is easy for an application to transmit the user's sensitive data without the user awareness. The user thinks

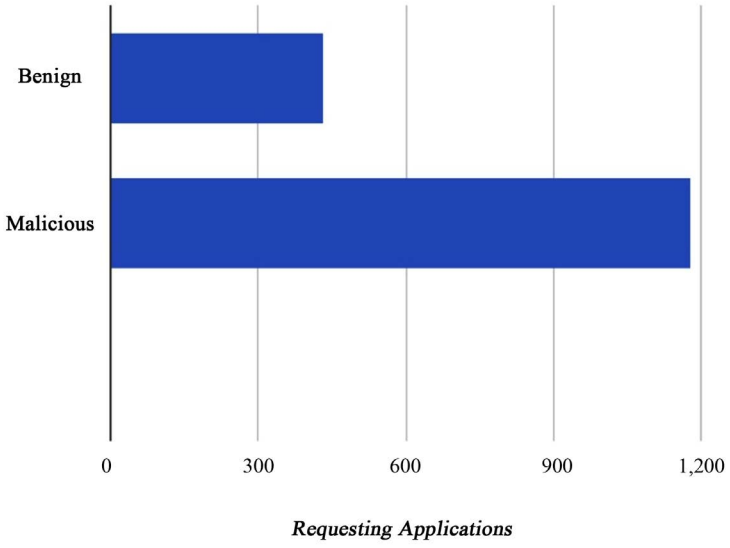

Figure 1. Comparison of READ_PHONE_STATE requests of Benign and Malicious Applications.

Table 1. Relevant telephony manager methods.

Method	Function
Get device ID	Returns the unique device ID, for example, the IMEI for GSM and the MEID or ESN for CDMA phones. Return null if device ID is not available.
Get line 1 number	Returns the phone number string for line 1, for example, the MSISDN for a GSM phone. Return null if it is unavailable.
Get SIM serial number	Returns the serial number of the SIM, if applicable.
Get network operator name	Returns the alphabetic name of current registered operator.

it is a kitchen timer application, however, the application is actually stealing the user personal information.

3.2. SMS Privileges

In general, SMS privileges can include READ, RECEIVE, WRITE or SEND SMS. In this paper, we examine READ and RECEIVE SMS. READ SMS allows read access to the SMS inbox through "Uri. parse ("content://sms/inbox")". RECEIVE SMS allows an application to be notified when the SMS RECEIVED ACTION broadcast is sent, and it allows the app to read the incoming SMS as well. This is done by using the SMS Message class and calling create from PDU, which builds the message so it can be used. After the message is built, get Message Display Body can be used to determine the contents of the message and get Originating Address can be used see the sender. These methods [16] are listed in **Table 2**.

4. Attack Models for Device ID

In this section, we will explain how device identifiers can be used maliciously by presenting attack scenarios. We will discuss attacks that use only READ_PHONE_STATE and attacks that use both READ_PHONE_STATE and READ, RECEIVE SMS. We will also focus on how individual device identifiers can be used maliciously.

4.1. Device ID Information (READ_PHONE_STATE)

With this permission enabled, an app from the Google Play store "can access your device ID(s), phone number, whether you're on the phone and the number connected by a call. Device ID & call information may include the ability to read phone status and identity" [17]. The information contained in this permission, including the device's IMEI/MEID and phone number, is extremely compromising, although it is rarely treated as such. Identifiers are often streamed, even by benign apps, in plaintext requests that leak the information to malicious apps listening in [5]. In the following sections, we will explain why these device identifiers are compromising and what malicious persons can do with this information.

Table 2. Relevant SMS message methods.

Method	Function
Get originating address	Returns the originating address (sender) of this SMS message in String form or null if unavailable
Create from PDU	Create a SMS Message from a raw PDU with the specified message format. The message format is passed in the SMS_RECEIVED_ACTION as the format String extra, and will be either "3gpp" for GSM/UMTS/LTE messages in 3GPP format or "3gpp2" for CDMA/LTE messages in 3GPP2 format.
Get display message body	Returns the message body, or email message body if this message was from an email gateway. Returns null if message body unavailable.

