Acute Analysis of Over-Privileged Android Apps

Basundhara Dey
College of Engineering and Computer Science
University of Central Florida
dey.basundhara@knights.ucf.edu

Rishi Wadhwani
College of Engineering and Computer Science
University of Central Florida
rishiwadhwani@knights.ucf.edu

***Abstract*— Android is of the most rapidly developing mobile operating system today with new upgrades bringing out huge improvements and massive abilities for mobile phones. The upgrades continually update a running system, supplanting and including incalculable archives over Android's brain boggling development, in the region of fundamental customer data and applications. Smartphone security examination has brought to light various vulnerabilities and short comings present in the security of the android operating system for today's smartphones. Smartphones are depicted by their ability to run untouchable applications, and Android takes this thought to the convincing, offering an enormous number of "apps (applications)" through application markets, i.e. Google Play Store. By virtue of the unlucky deficiency of essential access control approval, affected applications can be abused to either latently reveal a wide variety of private in-application data or in a way control certain security-fragile in-application settings or game plans that may in this way achieve system wide responses. In this paper, we discuss the stream state of Smartphone examination, fabricating a model application to show how the security helplessness happens in foundation for over privileged applications. We offer comprehension into what satisfies desires and promising direction for future investigation.**

Keywords*— *android, security, Smartphone, vulnerabilities, apps, applications, manifest, permission.

I. INTRODUCTION

Smart phones are a generally prevalent and developing space of registering innovation. Cell phones give an ultra-convenient interface to the Internet and the computational capacities to make it significant. Utilizing environment sensors, for example, GPS, cameras etc. they upgrade ordinary assignments with the abundance of data accessible from the Internet. Crucial to cell phones are applications, conversationally known as "Apps." To secure touchy assets in the smart phones, consent based permissions are utilized by cutting edge cell phone frameworks to keep untrusted applications away from the smartphone user. In Android, an application needs to expressly ask for an arrangement of authorizations when it is introduced. On the other hand, after consents are allowed to an application, there is no real way to assess and limit how these authorizations are utilized by the application to use delicate assets. Obviously, Android has pulled in an enormous number of assaults.

While these malware applications are clear samples containing undesirable practices, sadly indeed, even in evidently generous applications, there could likewise be numerous concealed undesirable practices, for example, protection intrusion. A vital part in the battle against these undesirable practices is the examination of delicate practices in Android applications. Customary investigation methods recreate program practices from gathered system executions.

On account of Android gadgets, the establishment process for another application approaches the client for authorization to get to certain ensured assets, for example, camera, contact lookup, or system network. On occasions the rundown of authorizations may appear to be inordinate and disconnected to the application's usefulness. Disregarding this, numerous individuals will concur and gift authorization without mulling over the potential security or protection suggestions. Android engineers need to unequivocally list the consents required for their applications. They have a tendency to incorporate a bigger set than what is really required by the system. This is because of the way that Android OS recognizes about 150 unique permissions. A significant number of these permissions are fundamentally the same to each other and most are archived vaguely, if by any stretch of the imagination. This leads the designers to rundown consents that are pointless, prompting "over-privileged applications". These applications can be misused by malevolent programmers, with potential unfriendly impacts on the client's security and protection. Our exploration addresses the issue of over-privileged applications by distinguishing the pointless permissions that may have been erroneously recorded by the designers.

The main contribution of the paper is developing an app using android studio to show how the over privileged apps creep into the user data and in the background of the running system, hack into the user's private data. Before this, we will look into Android's permission model to get a deeper understanding about how the permission system works and how the attack is possible using loopholes in Android Security. Lastly, we will try and find some possible ways to detect malicious apps which request for higher privileges than needed. In future, we will look forward to implement this ideas to build some tool (app) that will analyze the source code and consequently gather the minimum level of privilege that is needed for the smooth functioning of the application. This methodology is one of a kind in the strategy used to focus authorization necessities and, to support reception, is actualized nearby the current IDE suggested for Android advancement.

II. ANDROID PERMISSION SYSTEM

A. Smartphone Layout

Smartphones downloads applications from application markets and run them inside a center product environment. Existing cell phone stages depend on application markets and

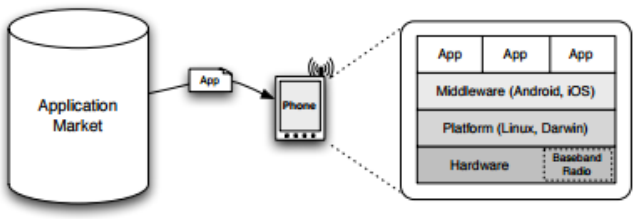

Figure 1. Basic Architecture of Smartphone

stage insurance instruments for security. We now diagram securities right now actualized in well-known stages.

