IRSTI 20.51

Y. Amirgaliev[1], B. Sayazhan[2], Y. Bakbergen[2]
[1]Suleyman Demirel University, Kaskelen, Kazakhstan
[2]Institute of Information and Computational Technologies, Almaty, Kazakhstan

ANDROID SECURITY ISSUES

Abstract. The smart phones usage has been increased rapidly over the last decade. Android provides third-party applications with an extensive API that uses the permission-based model, which allows Android applications to access user information, system information, device information and external resources of Smartphone. The developer needs to declare the permissions in Android Manifest file for the Android application. Then the user needs to accept that permission for successful installation of an Android application. These permissions are declarations. After installation, if the user allows the permission, the app can access resources and information anytime. Android OS is susceptible to various security attacks due to its weakness in security. In this paper, a number of vulnerabilities are explored in Android permission level and provide an approach for better security in Android Platform. An Attack Scenario is developed successfully for permission-based attacks in the android platform and provides the countermeasures for it.

Keywords: Android security, Permission based security, Access control, Granularity of access control, Policy administration, Over-claim of permission, Permission escalation attack, Data Theft.

***

Аннотация. За последнее десятилетие использование смартфонов быстро увеличилось. Android предлагает сторонние приложения с обширным API, который использует основанную на разрешении модель, которая позволяет приложениям Android получать доступ к информации о пользователях, системной информации, информации об устройстве и внешних ресурсах смартфона. Разработчик должен объявить разрешения в Android. Затем пользователь должен принять это разрешение. Эти разрешения являются декларациями. После установки, если пользователь разрешает разрешение, приложение может получить доступ к ресурсам и информации в любое время. ОС Android подвержена различным атакам из-за слабости безопасности. В этой статье ряд уязвимостей изучается на уровне разрешений на уровне Android и обеспечивает подход для повышения безопасности в платформе Android. Сценарий атаки успешно разработан для атак на основе разрешений на платформе Android и обеспечивает контрмеры для него.

Ключевые слова: Безопасность на базе Android, безопасность на основе разрешений, контроль доступа, гранулярность контроля доступа, администрирование политики, чрезмерное требование разрешения, атака эскалации разрешений, кража данных.

***

Аңдатпа. Соңғы онжылдықта смартфондарды пайдалану жылдам өсті. Android Android қосымшаларына пайдаланушылар, жүйе туралы ақпараттар, алаяқтық туралы ақпарат және сыртқы смартфондар туралы ақпаратқа қол жеткізуге мүмкіндік беретін стандартты рұқсат етілген үлгісін пайдаланатын кең API бар үшінші тарап қолданбаларын ұсынады. Әзірлеуші Android рұқсаттарын жариялауы керек. Сонда пайдаланушы бұл рұқсатты қабылдауы керек. Бұл рұқсаттар декларация болып табылады. Орнатудан кейін, егер пайдаланушы рұқсат берсе, бағдарлама кез-келген уақытта ресурстарға және ақпаратты алуға болады. Android OS қауіпсіздіктің әлсіздігіне байланысты әр түрлі қауіпсіздік шабуылдарына ұшырады. Android, Android, Android, Android, Android, Android және Android. Attack сценарий табысты Android платформасы рұқсат негізінде шабуыл үшін әзірленген және ол үшін қарсы қамтамасыз етеді.

Кілт сөздер: Android қауіпсіздігі, рұқсат ету қауіпсіздігі, кіруді бақылау, қол жеткізуді басқарудың грануллылығы, саясатты басқару, шамадан тыс рұқсат талаптары, эскалация шабуылдары, деректерді ұрлау.

I. INTRODUCTION

A miscellaneous working framework (OS) is programming that permits cell phones, tablet PCs, and different gadgets to run applications and projects. In the market, there are several types of mobile operating system, like Android, iOS, Windows and BlackBerry OS. The Android working framework is an open source and source code discharge by Google under Apache permit license, based on Linux-Kernel designed for smartphones and tablets. Android is one of the most popular operating systems for smartphones. At the last quarter of 2017, the total number of applications available in Google play store was 3.5 Million [1], and a total number of Android operation system-based smartphones sold was 2.32 Billion [2]. The market share of Android in the first quarter of 2017 was 84.4% whereas iOS, Windows, BlackBerry, and others hold 15%, 0.4%, 0.1% and 0.1 respectively [2]. Therefore, it is clear that Android has the widest market when compared others mobile operating systems. In Android, most application installed by google play store, although it is possible to install the applications from unknown sources. However, in iOS, the apps can be only installed from AppStore. It is one of the major security breaches in Android.More developers and more scientific said that iOS more secure than Android. Like MohdShahdi Ahmad et al. [3] indicated the

analysis of Android and iOS, in 2013. It is obvious and also occurs in real life that more adversaries try to hack Android based system than iOS. Attackers already regard smartphone as the target to steal personal information using various malware. In 2014, A, Kaur et al. [4] indicated that it is possible to revoke granted permissions from android application.