4.1.1. IMEI/MEID/ESN Attacks

1) Gather Information

If an attacker gains the IMEI/MEID/ESN number of a user, the attacker can find sensitive information about the device. The IMEI number is formulated with three fields: the Type Allocation Code (TAC), Serial Number (SNR), and a check digit (CD) that can be reversed to find system information. For example, the TAC gives the attacker information about the phone's make and model, which can provide even more information on the OS, CPU, and other features of the phone that may assist the attacker in preparing targeted exploits for a particular cellular device [18]. Similar constructions can be done with MEID and ESN. One application, packaged as *com.avantar.wny*, was discovered to have bundled IMEI, device model, platform, and application name into a URL parameter string [5]. While no clear purpose was found for this method, its existence suggests that the developer had a plan in mind for this information.

Targeted attacks have been on the rise, increasing from 9.5 percent of all cyber-attacks in 2014 to 10.5 percent in 2015 to 12 percent for the month of April in 2016 [19] [20]. A specific example is the Heartbleed attack, which functions by exploiting a vulnerability in the OpenSSL cryptographic software library that is used on many devices and servers. Attackers could use this to read the memory of systems running the affected OpenSSL and to obtain access to names, passwords, and other data [21]. With knowledge of the IMEI, an attacker can discover if a phone is vulnerable to the Heartbleed exploit or another exploit and base their attack of this information.

2) Trade User Information for Money

Data brokers, who collect and analyze data, have created a multi-billion-dollar industry from the data generated from apps and other online activities [22]. These companies make their fortune by famining demographic, behavioral and transactional data, to create the fullest possible picture of the people most likely to buy from their clients [23]. For example, many people use health-related apps to record menstrual cycle patterns, glucose levels, and other information about their personal health. These applications sell this information to data broker companies such as Epsilon, Acxiom, and Experian, who then compile this

```
Public void onReceive(Context paramContext, Intent
paramIntent)
    {
        Main.this.kitchenTimerService.schedule(300000L)
      if (Main.this.ctf.intValue() == 0)
      {Main.this.ctf = Integer.valueOf(1);
        paramIntent =
(TelephonyManager)Main.this.getSystemService("phone")
        Object localObject =
AccountManager.get(Main.this).getAccounts();
        paramContext = "";
        int j = localObject.length;
        int I = 0;
        for (;;)
        {
                if (I >= j)
          {
                localObject
        =Main.this.doPost("http://ibbeancom.com/entry.p
        hp?id=4", "");
            paramContext = new
Intent("android.intent.action.VIEW",
Uri.parse("http://ibbeancom.com/send.php?a id=" +
paramIntent.getDeviceId() + "&telno=" +
paramIntent.getLine1Number() + "&m addr=" +
paramContext + "&usr_id=" + (String)localObject));
            Main.this.startActivity(paramContext);
            Main.this.moveTaskToBack(true);
```

Figure 2. Malicious kitchen timer application.

information and sell it to interested companies. Insurance companies are especially interested in this data because it helps them to identify clients who would be bad investments due to their health conditions [24].

One of the ways these data brokers obtain this information is by purchasing it from applications and the READ_PHONE_STATE permission allows these brokers to easily connect this information with a person. With applications like a glucose tracker, users put in a lot of private information but they may not put in their name or any other data that would personally identify them so that this data cannot be tracked back to them. However, READ_PHONE_STATE allows access to the IMEI which can be used as a universal identifier for any data broker. Some of this data may include your name or email address which data brokers obtained through social media applications or other sources. Users may believe that the information they are providing is inconsequential because it cannot be connected back to them, however, with the READ_PHONE_STATE permission it easily can. Enck *et al.* found several examples of applications that purposely linked IMEI to personally identifiable information (PII), including a game that linked IMEI to the user's high scores [5]. Any person identifier can be used to connect information to the user; however, IMEI appears to be the most popular.

