Data Security Evaluation for Mobile Android Devices

Igor Khokhlov, Leon Reznik
Rochester Institute of Technology
Rochester, NY, USA
{ixk8996, lrvcs}@rit.edu

***Abstract*—Vast spreading of Android-based devices with embedded diverse sensors make them popular platforms for data collection in various research domains. However, Android OS complexity and vulnerabilities may facilitate sensor data manipulation and falsification that will have a high impact on data security and quality. This paper presents a data security and quality evaluation method based on a possibility assessment of malicious actions that exploit certain Android OS features and vulnerabilities as well as applications installed. It describes the Android OS architecture features related to security and major security mechanisms. The paper shows metrics chosen for data security evaluation and the developed method of their integration that computes Overall Security Evaluation Score representing sensor originated data trustworthiness level. The examples of Overall Security Evaluation Score calculation are presented and analyzed.**

I. Introduction

Nowadays, Android mobile OS is the most popular mobile platform in the world. Installed on about 315 million devices, in the third quarter of 2016 it dominated the smartphone market with a share of 86.8% [1]. Broad spread out of Android OS and its open-source nature have attracted software developers and led to enormous amount of applications available on the market, from games to scientific tools. These applications may have a very specific target [2] or used in various research domains.[3]. Modern smartphones are equipped with numerous sensors that make them a popular platform for conducting research in various areas.

Growing popularity of the Citizen Science significantly expanded sources of the data that are generated and could be used for research purposes. However, wider participation of non-professional researchers who bring their uncalibrated and untested devices to collect data may result in generation of huge amounts of unreliable and untrustworthy data. Researchers face novel challenges during data aggregation with data security evaluation being one of the most important. With at least several sensors embedded into each device, an Android-based smartphone represents a complex platform that combines communication, processing and data collection abilities . Data from Android devices could be corrupted or manipulated accidentally or intentionally. Due to complexity of such platforms, evaluation of data trustworthy level becomes an important issue.

Since Android OS initial release, a variety of vulnerabilities and attacks have been discovered [4] that demonstrates various shortcomings in the Android OS security mechanisms. The major protection components of the Android OS security framework are permission mechanisms, application sandboxing, and application signing.

This paper presents a method of sensor originated data security evaluation for Android-based devices. The method evaluates possibility of various data manipulation attacks. It is based on various metrics of a Android-based device and takes into account already discovered system vulnerabilities as well as possible ways of data falsification and manipulation. Due to modularity of the Data Security and Quality Evaluation Framework [5], developed method can be easily integrated into it. Thus, the proposed method may introduce additional metrics used for the data security and quality evaluation.

Android OS Architecture features related to the security components that might be explored by attackers are presented in the section II. Possible attacks of sensor data manipulation and falsification are described in the section III. Section IV is devoted to various metrics that are used in the sensor originated data trustworthiness evaluation. Finally, section V presents evaluation the workflow diagram, sensor data trustworthiness level definition and examples.

II. Current Android OS architecture features related to security mechanisms

Twelve major versions have been released since Android OS launch in 2009. Currently [6], 98% of all Android-based devices use one of five versions: Jelly Bean (versions 4.1.x - 4.3) with 11.3% share, KitKat (version 4.4) with 21.9%, Lollipop (versions 5.0 and 5.1) with 32.9%, Marshmallow (version 6.0) with 30.7% and Nougat (versions 7.0 and 7.1) with 1.2%. Despite a discontinued support of Jelly Bean, KitKat and Lollipop, these versions are used by 66.1% of all Android powered devices. However, the share of these versions is decreasing. Also, Android OS has various modifications made by hardware platform manufacturers. Such modifications may range from pre-installed applications to modifications in the OS kernel. This variety is called Android OS fragmentation. In addition, because of Android OS open source nature, various enthusiasts produced numerous unofficial versions.

A typical Android architecture is presented in Fig.1. It is composed from the basic layer which is the Linux kernel that communicates with platform hardware and sensors. The Hardware Abstraction Layer (HAL) provides standard interfaces of hardware components. Native C/C++ Libraries layer contains high performance libraries. Android Runtime (ART) and its predecessor Dalvik (for Android OS versions bellow 5.0) execute Java code.