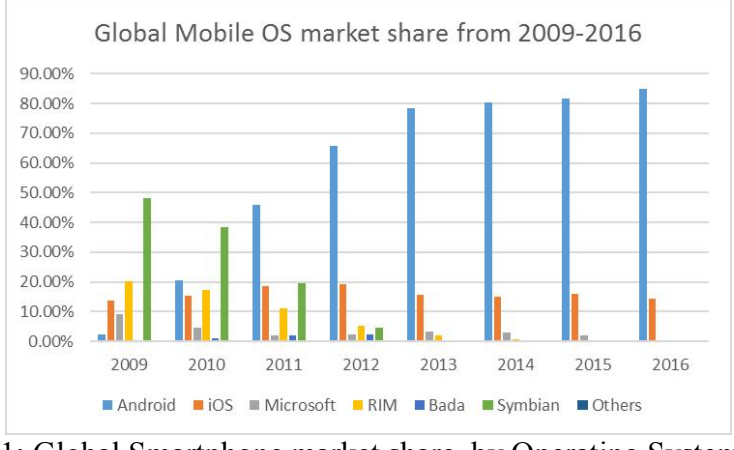

Fig.1: Global Smartphone market share, by Operating System [2]

Android smartphones are protected by a permission-based framework, which restricts third-party applications' accesses to sensitive resources such as SMS database and external storage in Android smartphones. The accesses to sensitive resources may lead to money loss. We can arrange many example, like Android malware may send premium rate messages, make premium rate calls, and generate large amount of network data without users' acknowledgment. Also, most problem will give when leakage of users' private information stored in smartphones such as contacts, emails, and even credit card numbers. Moreover, third-party application developers can leverage various smartphone sensors such as GPS, cameras, and microphones, then create applications that do more than what they claimed so as to collect users' private information stealthily [5].To understand better about Android Platform and apps in the OS let us consider Androids' Architecture in Figure(Figure 2).

The rest of the paper is organized as follows: Section II introduces the background of Android security, different types of Android apppermissions. Section III describes various security attacks on Android such as permission escalation attack, confused deputy attack, TOCTOU (Time of Check and Time of Use) attack. Section IV Security Attacks in Android. Section V Attack Scenario. Section VI Concludes the paper.

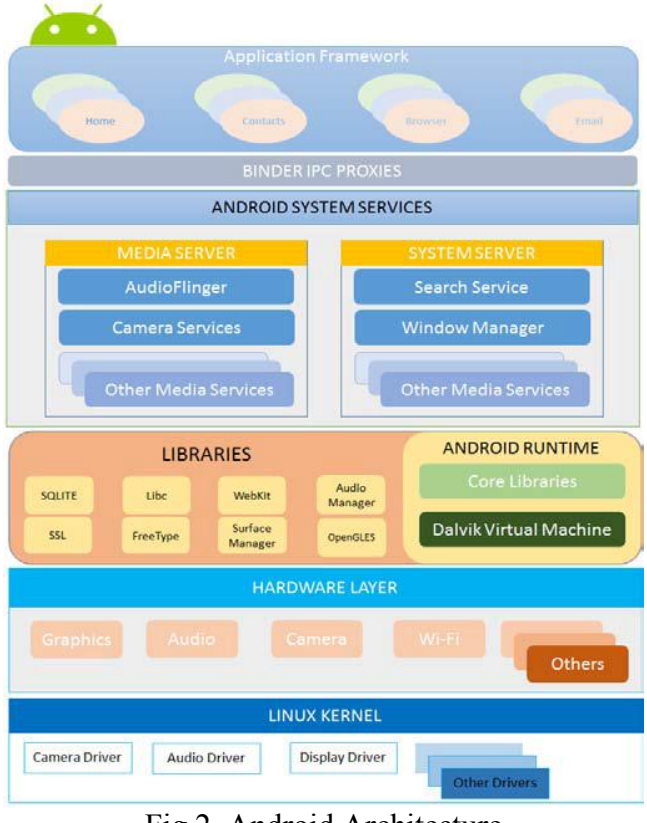

Fig.2. Android Architecture

II. BACKGROUND OF ANDROID SECURITY

Android is proposed as a software stack for mobile devices. IT consists of an operating system, an application framework, and core applications. Each Android application executes in a separate Dalvik virtual machine instance running as a unique user identity assigned at install-time. Thus applications are essentially isolated. This design provides promising security for file accesses and limits potential damage due to programming flaws such as buffer overflow [6].