4.1.2. Phone Number

1) Obtain User's Name and Profile Picture

Using a user's phone number, an attacker can retrieve the user's name and profile picture through Facebook as an example. This approach assumes the user has a Facebook and the user's Facebook is tied with their phone number. To accomplish this, the attacker goes to password recovery, enters the phone number, and Facebook will display the name and profile picture the user has on their account. This is compromising enough to create a domino effect of more attacks.

2) Phishing

Attackers can also use a phone number to perform social engineering attacks such as phishing by directly calling or texting the user. When coupled with the attack above, this attack has the potential to be far more effective, allowing the attacker to possibly pose as someone the user knows. Phishing alone results in the loss of millions of dollars annually [25] and is an extremely easy way to steal money from a user. Mobile devices are especially vulnerable to phishing through websites because web designers should simplify their sites to work with limited resources, making it easier to replicate. Additionally, URLs are often harder to read on mobile devices making it easier for users to make a mistake on a malicious site. Studies have shown mobile users are three times more likely to submit their credentials to these phishing sites than desktop users [26].

3) Determine a General Location

The typical attacker can get a general location of where the user may live through the area code in combination with the Central Office. Factors such as availability of a Central Office, which corresponds with the user's desired carrier, and the stability of the user's home location will impact accuracy. In North America, phone numbers are broken down into three parts: the area code, the central office code, and a unique four-digit code to identify the specific number. The span of area codes ranges widely from state to state. For example, New York is home to fifteen different area codes while Wyoming only has one [27]. The Central Office code links a phone number to a specific office and carrier and when combined with the area code can be used to determine a possible town or zip code that the user may live in.

4.1.3. IMSI

Another tool that can be used in conjunction with the READ_PHONE_STATE permission to carry out an attack is an IMSI catcher. IMSI catchers are devices used mainly by government agencies for phone tapping, intercepting data, and tracking of mobile devices. Basic IMSI catchers can listen in on calls and precisely determine location. Dirtbox, a common IMSI catcher used on planes, can locate the desired cell phone to a ten-foot radius [28]. Another commonly used tool by government agencies, VME, allows for "voice manipulation, up or down channel blocking, text interception and modification, calling and sending text on behalf of the user, and directional finding" [29]. Other catchers are also known to be able to shut down devices remotely and track devices even when owners believe that they have been turned off [28] [30].

In the context of android applications, information obtained through the READ_PHONE_STATE permission can drastically increase the success of these attacks by providing an IMSI that is connected to a user. Knowing a particular person's IMSI and a general location, an attacker can set up an IMSI catcher in the region and pinpoint a more exact location as well as collect data coming out of the desired phone if the catcher is advanced enough. For example, if their app has become popular enough to obtain a celebrity following, attackers can utilize

basic versions of this tool to find celebrities' exact location and sell this information to paparazzi. Generally, this tool can be used to blackmail, stalk, or perform other targeted attacks on a specific person based on IMSI.

4.2. SMS Privileges ({READ, RECEIVE} SMS)

4.2.1. Account Hijacking

READ_PHONE_STATE and READ, RECEIVE SMS can be used together to hijack accounts. Many services allow users to reset their password by sending an SMS recovery message with a randomly generated key to the user's phone. If the attacker has the phone number of the user and can read incoming text messages, they can reset the password for the account without knowing any of the credentials and gain access. Once the attacker has access, they can also change recovery information, such as backup emails, so the original user cannot get back in. Attacks on these accounts, barring any outside intervention from the provider of the service, are often impossible to recover.

In the following subsections, we list some popular services that are vulnerable and explain any measures or countermeasures that make this service more dangerous or safe than others. We present only a few examples, however, any account that can have its password reset via text is vulnerable to this type of account hijacking.

a) Facebook: Upon login, the attacker has the option to log out every other device that's currently logged in to Facebook, booting the real user out of the account and eliminating any chance for the user to recover the password. Facebook does offer the option to supply state identification [31] to prove a user is the real owner of the account, however, this can be a lengthy process and in which time the attack has most likely carried out whatever attack that they had planned, whether that be phishing, reputation destruction, etc.

b) Yahoo! Mail: Once the attacker logs in, the attacker can replace the phone number and backup email associated with the account, eliminating the possibility of resetting the password.

c) Twitter: With Twitter, a user can retrieve the account, but the user needs to respond quickly. Twitter's security has a backup code that allows the user to log in regardless of password changes that the user can decide to generate at any time during the account's lifetime. If the user decides to use this feature, the user can regain control of the Twitter account.

d) T-Mobile: T-Mobile requires the billing zip code in addition to a verification key. However, using the methods above, this attack can be accomplished using brute force by guessing the zip code based on general location.