ISSN 2305-7254

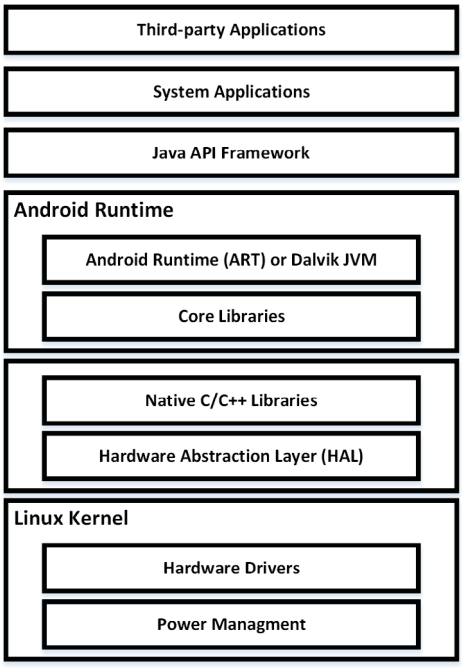

Fig. 1. Typical Android Architecture

Android is based on Linux kernel. It implements the Linux Discretionary Access Control (DAC). Each application is assigned with Unique ID (UID) and has its own sandbox. Such approach prevents applications to interfere with each other and represents one of the main protection mechanism in the system. System permissions present another important protection. Permissions restrain an application from an unauthorized access to the system's resources. They are declared in the Manifest file, which is an inseparable part of an Android OS application. System permissions could be classified into four levels: normal, dangerous, signature and signatureOrSystem. Prior to Android OS version 6.0, a user had to accept all declared permissions during an installation process. In version 6.0 new Runtime Permissions of the dangerous level were introduced. Now a user can install an application with declared dangerous permissions but without granting such permissions.

Any application installed on an Android OS is signed by the developer. In order to update an application, install a new version and keep all application and user data in the application sandbox, new version application should be signed with the same signature. Otherwise, Android OS requires deleting an application and clearing all data related to it. Such an approach prevents cases when an infected application replaces an old version and captures user data. However, an application signature mechanism works only if a developer of the infected version does not have the signature key of the original application.

Each application may consist of several components that may have various types: Activity, Service, Content Provider, and Broadcast Receiver. In addition, an application may have multiple entry points, each corresponds to a specific component. Service components can perform actions in background without any explicit signs, that creates a good ground for possible malicious actions. Moreover, an application or its component can be launched by other applications directly or through broadcasting an Intent. For example, the application "A" consists of two components: Activity and Service. The application "B" can launch a service component of the application "A" and pass to it any data it needs. The service component of the application "A" should be capable to process the data received.

The Google Play Store represents another Android OS security frontier. All applications that are uploaded there have been checked with Google's Bouncer malware prevention system. However, due to code obfuscation and other techniques that help to hide malicious code, Bouncer may not guarantee that the Google Play Store is malware free [7].

III. Possible attacks of sensor data manipulation

In this section possible, scenarios of the malicious sensor data manipulations are presented. We consider two sensor data manipulation scenarios: data manipulation performed by a smartphones' legitimate user or by another person. This section describes possible sensor data manipulation on a physical device, not on a virtual device. On an android virtual device (AVD) any sensor data can be easily emulated through AVD manager, which is included in the Android SDK.

A. Possible malicious data manipulations performed by smartphones' legitimate user

This category includes more scenarios than the second one. Smartphone's legitimate users commonly have more access, including connection to a PC, than other persons. We consider only deliberate malicious actions. Data from sensors can be manipulated by various software tools.

1) Touchscreen and buttons data integrity violations: These attacks produce fake data from touchscreen and hardware buttons. Commonly this data can be emulated through Android Debug Bridge (ADB) shell [8]. ADB is a software tool that allows communicating with an Android device. ADB facilitates device actions such as: installing applications, debugging applications, and provides access to a Linux shell, which can be used for running various commands on the device. ADB can have a remote access to a device either trough wired or wireless connections. For example, the following two commands successfully emulate power button pressing:

```
adb shell sendevent /dev/input/event0 1 116 108
adb shell sendevent /dev/input/event0 0 0 0
```

However, such method requires a device to turn on a "Debug mode" and to connect to the PC with installed Android SDK software. These commands can be executed either through wired or wireless connections. In the same way data from a touchscreen, volume buttons, headphone jack events, and a keypad can be emulated.