B. *Android Permissions*

Based upon a Linux bit, Android uses working framework primitives, (for example, methods and client IDs) and a Java Virtual Machine (Dalvik) to segregate applications giving a wellbeing sandbox. The Android stage presents around 130 application level authorizations that are asked for at introduce time and quietly upheld whenever the application is executed. Not at all like other protection models, for example, is the iPhone, the client not incited when an application asks for an asset amid execution.

Introduce Install time authorizations fill numerous needs. They give: a) client assent, b) barrier top to bottom, and c) audit triaging. Introduce time consents give barrier top to bottom by characterizing a greatest benefit level, obliging an assault on an application to moreover endeavor a stage defenselessness to perform undertakings outside of the application's extension. Studies have likewise observed that applications don't just demand each authorization, making them profitable traits. A time-of-use authorization is affirmed by the client when the application executes a delicate operation, e.g., iOS's brief to permit an application access to area. An introduce time authorization is sanction by the client when the application is introduced. For Android, this is the client's just chance to deny access; the client must acknowledge all authorization demands or not introduce the application.

Android consents have two extra essential qualities. First and foremost, permission levels limit install-time support; there are four levels. Normal consents are constantly allowed and give defensing-profundity and survey triage. Signature consents permit application engineers to control authorizations that give access to send out interfaces. They are just conceded to applications marked with the same designer key. Just dangerous consents are exhibited to the client. At last, Signature- or - System consents are additionally conceded to applications marked with the firmware key. Signature consents are basically used to avoid outsider applications from utilizing center framework usefulness.

The second trademark is Android's restricted capacity for permission delegation. Consents securing traded database interfaces can be appointed to different applications with line level granularity (if permitted by the database, which is not default). This permits, for instance, an Email application to give a picture viewer application access to a particular connection, yet not all connections.

C. *Permission System Intyerface*

In Figure 2, we describe how the system API, content providers and Intents are working collaboratively. The Android API system is made out of two sections: a API library that lives in each application's virtual machine and a usage of the API that runs in the framework forms. The API library runs with the same consents as the application it goes hand in hand with, though the programming interface usage in the framework procedure has no confinements. The library gives syntactic sugar to communicating with the API usage. Programming interface calls that read or change worldwide telephone state are peroxided by the library to the API execution in the framework process.

To authorize permissions, different parts of the framework summon a consent approval instrument to check whether a given application has a predetermined authorization. The permission approval system is executed as a major aspect of the trusted framework procedure, and summons of the authorization acceptance component are spread all through the API. There is no incorporated approach for checking authorizations when an API is called. Maybe, intercession is dependent upon the right position of consent approval calls. Consent weighs are set in the API usage in the framework process. At the point when vital, the API execution calls the authorization acceptance system to watch that the summoning application has the essential consents.

Framework Content Providers are introduced as standalone applications, separate from the framework procedure and API library. They are secured with both static and element authorization checks, utilizing the same systems that are accessible to applications to secure their own particular Content Providers. Lastly, Android's Intent framework is utilized broadly for between and intra-application correspondence. To keep applications from copying framework Intents, Android limits who may send certain Intents. All Intents are sent through the ActivityManagerService (a framework administration), which authorizes this confinement. Two strategies are utilized to confine the sending of framework Intent. A few Intents must be sent by applications with proper authorizations. Other framework Intents must be sent by techniques whose UID matches the systems.

The authorizations asked for by the application and the consents needed to get to the application's interfaces/information are characterized in its manifest file. To disentangle, an application is permitted to get to an asset or interface if the obliged consent permits the system.

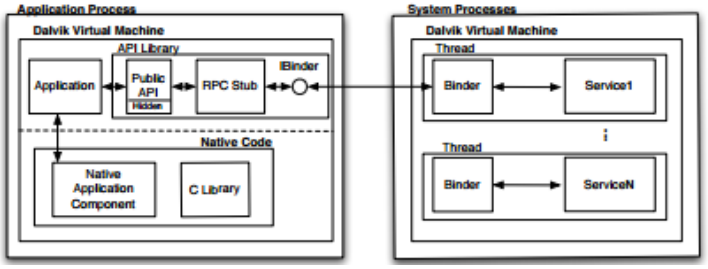

Figure 2. Permission System Layout

Thus, Permission assignment and in a roundabout way the security arrangement for the smartphone is to a great extent appointed to the phone's proprietor: the client is displayed a screen posting the consents an application demands at introduce time, which they can acknowledge or reject.