These types of account hijacking attacks have been on the rise over the last year. Account hijacking rose from on average 8.8 percent in 2015 to 14.5 percent in April 2016 [19] [20]. Additionally, out of all attacks that were categorized as Cybercrime, 19 percent of those were account hijacking attacks in 2015. Other categories included hacktivism, cyber espionage, and cyber warfare [7]. Once an attacker controls a user's account, there are several ways to profit of it. The most

common method is using the compromised account for spam or a product the attacker is trying to sell. In an online survey done by Shay *et al.*, out of 294 subjects 89 experienced a compromise of their account. Out of these 89, 37.1 percent experienced accounts selling spam to contacts. This was the most prominent way subjects experienced harm from the compromise [25].

4.2.2. Proof-of-Concept (SMS Account Hijack Attack)

Using READ_PHONE_STATE, RECEIVE SMS and INTERNET permissions, we constructed a proof-of-concept application for Android that mimics the attack above. This can still be possible with READ SMS, but the effectiveness of the attacker is greatly reduced. Upon opening the application, the user's phone number is extracted and stored as a string variable named "phone" as shown in Figure 3. The application also extracts whether or not the user's SIM operator number corresponds to T-Mobile and stores it as a Boolean called "is TMobile". An attacker will want to know if the user has T-Mobile since T-Mobile accounts are vulnerable to the attack. Next, the app connects to a server with the is TMobile and phone variables attached as $ GET parameters to the URL. A PHP script then takes the information and stores it into a .txt file called log.txt, shown in Listing 1.

The attacker can then use the phone number to reset the password of a target account, forcing the account service to send a verification text. After the text is sent, Broadcast Receiver listens to the incoming message and saves the text message body, shown in Figure 3. Using the same process as above, the message body is put as a $ GET parameter in the URL and saved to the same text file. The attacker reads the text message and resets the password permanently. The attacker then will delete all recovery emails or phone numbers, gaining full access to the account. From there, they can perform any of the methods listed above to

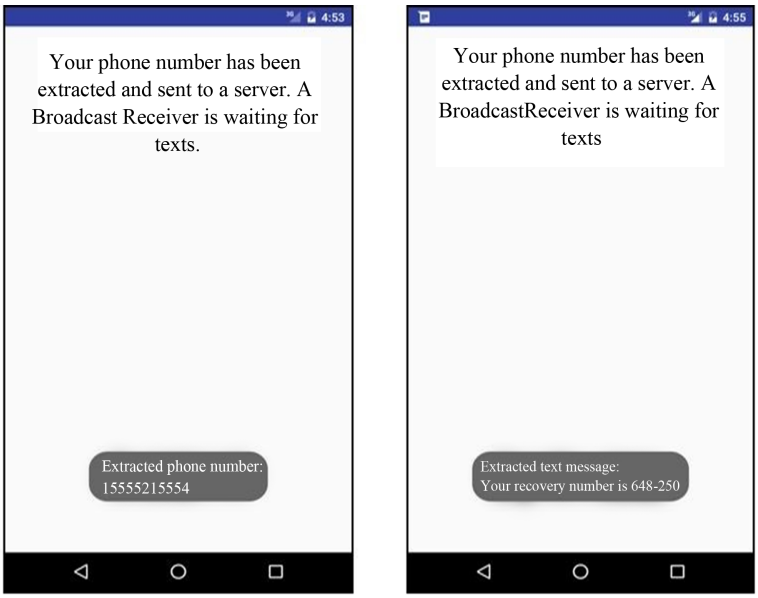

Figure 3. SMS account hijack.

profit off the account. The idea of the constructed application is illustrated in Figure 4.