2) GPS data falsification: This attack includes a possible production of fake GPS data. For spoofing GPS data, one can write an application or use an existing applications that are available in the Internet or Google Play Store [9]. This method requires enabled Developer menu in the Android OS settings, which is disabled by default, with turned on "Allow

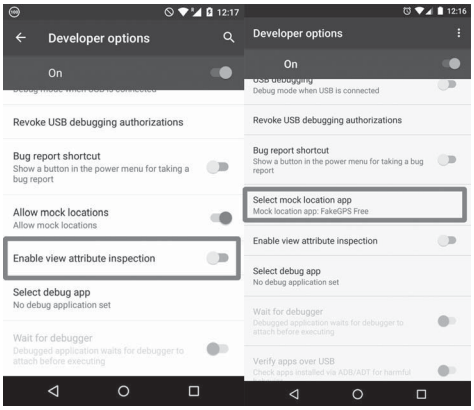

Fig. 2. Mock Location Settings. Android v.5.1.1 (left) and Android v.7.1.1 (right)

Mock Location" option (Fig.2). However, if an Android system is "rooted", a location can be faked even without enabling the developer menu. "Rooted" device means that installed applications can gain SuperUser privileges. Applications with SuperUser privileges have read/write rights to any file and folder in the system.

In order to develop the protection mechanism against this attack, the author conducted with various Android OS versions an empirical study that demonstrated the behavioral deviations. For example, Android v.7.1.1 has been found to keep true location data even with enabled "mock location" option. That leads to continuous changing data between true and fake. Such behavior of GPS sensor can be used to detect this attack. However, Android v.5.1.1 does not restore a true data and this protection mechanism will not work on this version. That requires a further investigation.

3) Environment, position and motion sensor data manipulation: This attack attempts to manipulate data from environment, position and motion sensors categories. The environment sensors category includes following sensors: ambient temperature sensor, illuminance sensor, ambient air pressure sensor, ambient relative humidity sensor, device temperature sensor. The position sensors category includes magnetometer and proximity sensor. Also, position sensors include software sensors based on the sensors from motion category. The motion category includes: 3-axes accelerometer sensor, 3-axes gyroscope and step-counter. In order to manipulate data from the sensors of these categories, one need to modify corresponding files in the "/dev/input/" directory. Changing files in this directory requires SuperUser (root access) privileges. If one gets SuperUser rights, they gain unlimited possibilities for fulfilling this attack.

B. Sensor data manipulation by non-legitimate user of a device

Certain attacks can be performed by a non-legitimate user. However, such attacks require gaining the permissions from the dangerous category and/or SuperUser rights, Table I. In the versions preceding Android v.6.0 user had to accept all declared permissions during an installation process or reject an application installation. In the version 6.0 a runtime permission model of the "dangerous" category was introduced.

TABLE I. DANGEROUS PERMISSION GROUPS

Permission Group	Permissions
Calendar	• READ_CALENDAR • WRITE_CALENDAR
Camera	• CAMERA
Contacts	• READ_CONTACTS • WRITE_CONTACTS • GET_ACCOUNTS
Location	• ACCESS_FINE_LOCATION • ACCESS_COARSE_LOCATION
Microphone	• RECORD_AUDIO
Phone	• READ_PHONE_STATE • CALL_PHONE • READ_CALL_LOG • WRITE_CALL_LOG • ADD_VOICEMAIL • USE_SIP • PROCESS_OUTGOING_CALLS
Sensors	• BODY_SENSORS
SMS	• SEND_SMS • RECEIVE_SMS • READ_SMS • RECEIVE_WAP_PUSH • RECEIVE_MMS
Storage	• READ_EXTERNAL_STORAGE • WRITE_EXTERNAL_STORAGE

The dangerous category consists of several groups, where each group may include several permissions. Dangerous permission groups are presented in the Table I.

Declared permissions are not shown to the user during an installation process. However, if an application needs to use some of a device's resources, it should explicitly ask a user for granting a permission (Fig.3) with an explanation why an application needs this permission and resources. If a user granted at least one permission from the group, other permissions from the same group are granted automatically. All requested permissions should be declared in the manifest file. This approach may solve a problem with over-permission applications, when an application requests more permissions that it needs and a user have no choice but to accept them.