Android restricts accesses to critical resources using permissions. A permission is simply a unique text string which can be defined by Android or third party developers. According to the documentation for Android developers, there are currently more than 137 permissions [7], which are defined in Andorid operating system, ranging from access to camera (CAMERA), full access to the Internet (INTERNET), dialing a phone number (CALL_PHONE), and even disabling the phone function permanently (BRICK). According to the study of Wei et al. (2012) [8], the number of Android defined permissions keeps increasing since the first widely used release (API level 3). The expansion of the permission set aims at not only

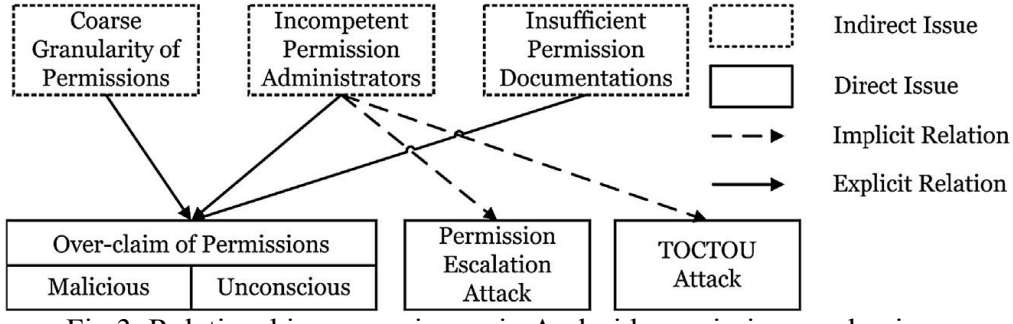

Fig.3. Relationship among issues in Android permission mechanism.

providing finer-grained permissions but also controlling accesses to new hardware features [8]. In addition to Android defined permissions, application developers can also declare customized permissions so as to protect their own critical resources.

Permission may be required when an application is interacting with system resources, including calling system API functions, and reading from and writing to file systems. Granted permissions are assigned to an application's sandbox and inherited aby all of the components, while required permissions are assigned to application components [9]. In the manifest file to an application, which is included in the application package, the application declares the permissions which it requires to achieve its functionality, as well as defines the permissions for protecting its own components and resources. A permission can be associated with one of the following four protection levels [10]:

- Normal Permissions: Normal permissions don't specifically hazard the client's privacy. Normal permissions need not be declared in the AndroidManifest.xml file and causing no hard to users. These permissions are granted automatically. Example:

KILL_BACKGROUND_PROCESSES,
SET_WALLPAPER,
UNINSTAL_SHORTCUT,
WRITE_SYNC_SETTINGS

- Dangerous Permissions: Dangerous Permissions can access critical resources of the mobile. A high-risk permission which allows applications to access potential harmful API calls such as leaking private user data or control over smartphone device. Example:

CONTACTS: READ_CONTANTS, WRITE_CONTACTS,
GET_ACCOUNTS
LOCATION: ACCESS_FINE_LOCATION,
ACCESS_COARSE_LOCATION
SMS: SEND_SMS, RECEIVE_SMS, READ_SMS, RECEIVE_WAP_PUSH,
RECEIVE_MMS

STORAGE: READ_EXTERNAL_STORAGE,
WRITE_EXTERNAL_STORAGE

Android Marshmallow 6.0 has classified the permissions into normal and dangerous permissions. Whenever the app needs to use dangerous permissions, it explicitly asks the user to confirm with the permission. Thus, Android 6.0 and higher versions provide explicit permission notification to access critical resources. Nowadays Android users who have Android 6.0 or higher is more than 62.3 percentage. This other 37.7 percentage of outstayed Android operating system updates are not available for most of the older devices. Therefore, old systems security threats related to app permissions are still not solved.

- Signature/System permissions regulate access to the most dangerous privileges, such as the ability to control the backup process or delete application packages. These permissions are difficult to obtain: Signature permissions are granted only to applications that are signed with the device manufacturer's certificate, and SignatureOrSystem permissions are granted to applications that are signed or installed in a special system folder. These restrictions essentially limit Signature/System permissions to pre-installed applications, and requests for Signature/System permissions aby other applications will be ignored.