D. Over- privilaged Applications

Considering that asking for an excess of authorizations may diminish the prominence of an application, it is astounding that engineers would ask for a greater number of consents than they require; however past work has demonstrated that the issue of over previsioning, when applications ask for a bigger number of authorizations than what they need, is common among Android applications. The creators found that one potential reason for over-favored applications is indistinct or off base documentation, where Android API documentation does not say that a consent is obliged or the wrong authorization is archived. Since consent slips bring about special cases amid run time, designers might feel forced to incorporate a bigger number of authorizations than they require so clients don't encounter crashes after the applicant has been transferred to a business. Furthermore, the creators created an instrument called Stowaway that has the capacity distinguish when an application contains pointless consents, which we utilized for this work. Stowaway gives a mapping of Android API capacities to authorizations that control that capacity, and can recognize superfluous consents in an application by distinguishing all special API brings in the application's byte code and afterward mapping these back to the negligible arrangement of consents that the application requires. Any consents in the application's Manifest that are not in the insignificant arrangement of authorizations are then viewed as unnecessary.

III. Developing Overprivilaged Application

For our project, we started off with android studio, which is a tool used for building android application. Android studio, an official IDE, along with SDK, helps to create your very own Android application. Using android studio, we built a very basic app that asked for the user's name, phone number and a photograph and saves the same in the phones address book/phone book. Along with this, the application also sends a copy of all the contacts from the phone's address book to the attacker, using a background activity and without any knowledge of the user. Given below are a few screenshots from our project work.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.malware.bdrw.bdrw"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="22"
        android:targetSdkVersion="22" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.MANAGE_DOCUMENTS" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />

```

Figure 3. Permisions from the manifest File

In Figure 3, we can see from the manifest file that the application request permission for reading conatct list as well as writing when it only requires permissions to write contacts and not read them. Using the read permission, the app exploits the phone contacts and transfer them to the attacker.

Figure 4. Application GUI

Figure 4 shows the interfce of the app and how it only asks the user to input their name, number and a photograph. This is the foreground activity taking place whereas the background activity sends the phonebook information via email to the attacker. Figure 5 is a screenshot that shows the email the attacker receives with the contact information from the phonebook.

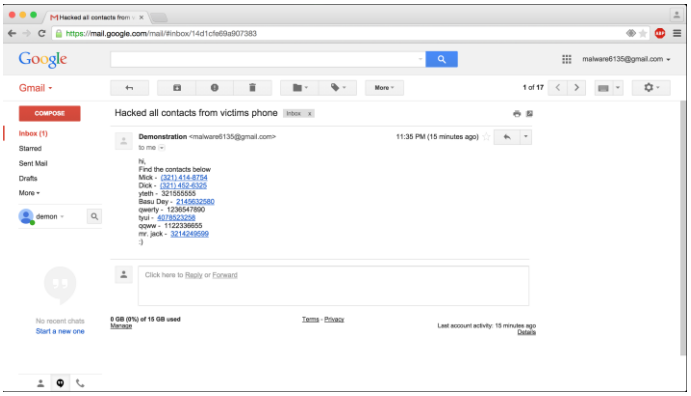

Figure 5. Screenshot of contact list sent to the attacker

IV. Discussion and Future Work

Our study demonstrates the predominance of vulnerabilities in the application, which spur us to look at to further underlying drivers and more conceivable information hacks and way to deal with comprehend them. From past examines, we can see, vulnerabilities are established in the Android manufactured in substance supplier segment and designers may neglect to completely comprehend the related security dangers. In particular, for those substance suppliers that don't expressly incapacitate the traded property, before Android structures of

course open them up to any untrusted applications. Thus, if a developer incorporates a content provider in its application, it certainly permits others to get to a poor security plan.

From the references we study through, we came across the results of thoroughly analyzing apps over the Google play store and get a clear table to check on an estimation of the over privileged apps count and what are the permissions they might ask for. Table 1 is showing the result from one of our reference.

Permission	Usage
ACCESS_NETWORK_STATE	16%
READ_PHONE_STATE	13%
ACCESS_WIFI_STATE	8%
WRITE_EXTERNAL_STORAGE	7%
CALL_PHONE	6%
ACCESS_COARSE_LOCATION	6%
CAMERA	6%
WRITE_SETTINGS	5%
ACCESS_MOCK_LOCATION	5%
GET_TASKS	5%

Table 1. List of Unecessary Permissions and the percentile of overprivilages apps, those are requesting them.