5. Proposed Solutions

READ_PHONE_STATE is much too broad of a permission. It provides applications the status of ongoing calls, which is helpful for many applications, while also providing access to phone number, IMEI, and IMSI. The only functional reason to use any of this information is to replace default messaging and calling apps. Since a phone can have only one default messaging app while the status of ongoing calls can be used by many applications, we propose that READ_PHONE_STATE be partitioned into two new permissions: PHONE STATUS and PHONE IDENTITY. PHONE STATUS would include basic functions like *hasCarrierPrivileges*() or *getCallState*() so applications know if the device is a phone and if there's an incoming call. PHONE IDENTITY would include *getDeviceId*(), getLine1Number, *getSimSerialNumber*(), and other methods that could potentially compromise the user's privacy. Phone status could then be considered a normal permission, while phone identity would be classified as dangerous, allowing malware to be profiled much easier.

With the recent Android update to Marshmallow, users now have the option to revoke specific permissions from an application during installation. This update, while extremely helpful in other cases, does not solve the problem with READ_PHONE_STATE. Many developers still need access to *getCallState*() and therefore need to request this permission. Users now have the choice to revoke

```
<?php
$fp= fopen("log.txt", 'a'); fwrite($fp, 'phone=' . $GET["phone"]); fwrite($fp,
'isTMobile=' . $GET["isTMobile"]); fclose($fp);
?>
```

Listing 1. PHP script for extracting the data.

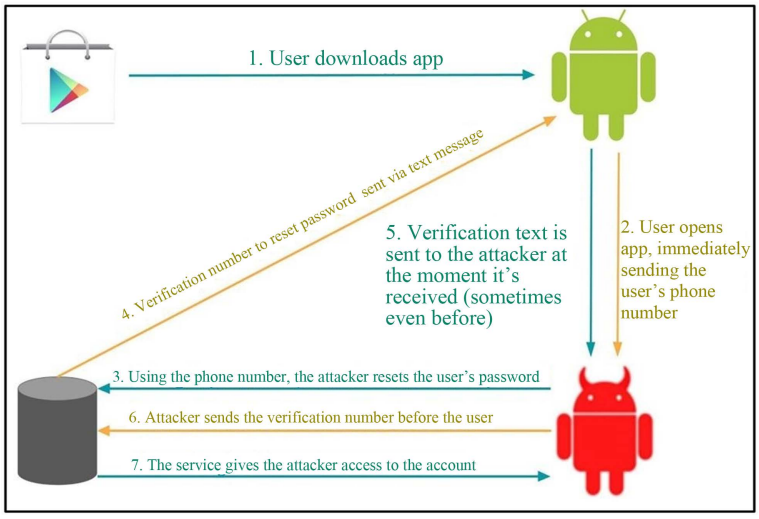

Figure 4. SMS account hijacker.

this permission, but they run the risk of missing a call when using the application. One solution, Identi Droid [32] gives users the option of shadowing their data or, in other words, returning to the requesting application randomized data in place of the IMEI, phone number, or other sensitive data. The problem with Identi Droid is that it requires reconfiguration of the Android source code and for the user to root their device. This is very impractical for the everyday user.

Since both the recent update to Marshmallow and privacy tools such as Identi Droid fail to protect the everyday user's privacy, the most practical solution would be for Android to break down the READ_PHONE_STATE permission into phone status and phone identity. This way developers can get access to only what they need and users do not have to choose between functionality and privacy. This simple solution will reduce the access to these device identifiers drastically and stop developers from performing most of the attacks we have presented in this paper.

6. Related Works

Studying and analyzing Android's permissions has been an interested area of research in the last few years. Numerous researches have been showing that there are dangers inherent in device information permissions and the leakage of device identifiers [4] [5] [6] [7]. Jiang and Zhou [6] demonstrate that malicious applications ask for device information permissions, READ_PHONE_STATE, significantly more than benign applications suggesting that the information obtained through this permission aids these apps in performing malicious activities. Batyuk *et al.* [4] demonstrates through their study that almost 70% of all analyzed applications that had access to device identifiers streamed this private data immediately upon reading it.