IV. SECURITY ATTACKS IN ANDROID

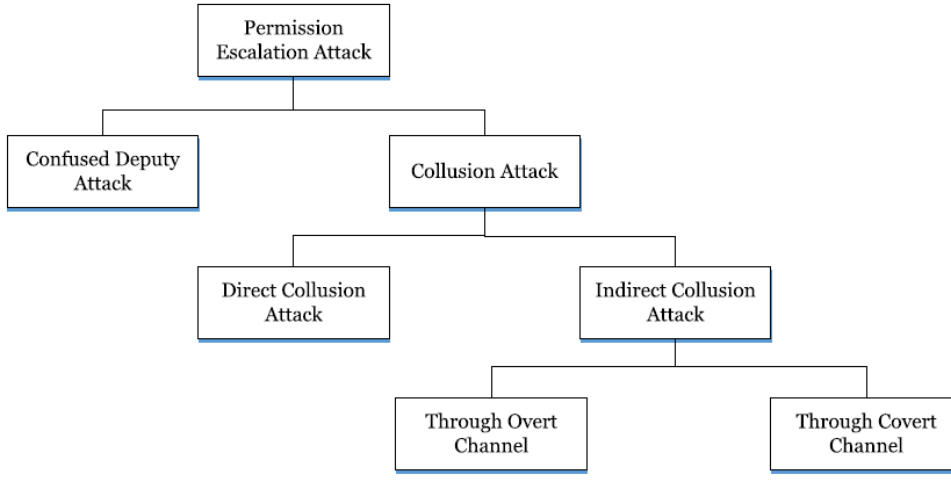

Fig.4. Categorization of the permission escalation attack

A. Over-claim of permissions

Over-claim of permissions is probably the most severe threat to Android security. IT directly breaks the principle of least privilege (PLP) (11). This violation of PLP exposes users to potential privacy leakage and financial

losses. For example, if a standalone game application requests for the SEND_SMS permission which is unnecessary, the permission can be exploited to send premium rate messages without users' acknowledgment.

Felt identified that 56% of the over-privileged applications have only one extra unnecessary permission and 94% have 4 or fewer extra permissions. The low degree of over-claim of permissions per-application indicates that developers attempt to add correct permissions rather than arbitrarily request for a larger number of unnecessary permissions. Developers may make wrong decisions because of several reasons, concluded by Felt [12], including: at first, developers tend to request for permissions with names that look relevant to the functionalities they design, even if the permissions are not actually required; second, developers may request for permissions which should be requested by deputy applications instead of their own applications; finally, developers may make mistakes due to using copy and paste, deprecated permissions, and testing artifacts.

As shown in Figure 3, the issue of over-claim of permissions can be categorized into: malicious and unconscious. Three other issues, including coarse granularity of permissions, incompetent permission administrators, and insufficient permissions documentation, are of over-claim of permissions.

B. Permission Escalation Attack

In contrary to the general belief that the damage imposed by an Android malware is limited to an application's sandbox, the permission escalation attack allows a malicious application to collaborate with other applications so as to access critical resources without requesting for corresponding permissions explicitly [13].

Figure 4 shows the categorization of the permission escalation attack. The permission escalation attack can be classified into two categories: confused deputy attack and collusion attack.

The confused deputy attack exploits the vulnerabilities in unprotected interfaces of privileged benign applications [14]. As shown in Figure 5, the application A_1 is not granted with the permission P_1; C_1, which is a component of A_1, cannot directly access system resource R_1 protected by permissions P_1. However, C_1, can access R_1 transitively if application A2 is granted with permission P1 and one of A2's component, C2 does not require any permission to be accessed. As a result, C1 can access R_1 through C2 and C2 [13].

In addition, the collusion attack can be carried out by multiple applications in generating a joint set of permissions which enables them to perform an unauthorized or malicious actions [14]. The collusion attack can be further classified by the way applications communicate with each other, into direct collusion attack where applications communicate via a third application or component in between [13].

The indirect collusion attack usually involves another application or component as a mediation which can provide either overt channels or covert

channels [15]. Overt channels, such as buffers, files and I/O devices, use a data object as the entity to hold certain information. In other words, the entity is an object that is normally viewed as a data container [16]. Other examples of overt channels include shared preferences, system logs, and UNIX socket communication [15].