However, application "A" can reasonably request a bunch of dangerous permissions and the user may approve them. Also, application "A" has various components like services and broadcast receivers. One of such receivers may have malicious intentions and can be triggered with a special event outside the application "A". The application "B" does not request dangerous permissions, but designed to work with malicious component from the application "A". Application "B" can send a special event and data that triggers the malicious component in "A". In such a way, application "B" implicitly gains dangerous permissions. The same scenario can be used with the third party applications that have root access. Obfuscation techniques make analysis of such applications very difficult. Also, these applications may detect installed anti-malware software and does not reveal malicious intentions [10]. To evaluate this attack possibility external information is required. The declared permissions and applications rating in the Google Play Store can help. However, signatures of the installed applications should be compared with the applications signatures in the Google Play Store.

Android OS is an open source project (AOSP) and can be modified. Almost all smartphones vendors use Android OS with various levels of modification. Such modifications very often include stock applications customization, adding vendors application, and third-party applications, which tend

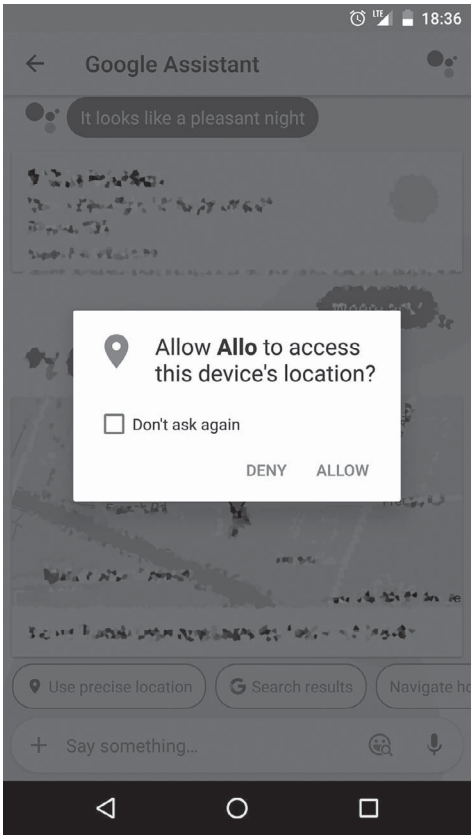

Fig. 3. Runtime Permission dialog example. “Allo” application is asking for location permission.

to increase possible vulnerabilities [11]. It does not mean, that original version of AOSP does not contain vulnerabilities. Smartphones branded by Google Inc. have versions of Android OS with the minimal number of modifications. Moreover, Google releases a security update that fixes discovered security vulnerabilities for latest Nexus and Pixel devices each month.

In addition, due to the Android open source nature, there exist numerous custom versions of AOSP-based Android OS made by enthusiasts or software companies. One of the most famous is the former “Cyanogenmod” and now known as “LineageOS” [12]. Less famous custom versions of Android OS, often made by several developers are likely to have more software flaws.

IV. Metrics of sensor originated data security evaluation

This section determines some metrics that should be measured in order to evaluate a security level of the sensor originated data. These metrics are related to Android OS security mechanisms and possible attacks that were described in the section II and III. They complement metrics that were introduced by Weiss et al.[13]. As not all metrics can be measured on a smartphone, we split metrics into two groups: Internal metrics, which are calculated using data only from a device and External metrics, which are calculated with external information. Metric’s classification and score used for their calculation are presented in Table II.

A. Internal metrics

1) Root access: As shown earlier, SuperUser privileges, also known as “root access”, have a big impact on the possibility of the sensor data manipulation. Gaining root access allows to fake any sensor data on a device. Often, root access can be achieved through flashing third-party Android OS image. Also, there exist software that allows to gain root access using system vulnerabilities, for example, “Kingo Root” software [14]. “Root access” vulnerability metric may have two values: 0 if root access is gained and 1 otherwise.

2) Unlocked bootloader: Bootloader is used to load either OS or recovery software. Locked Android OS bootloader does not allow to load unverified or untrusted third-party recovery software or Android OS. In most cases to gain “root access” unlocked bootloader is required. Sometimes, a bootloader unlock process involves dangerous procedures that can break a device. Some vendors, for example HTC, cancel or limit warranty after bootloader was unlocked [15]. Unlocked bootloader vulnerability metric may have two values: 0 if bootloader is unlocked and 1 otherwise.