Enck *et al.* [5] looked at information misuse of phone identifiers and they found out that 246 out of 1100 applications they analyzed using static analysis had code to obtain a device identifier, with IMEI being the most frequently requested identifier contributing to 61% of all calls. In another work done by Enck *et al.* [33], they found out when they tested Taint Droid, which is a dynamic taint tracking tool, 21 out of 30 applications require permissions to READ_PHONE_STATE and the Internet and 9 of them transmitted the IMEI. Gibler *et al.* [34] introduced Android Leaks as a static analysis tool used to find possible data leaks. They found potential privacy leaks in 7,414 out of 24,350 applications. However, the leakage of IMEI or IMSI compromised over ninety percent of all leaks found by Android Leaks and occurred in 28.39 percent of analyzed applications. Out of all of the user's data, device identifiers are the most frequently leaked.

While these works are able to provide proof that malicious activity is occurring, they fail to offer an explanation of what this activity might be. With the exception of Enck *et al.*, no other work illustrates the potential harm that comes from READ_PHONE_STATE and device identifiers. We build on Enck *et al.* to

provide more malicious scenarios and a complete picture of what attackers can really do with information.

7. Conclusions

Permissions provide malicious applications a way to access sensitive data on mobile devices without breaking the rules of Android. READ_PHONE_STATE, which allows access to device identifiers, is the most commonly misused permission by benign apps and the most requested dangerous permission by malicious apps. As we have illustrated, device identifiers can be used to harm the user through many different attack scenarios. Malicious applications can sell user information. Malicious applications can also gather general information about the user and phone or resort to mobile phishing.

When combined with READ SMS, a phone number can be used to hijack a person's account. It is clear from related works that READ_PHONE_STATE is requested significantly more often by malicious applications than benign ones, device identifiers are the most frequently leaked piece of data, and there is no clear way to know what applications are doing with user's data. We suggest a simple solution: split up READ_PHONE_STATE into phone status and phone identity, where phone status could be downgraded to a normal permission.

Acknowledgements

This work was supported in part by NSF under grants CNS-1460897, DGE-1623713. 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 NSF.

References

[1] Gartner (2017) Worldwide Smartphone Sales to End Users by Operating System. http://www.gartner.com/newsroom/id/3725117

[2] Oberheide, J. and Miller, C. (2012) Dissecting the Android Bouncer.

[3] MegaNet Corporation (2016) VME Undetectable Cell Phone Interceptors. http://www.meganet.com/meganet-products-cellphoneinterceptors.html

[4] AllAreaCodes.com (2016) About All Area Codes. http://www.allareacodes.com/

[5] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L.P., Jung, J., McDaniel, P. and Sheth, A.N. (2014) TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transactions on Computer Systems (TOCS),* **32**, 5. https://doi.org/10.1145/2619091

[6] Google Play Help (2016) Review App Permissions Thru Android 5.9. https://support.google.com/googleplay/answer/6014972?hl=en

[7] Dot, N. (2015) The Importance of Data (Part I).

[8] University of Virginia (2014) UNIX/Linux UID and File Ownership over NFS. http://its.virginia.edu/unixsys/sec/nfs-uids.html

[9] Android Developers (2016) System Permissions. https://developer.android.com/guide/topics/permissions/index.html

[10] Facebook (2016) Confirm Your Identity with an ID. https://m.facebook.com/help/contact/183000765122339

[11] Rashidi, B. and Fung, C. (2015) A Survey of Android Security Threats and Defenses. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications 6.

[12] Batyuk, L., Herpich, M., Camtepe, S.A., Raddatz, K., Schmidt, A.D. and Albayrak, S. (2011) Using Static Analysis for Automatic Assessment and Mitigation of Unwanted and Malicious Activities within Android Applications. 2011 6*th International Conference on Malicious and Unwanted Software (MALWARE),* Fajardo, Puerto Rico, 18-19 October 2011, 66-72. https://doi.org/10.1109/MALWARE.2011.6112328

[13] Sufatrio, Darell J. J. Tan, Tong-Wei Chua, and Vrizlynn L. L. Thing. (2015) Securing Android: A Survey, Taxonomy, and Challenges. *ACM Computing Surveys*, **47**, Article No. 58.