In contrast, convert channels use entities which are normally not intended to be used for communication [17] [15]. It is possible to use covert channels to bypass the Android's middleware layer [13]. Examples of convert channels include file locks [15], change of screen state, change of vibration settings [13], type of intents, threads enumeration [16], etc. Marforio et al. measured the throughput of a number of overt and covert channels. The measurement shows that even covert channels with low throughput are still sufficient to exchange private information [16].

It allows a malicious application to collaborate with other applications so as to access critical resources without requesting for corresponding permissions explicitly.

C. Collison Attack

Android supports shared user ID [18] [19]. It is a technique wherein two or more application share the same user id so that they can access the permissions which are granted to each other. For example: If application A has permissions to READ_CONTACTS, READ_PHONE_STATUS and B has permission to READ_MESSAGES, LOCATION_ACCESS, if both the application use the same id SHAREDUSERID, then it is possible for application A to use the permissions granted to itself and the permission granted to B. Similarly, it is possible for application B to use the permissions granted to itself and the permissions granted to A. Every Android application has unique ID that is its package name. Android supports shared User ID. It is an attribute in Android Manifest.xml file. If this attribute assigned with the same value in two or more applications and if the same certificate signs these applications. They can access permissions granted to each other.

Collision attack has been classified as direct collision attack and indirect collision attack. A direct collision attack is wherein application communicates directly. In Indirect collision attack application communicates via third party application or component.

D. Time of Check and Time of Use Attack (TOCTOU attack)

The vulnerability of TOCTOU exists in Android mainly due to naming collusion. No naming rule or constraint is applied to a new permission declaration [20]. Moreover, permissions in Android are represented as strings, and any two permissions with the same name string are treated as equivalent even if they belong to unrelated applications [5]. Malicious application developers may exploit this flaw. Suppose a malicious developer manages to trick a user into installing malicious application A which declares permission P', and another malicious application P' is the same as permission P which

protects accesses to a critical resource. Afterward the user uninstalls application A and installs benign application C which declares permission P. Now the malicious application B would be able to use permission P' to access the critical resource.

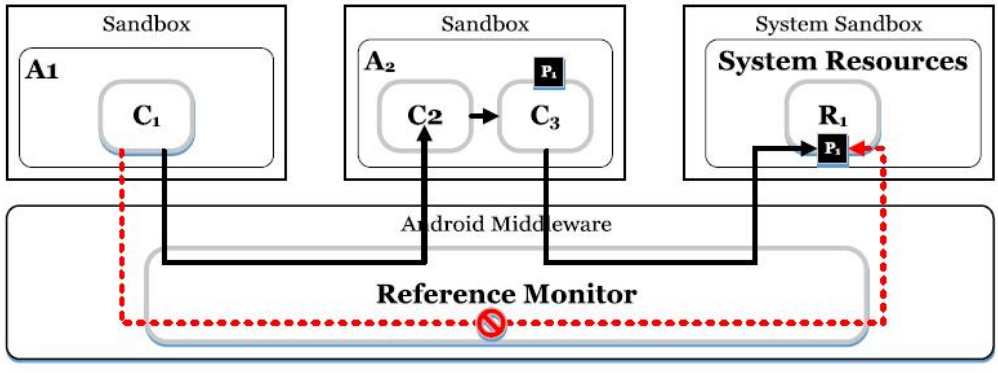

Fig.5. Permission escalation attack [13]

V. ATTACK SCENARIO

An attack scenario is the number of processes of the targeted device or victim's attack, which is going to be happened in order to put some damages or misuse on the targeted device or network. In another definition, an attack is an action that put a potential violation in the system or victim's device. An attack can be classified according to the way of performance., whether this performance is done by a person or a software. The person who executes such type of an attacker is to misuse the vulnerabilities of an organization or a device, which can be a smartphone, running Android OS, etc. An attack [21] situation describes here the actors if an information system, their secure capabilities as well as possible attackers and their goals.

An attack scenario of Permission-Based Attack o Android devices like smartphones, Tablets and so has been developed. Today most of the apps for Android devices are eager to have access in almost whole device, including hardware, middleware and software frameworks. These apps are putting a bunch of request permissions in their manifest files and asking the end-user while installation.

The user who knows or does not know anything-about device or personal information security, simply grant the permission otherwise apps installation will be canceled which is the users' unwanted action. The user has only two options either grant the permission or simply cancel the installation, which is not good from the app side.

Having access to almost all the device and user's privileges.