3) Device lock: Device lock is very important mechanism to secure one’s personal data and the device from undesirable actions from fraudsters. An unlocked device makes easier for non-legitimate users to install various applications, unlock bootloader, gain root access without a legitimate user awareness. Android OS by default provides a few primary lock mechanisms: pin code, pattern and password. A user can choose only one primary lock mechanism. Also, Android OS provides secondary unlock mechanisms, that make unlock process easier. These mechanisms are:

- Fingerprint - if device have fingerprint sensor, legitimate user’s fingerprint can be used to unlock a device;
- Trusted Face - a device uses a front camera to recognize a user’s face and unlock a device;
- Trusted Places - keeps a device unlocked in chosen locations within a radius of up to 80 meters;
- Trusted Voice - unlocks a device by phrase “OK, Google” if the system recognizes a user’s voice;
- Trusted Devices - unlocks a device if it connected to chosen device(s) via bluetooth. Also, a NFC tag can unlock a device;
- On-Body Detection - if a device was unlocked this mechanism keeps it unlock while a user carries the device. Uses an accelerometer and in some devices can learn and identify user’s walk patterns.

A user can choose several secondary lock mechanisms. However, these mechanisms are less secure and, in some cases, the Android OS requires to use a primary unlock mechanism. For example, after a device reboot or four hours of inactivity, the device can be unlocked only with a primary mechanism.

Assigned score could be: 0 - no lock mechanism, 1 - secondary lock mechanism, 2 - only primary lock mechanism.

4) Android OS version: It is likely that the latest versions of Android OS have less vulnerabilities than older ones. Although

this metric is measured on a device, it is used in the external metrics measurement.

Assigned score could be: 0 - outdated Android OS version, 1 - previous Android OS version, 2 - the latest Android OS version.

5) Security patch version: Some vendors, for example Google, release patches that fix discovered vulnerabilities. The latest security patch that is installed on a device makes a device more secure and increases overall security score. Although this metric is measured on a device, it is used in the external metrics measurement.

Assigned score could be: 0 - outdated security patch, 1 - previous security patch, 2 - the latest security patch.

6) Device model: Device model along with an OS version and installed security patches can provide information about a device security in terms of available vulnerabilities.

Assigned score could be: 0 - unknown device model, 1 - a known device with known vulnerabilities or inaccurate sensors, 2 - a known device with accurate sensors and without known vulnerabilities.

7) Unknown sources of application: A user of a device can allow to install applications from unknown sources, for instance, not from the Google Play Store. Installing application from the unknown source may lead to a device contamination by various kinds of malware, including applications that can manipulate sensor data.

Assigned score could be: 0 - if unknown sources are allowed, 1 - otherwise.

8) Installed applications: Installed applications with signatures are used to evaluate potentially harmful software that can manipulate sensor data. Also, the fact that a well known application that is installed on a device has signature different from the same application at the Google Play Store, can serve as an evidence that an application was repackaged and can contain various modifications, including malicious components. A list of installed applications along with the signatures are transfered to the server for further analysis. This metric is used for the external metric calculating only.

9) Developer options menu: Enabled "developer option" menu in the settings allows to manipulate sensor data, for example, GPS data. However, if root access is gained on the device, a developer setting may have a very little influence on the overall data security evaluation.

Assigned score could be: 0 - if developer option is turned on, 1 - otherwise.

B. External metrics

This subsection describes metrics that may use external information for their calculation.

1) Installed applications rating: This metric combines several values. First of all, it takes into account what applications are installed on the device. Secondary, it compares signatures of the installed applications with the signatures of the same applications at the Google Play Store. If signatures match, we take in account a rating of an application.

TABLE II. MEASURED METRICS

Metric	Symbol	Group	Scale
Root Access	M_R	Internal	Boolean
Unlocked Bootloader	M_{ub}	Internal	Boolean
Device Lock	M_{dl}	Internal	From 0 to 2
Android OS Version	M_V	Internal	From 0 to 2
Device Model	M_M	Internal	From 0 to 2
Installed Security Patch	M_{sp}	Internal	From 0 to 2
Unknown Sources	M_{us}	Internal	Boolean
Installed Applications	M_{ia}	Internal	N/A
Developer Option Menu	M_{do}	Internal	Boolean
Installed Application Rating	M_{ar}	External	From 0 to 3
System Vulnerabilities	M_{sv}	External	From 0 to 9
Device Rating	M_{dr}	External	From 0 to 9

Assigned score may range from 0 to 3, where 0 - a known application for data manipulation is installed, 1 - an unknown third-party applications is installed, 2 - all applications are known, but signatures are different from those that at the Google Play Store, 3 - all applications are known and signatures match with those that at the Google Play Store.