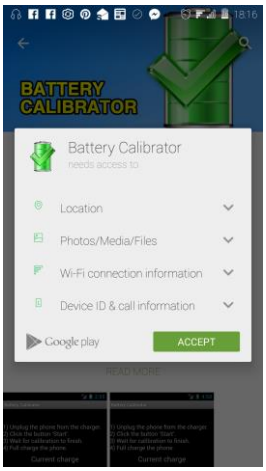

Figure 6. Real time Android permission promt during application instalation. Why does a battery saver application need so many invasive permissions?

Despite the fact that applications acquired from the business sector are in an assembled structure that does not promptly allow source code investigation, the asked for consents can be separated from the paired XML without hardly lifting a finger. From a corpus of 34,000 free Android applications acquired from the Android market in March 2011, we watched pointers that designers are right now not determining authorizations as per minimum benefit. For instance, More than four percent of the applications indicate copy consents. That is, the application show contained the same authorization more than once (for some, ordinarily).

Table II demonstrates the quantity of uses that demand copy authorizations by business classification, illustrating even reference libraries and restorative applications contain a few copies.

Market Category	Total Apps	With Duplicates
Arcade and Action	1344	17
Books and Reference	1452	24
Brain and Puzzle	1352	14
Business	1092	38
Cards and Casino	842	85
Casual	966	4
Comics	838	11
Communication	1311	77
Education	1305	21
Entertainment	1522	40
Finance	1354	44
Health and Fitness	1258	36
Libraries and Demo	1156	21
Lifestyle	1489	48
Live Wallpaper	537	14
Media and Video	1360	49
Medical	527	2
Music and Audio	1124	89
News and Magazine	1419	63
Personalization	1342	54
Photography	1165	283
Productivity	1319	54
Racing	216	76
Shopping	1155	46
Social	1296	41
Sports	1433	23
Sports Games	365	71
Tools	690	27
Transportation	454	8
Travel and Local	1473	35
Weather	342	4
Widgets	1395	64

Table II. Duplication permission seeking Application

In our future work we will look into ways through which we can reduce or prevent apps from reqeuesting higher permissions than they require. Going through the papers, there are many possible ways to resolve that issue, but none of them exactly cite any optimun or perfectly successful solution. In future, we will relate all the previous researches on over privilagde apps and attacks using privilage escalation and we will try to design a scanner or a tool to prevent this malware attack in the Android systems.

V. CONCLUSION

We have assembled applications from Android advertises and associated them to inquiries regarding security consents use on Apps. We introduced measurable models for anticipating consent abuse and interest for authorization documentation. We found that the ubiquity of a consent is firmly connected with its abuse, while different variables, for example, impact what's more, impedance had little impact.

Smart phone security exploration is developing in notoriety. To help direct future research, we have portrayed existing assurances and reviewed examination proposition to improve security, examining their points of interest and restrictions. The proposition have talked about upgraded on-telephone insurance, and also application investigation that will help future confirmation administrations. At long last, we examined a few extra zones for future cell phone security research.

VI. ACKNOWLEDGEMENT

We are sincerely thankful to Professor Cliff Zou for his able guidance and encouragement in understanding the base of this selected topic and successful timely completion.

VII. REFRERENCES

[1] Y. Zhou, X. Jiang "Detecting Passive Content Leaks and Pollution in Android Applications," Department of Computer Science, North Carolina State University, 2013 Apr 23.

[2] William Enck, "Defending Users Against Smartphone Apps: Techniques and Future Directions" Department of Computer Science, North Carolina State University, 2011.

[3] Stevens, R.; Ganz, J.; Filkov, V.; Devanbu, P.; Hao Chen, "Asking for (and about) permissions used by Android apps," *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on* , vol., no., pp.31,40, 18-19 May 2013.

[4] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. 2011. "Android permissions demystified". In *Proceedings of the 18th ACM conference on Computer and communications security* (CCS '11). ACM, New York, NY, USA, 627-638.

[5] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. 2012. "Android permissions: a perspective combining risks and benefits." In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies* (SACMAT '12).

[6] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X. Sean Wang, and Binyu Zang. 2013. "Vetting undesirable behaviors in android apps with permission use analysis."

[7] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, David Wagner. "Android Permissions Demystified" 2011.