- **Possible Attackers:** Attacker who is the App developer knows that the user either should install or simply cancel the installation.

- **Possible Vulnerabilities:** Having access to almost all the device properties and end user's data.
- **Resources Affected:** Android Device, Android Device's Network, Android User's data and so.

```
public static final class permission_group {
    public static final String CALENDAR = "android.permission-group.CALENDAR";
    public static final String CAMERA = "android.permission-group.CAMERA";
    public static final String CONTACTS = "android.permission-group.CONTACTS";
    public static final String LOCATION = "android.permission-group.LOCATION";
    public static final String MICROPHONE = "android.permission-group.MICROPHONE";
    public static final String PHONE = "android.permission-group.PHONE";
    public static final String SENSORS = "android.permission-group.SENSORS";
    public static final String SMS = "android.permission-group.SMS";
    public static final String STORAGE = "android.permission-group.STORAGE";

    public permission_group() {
        throw new RuntimeException("Stub!");
    }
}

public static final class permission {
    public static final String ACCESS_CHECKIN_PROPERTIES = "android.permission.ACCESS_CHECKIN_PROPERTIES";
    public static final String ACCESS_COARSE_LOCATION = "android.permission.ACCESS_COARSE_LOCATION";
    public static final String ACCESS_FINE_LOCATION = "android.permission.ACCESS_FINE_LOCATION";
    public static final String ACCESS_LOCATION_EXTRA_COMMANDS = "android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"
    public static final String ACCESS_NETWORK_STATE = "android.permission.ACCESS_NETWORK_STATE";
    public static final String ACCESS_NOTIFICATION_POLICY = "android.permission.ACCESS_NOTIFICATION_POLICY";
    public static final String ACCESS_WIFI_STATE = "android.permission.ACCESS_WIFI_STATE";
    public static final String ACCOUNT_MANAGER = "android.permission.ACCOUNT_MANAGER";
    public static final String ADD_VOICEMAIL = "com.android.voicemail.permission.ADD_VOICEMAIL";
    public static final String ANSWER_PHONE_CALLS = "android.permission.ANSWER_PHONE_CALLS";
    public static final String BATTERY_STATS = "android.permission.BATTERY_STATS";
    public static final String BIND_ACCESSIBILITY_SERVICE = "android.permission.BIND_ACCESSIBILITY_SERVICE";
    public static final String BIND_APPWIDGET = "android.permission.BIND_APPWIDGET";
    public static final String BIND_AUTOFILL_SERVICE = "android.permission.BIND_AUTOFILL_SERVICE";
```

Fig.6. Android Manifest class

Attack scenario step-by-step:

1. The Developers develop an app and upload it to an Application Store.
2. The user downloads the App in an Android Device from the App Store
3. Before installation, the App provides a list of permissions for the users to allow access to the Device and Data in that device
4. The user has two options, either grant the requested permission or simply cancel the installation, no other choice.
5. The user simply grants the requested permissions either understands the permissions or unknowingly, proceeding the installation.
6. Once the installation is completed, the App Developers have full access to that Android device and its data as of figure 7 (a), (b).