2) System vulnerabilities: This metric takes into account three internal metrics: the Android OS version, a device model and versions of the installed security patches. To compute the value of this metric, an analysis of a device's system image is required.

Assigned score may range from 0 to 2, where 0 - System contains well known vulnerabilities, 1 - system does not contain well known vulnerabilities, but a device has inaccurate sensors, 2 - system does not contain well known vulnerabilities, and a device has accurate sensors.

3) Device rating: This is an external metric that does not depend on the internal metrics. This metric value depends on how reliable a device was in the past. Giving a fake data decreases rating.

Scale varies from 0 to 9, where 0 - if a device provided fake data more than 8 times, and 9 - if a device never provided fake data.

V. SECURITY EVALUATION DEFINITION AND EXAMPLES

A. Overall security evaluation

The general diagram of an overall security evaluation is presented in Fig. 4, and consists of two major parts: the Android part and the Remote part. The Android part is divided into two modules: an Android application and an Android service. First, a user launches an Android application to gather sensor's data. Then the application starts a service and sends a request for the internal security score and the metrics. The service calculates all required metrics, computes the internal score and passes it back to the application along with all the metrics. The application sends sensor data, the internal score and the metrics required for external metrics measurement to the remote part.

On the remote part, received data are stored for further processing. The external score is computed using received metrics. Overall Security Evaluation Score (OSES) is computed by fusing both the external and the internal scores. For computing the external score, the remote part may use external information, for example, a device rating, application ratings, vulnerabilities database, etc.

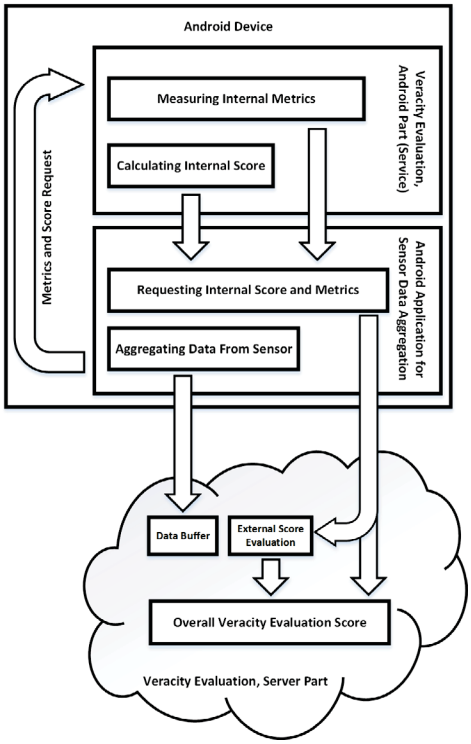

Fig. 4. General diagram of Overall Security Evaluation.

B. Overall security evaluation score

OSES represents likeliness of the sensor data manipulation. Score may range from 0 to 9, where 9 - sensor data have maximum trustworthiness and 0 for the lowest trustworthiness. However, **0** does not mean, that data has been manipulated, it means that data could be manipulated easily. Overall Security Evaluation score is presented by equation (1):

$$M_{veracity} = \frac{9}{13} \times (M_R + M_{ub} + M_{us} + M_{do} \times M_R + M_{dl} + \frac{1}{3} \times (M_{ar} + M_{sv} + M_{dr})) \quad (1)$$

where,

$$M_{sv} = \frac{3}{2} \times (M_V + M_M + M_{sp})$$

C. Overall security evaluation examples

As the first example, we consider Smartphone Google Nexus 6P with the latest Android v.7.1.1 with the latest security patch installed, turned off developer menu, without root access, locked bootloader, activated fingerprint unlock mechanism, disabled unknown sources, and installed applications are only from Google Play Store with original signatures. A User has just registered the device. In this case OSES equals:

$$M_{veracity} = \frac{9}{13} \times (1+1+1+1 \times 1+1+ + \frac{1}{3}(3+\frac{3}{2} \times (2+2+2)+9) = \frac{9}{13} \times 12 = 8.3$$