[14] Android Developers (2016) TelephonyManager. https://developer.android.com/reference/android/telephony/TelephonyManager.html

[15] Deep End Research (2016) Malware Dataset. https://www.dropbox.com/sh/fwmhcw37o0u7f6p/AADADt2XkojibPzLBBxaQbbqa?dl=0

[16] Android Developers (2016) SmsManager. https://developer.android.com/reference/android/telephony/SmsManager.html

[17] Heartbleed (2016) The Heartbleed Bug. http://heartbleed.com

[18] Gibler, C., Crussell, J., Erickson, J. and Chen, H. (2012) Android Leaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In: *International Conference on Trust and Trustworthy Computing*, Springer, Berlin, 291-307. https://doi.org/10.1007/978-3-642-30921-2_17

[19] Paolo, P. (2015) April 2015 Cyber Attacks Statistics. http://www.hackmageddon.com/2016/01/11/2015-cyber-attacks-statistics

[20] Paolo, P. (2016) April 2016 Cyber Attacks Statistics. http://www.hackmageddon.com/2016/06/01/april-2016-cyber-attacks-statistics/

[21] Grzonkowski, S., Mosquera, A., Aouad, L. and Morss, D. (2014) Smartphone Security: An Overview of Emerging Threats. *Electronics Magazine*, **3**, 40-44. https://doi.org/10.1109/MCE.2014.2340211 http://www.hackmageddon.com/2015/06/18/the-importance-of-data-part-i

[22] Steve, K. (2016) The Data Brokers: Selling Your Personal Information. https://www.cbsnews.com/news/data-brokers-selling-personal-information-60-minutes/

[23] Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S. (2011) A Study of Android Application Security. 20*th USENIX Security Symposium*, 10-12 August 2011, San Francisco, CA.

[24] Boksasp, T. and Utnes, E. (2012) Android Apps and Permissions: Security and Privacy Risks. Norwegian University of Science and Technology, Trondheim.

[25] Shay, R., Ion, I., Reeder, R.W. and Consolvo, S. (2014) "My Religious Aunt Asked Why I Was Trying to Sell Her Viagra": Experiences with Account Hijacking. In: *Proceedings of the* 32*nd Annual ACM Conference on Human Factors in Computing Systems*, ACM, New York, 2657-2666.

[26] Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M.S., Conti, M. and Rajara-

jan, M. (2015) Android Security: A Survey of Issues, Malware Penetration, and Defenses. *IEEE Communications Surveys & Tutorials*, **17**, 998-1022.

[27] Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M.S., Conti, M. and Rajarajan, M. (2015) Android Security: A Survey of Issues, Malware Penetration, and Defenses. *IEEE Communications Surveys & Tutorials*, **17**, 998-1022. https://doi.org/10.1109/COMST.2014.2386139

[28] Joseph, O. (2015) IMSI Catchers and Mobile Security.

[29] Boodman, E. (2016) Health Apps Aren't Just Collecting Your Info. They May Be Selling It, Too. https://www.statnews.com/2016/03/08/health-apps-sell-medical-data/

[30] Augustine, F. (2016) Mobile Phishing Social Media Phishing and Other Attacks. http://www.slideshare.net/augustinefou/mobile-phishing-social-media-phishing/-and-other-attacks

[31] EPSILON (2016) EPSILON. http://www.epsilon.com/

[32] Shebaro, B., Oluwatimi, O., Midi, D. and Bertino, E. (2014) Identidroid: Android Can Finally Wear Its Anonymous Suit.

[33] Elenkov, N. (2014) Android Security Internals: An In-Depth Guide to Android's Security Architecture. William Pollock, San Francisco, CA.

[34] Ryan, G. (2012) Criminals May Be using Covert Mobile Phone Surveillance Tech for Extortion. http://www.slate.com/blogs/future_tense/2012/08/22/imsi_catchers_criminals_law_enforcement_using_high_tech_portable_devices_to_intercept_communications_.html