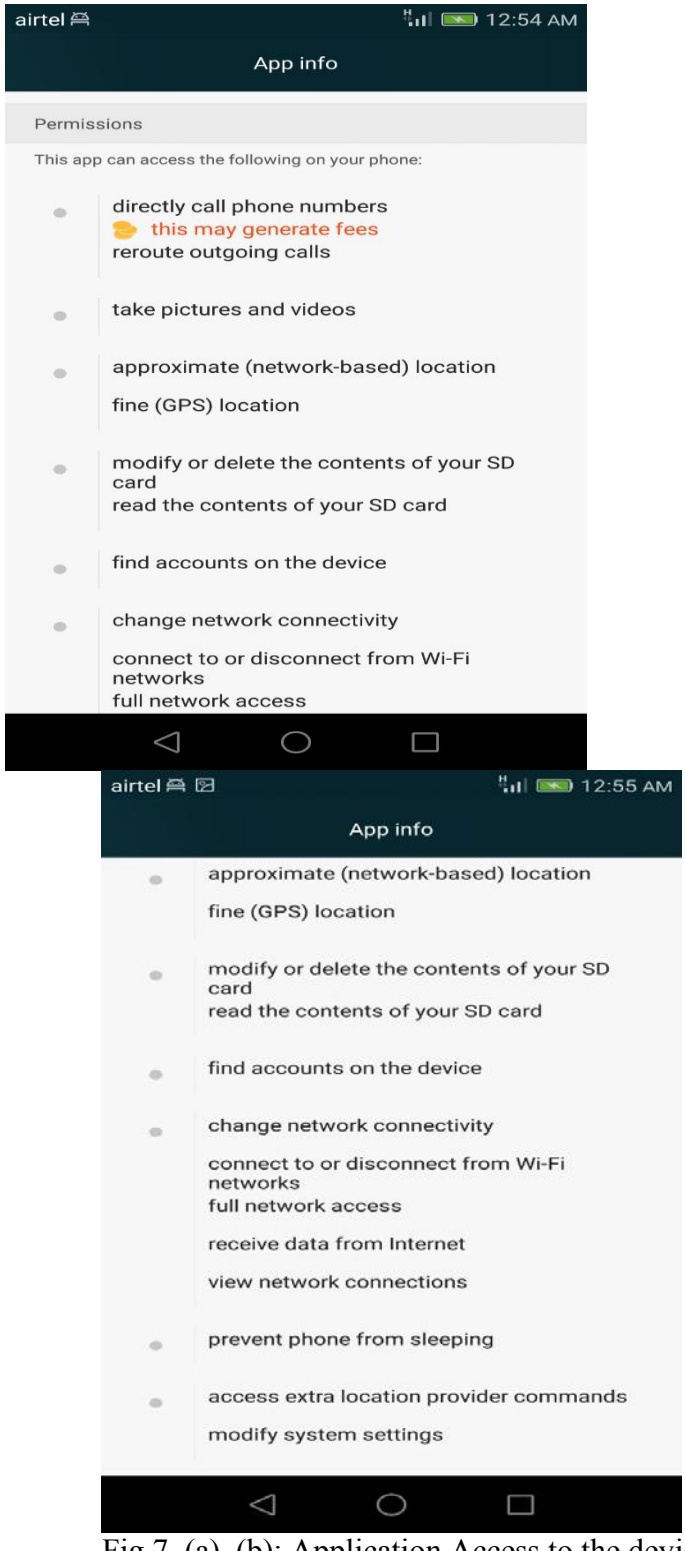

Fig.7. (a), (b): Application Access to the device after installation

Some example of this attack, is the simply our app that use every day. It asks so many permissions to work with user and without giving this permission user can't interact with this application. This app, store all data where user give permission. It's Facebook app. Ok, let's go ahead and try to see what our app store about user by permissions that user has granted.

First go to Settings and click 'Download a copy of your data' they'll email it to you. Second Downloaded my Facebook data as a ZIP file. Finally extract it and see what inside. It was so unbelievable, that our trusted and the most used app store all data about user that confidential. Then which app we will trust? Yes, answer is no one. If application asks many permissions user need to check, does this permission so necessary to work.

And if we came back to Facebook user can easily see that data which it stores is contain a historical record of every single contact on my phone, including ones I no longer have, metadata about every text message I've ever received or sent even user don't use Messenger (also Facebook app for messaging) for SMS and the metadata of every cellular call user has ever made, including time and duration.

I've written a script to dump all cell records in the ZIP file and print statistics Here is what it shows on my profile, pretty chilling as shows figure 8.

Fig.8. All records that Facebook app stores.

VI. COUNTER MEASURES

Preventing permission is the main goal in this Paper. There are number of ways to prevent permissions theoretically, Apps installation without asking permissions causes to installation of malware or tracking software in the device. Even though Google Play Store tries too much to manage the Apps it hosts, but there are still many apps to be truly effective. The responsibilities left to the end-users to protect the device. There are couple of ways to protect an Android device from malicious software that taking over the Android device.

The users can be requested to stop automatic updates in Android. Although automatic updates are useful in general, but there might be a time that is going in the background without end-users knowledge and can install malicious codes in the device.

The second way to prevent permission is to stop unsigned Application installing. Users can have the choice of installing apps either from Google Play Store or from any other Application Stores like Amazon Store. Android

developers recommend using Play Store for installing any app in any Android device, because it is secure and reliable than any other Application Stores.

VII. CONCLUSION

This Paper review Android permission overalland Android background security, a Permission-Based Attack and small review about Attack Scenario developed to inspect and demonstrate permissions attacks on devices running Android operating system platform. Also reviewed Facebook app that store so many data about user. Android security need more improvement, android security developer can say that it is not enough that system and code is secure, also user should know base security rules and protocols. Nevertheless, Android security developer must minimize these vulnerabilities by providing more securely platform and protocols. For Future Work, first is the app developers' consideration through requesting permission as required for the app. Then, is the Android management to do not allow the users to install unrelated apps installation in an Android device.

References:

1 Number of Google play store apps 2017 | statistic, Statista, 2014. [Online resource]. Available:http://www.statista.com/statistics/266210/number-ofavailable-applications-in-the-google-play-store. (Date of commitment: 05.06.2018).

2 "Smartphone users worldwide 2014-2020 | statistic," Statista, 2017. [Online resource]. Available: https://www.statista.com/statistics/330695/number-ofsmartphone-users-worldwide. (Date of commitment: 06.06.2018).

3 Ahmad, M. S., Musa, N. E., Nadarajah, R., Hassan, R., and Othman, N. E., "Comparison between android and iOS operating system in terms of security" // 8th International Conference on Information Technology in Asia (CITA), Jul. 2013. – Pp. 10-18.

4 Kaur, A. and Upadhyay, D., "PeMo: Modifying application'spermissions and preventing information stealing onsmartphones" // 5th International Conference -Confluence The Next Generation Information Technology Summit (Confluence), Sep. 2014. – Pp. 6-20.

5 Fragkaki, E., Bauer, L., Jia, L., Swasey, D. Modeling and enhancing android's permission system. // Computer Securitye ESORICS, 2012. Springer – Pp. 1-18.

6 Enck, W., Octeau, D., McDaniel, P., Chaudhuri, S. A study of android application security.// In: Proc. of USENIX security, 2011. – Pp. 5-13.

7 Android. Manifest. permission [Online resource] Available: http://developer.android.com/reference/android/Manifest.permission.html; 2013b. (Date of commitment: 20.05.2018)

8 Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M. Permission evolution inthe android ecosystem. // In: Proc. of ACSAC. ACM; 2012. – Pp. 31-40.

9 Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A. XManDroid: a new Android evolution to mitigate privilege escalation attacks. – Technische Universita¨ t Darmstadt; 2011a [Technical Report; Technical Report TR-2011-04]. – Pp. 5-9.

10 Android http: // developer. android.com / guide / topics / manifest / permission-element.html; 2013a.

11 Saltzer, J.H. Protection and the control of information sharing inmultics. Commun ACM 1974;17(7) – P. 388-402.

12 Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D. Android permissionsdemystified.// In: Proc. of ACM CCS. ACM; 2011b. – Pp. 627-638.

13 Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.. XManDroid: a new Android evolution to mitigate privilege escalation attacks. – Technische Universita¨ t Darmstadt; 2011a [TechnicalReport; Technical Report TR-2011-04]. – Pp. 5-25.

14 Dietz, M., Shekhar, S., Pisetsky, Y., Shu, A., Wallach, D. Quire: lightweight provenance for smart phone operating systems. // In: Proc. of USENIX security; 2011. – Pp. 23-35.

15 Schlegel, R., Zhang, K., Zhou, X., Intwala, M., Kapadia, A., Wang, X. Soundcomber: a stealthy and context-aware sound trojan forsmartphones. // In: Proc. of NDSS; 2011. – Pp. 17-33.

16 Marforio, C., Ritzdorf, H., Francillon, A., Capkun, S. Analysis of thecommunication between colluding applications on modernsmartphones. // In: Proc. of ACSAC. ACM; 2012. – Pp. 51-60.

17 Kemmerer, R.A. A practical approach to identifying storage andtiming channels: twenty years later. // In: Proc. of ACSAC. IEEE; 2002. – Pp. 109-118.

18 Fang, Z., Han, W., and Li, Y., "Permission based Android security: Issues and counter measures"// Computers &Security, Jun. 2014. – vol. 43, – Pp. 205–218.

19 Xing, L., Pan, X., Wang, R., Yuan, K., and Wang, X., “Upgrading Your Android, Elevating My Malware: Privilege Escalation Through Mobile OS Updating” // IEEE Symp. Secur. Priv., 2014. – P. 101-110.

20 Shin, W., Kwak, S., Kiyomoto, S., Fukushima, K., Tanaka, T. A small but non-negligible flaw in the android permission scheme. // In: IEEEPOLICY. IEEE; 2010. – P. 107-110.

21 Eric Cole et. al “Constructing Attack Scenarios for Attacker Profiling and Identification”, [Online source]. Available: “http://www.securityhaven.com /docs/ Constructing Attack Scenarios for Attacker Profiling and Identificationv6. pdf, Jun2010. (Date of commitment: 01.06.2018).