As the second example, we consider Smartphone Google Nexus 5 with the previous Android v.6.0.1 with the latest security patch installed, turned on developer menu, without root access, locked bootloader, without unlock mechanism, disabled unknown sources, and installed applications are only from Google Play Store with original signatures, but there is the application for faking GPS data, among installed applications. A user has just registered the device. In this case, OSES equals:

$$M_{veracity} = \frac{9}{13} \times (1+1+1+0 \times 1+0+ + \frac{1}{3}(0+\frac{3}{2} \times (1+2+2)+9) = \frac{9}{13} \times 8.5 = 5.9$$

Lower OSES in the second example does not mean, that data from the device has been manipulated. However, it represents the higher possibility that sensor data could be manipulated.

VI. Conclusion

Embedding numerous sensors into Android-based devices makes them a very convenient platform for research in different domains. However, due to complexity of Android-based systems and existing vulnerabilities, multiple scenarios of data manipulation and falsification are possible.

This paper investigates an Android OS architecture and its major security mechanisms that might have an influence on the trustworthiness of sensor originated data. Also, we investigated potential vulnerabilities that could arise due to the number of possible attacks and how they may affect the data trustworthiness.

The following metrics have been developed to evaluate the data security: root access vulnerability, unlocked bootloader vulnerability, device lock, Android OS version, security patch version, device model, unknown sources of applications, installed applications, developer options menu, installed applications rating, system vulnerabilities, device rating. Those metrics could be classified into the internal and the external groups, depending on the information used for their calculation.

We developed the method for metrics integration that computes Overall Security Evaluation Score representing overall data trustworthiness. Two test cases have been researched. The examples of OSES calculation have been presented and explained.

Acknowledgment

This research was supported in part by the National Science Foundation (award # ACI-1547301)

REFERENCES

[1] "IDC: Smartphone OS Market Share." [Online]. http://www.idc.com/promo/smartphone-market-share/os . Accessed: February 13, 2017.

[2] B. T. f. Ornithology, "BirdTrack." [Online]. https://play.google.com/store/apps/details?id=org.bto.btapp . Accessed: February 22, 2017.

[3] S. M. Communications, "Citizen Science (Unreleased)." [Online]. https://play.google.com/store/apps/details?id=com.sonymobile.activityathome . Accessed: February 22, 2017.

[4] "All vulnerabilities - Android Vulnerabilities." [Online]. http://androidvulnerabilities.org/all . Accessed: February 5, 2017.

[5] I. Khokhlov, L. Reznik, A. Kumar, A. Mookherjee, and R. Dalvi, "Data Security and Quality Evaluation Framework: Implementation Emperical Study on Android Devices," in *published in these proceedings*, 2017.

[6] "Dashboards | Android Developers." [Online]. https://developer.android.com/about/dashboards/index.html . Accessed: February 13, 2017.

[7] L. Stefanko, "Android trojan drops in, despite Googles Bouncer." [Online]. http://www.welivesecurity.com/2015/09/22/android-trojan-drops-in-despite-googles-bouncer/ . Accessed: February 18, 2017.

[8] M. Mohamed, B. Shrestha, and N. Saxena, "SMASheD: Sniffing and Manipulating Android Sensor Data for Offensive Purposes," *IEEE Transactions on Information Forensics and Security*, 2016. [Online]. http://ieeexplore.ieee.org/abstract/document/7605458/

[9] IncorporateApps, "Fake GPS Location Spoofer Free." [Online]. https://play.google.com/store/apps/details?id=com.incorporateapps.fakegps . Accessed: February 18, 2017.

[10] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: a survey of issues, malware penetration, and defenses," *IEEE communications surveys & tutorials*, vol. 17, no. 2, pp. 998–1022, 2015. [Online]. http://ieeexplore.ieee.org/abstract/document/6999911/

[11] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, "The impact of vendor customizations on android security," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 623–634. [Online]. http://dl.acm.org/citation.cfm?id=2516728

[12] LineageOS, "LineageOS LineageOS Android Distribution." [Online]. http://lineageos.org/ . Accessed: February 19, 2017.

[13] R. Weiss, L. Reznik, Y. Zhuang, A. Hoffman, D. Pollard, A. Rafetseder, T. Li, and J. Cappos, "Trust evaluation in mobile devices: An empirical study," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 25–32. [Online]. http://ieeexplore.ieee.org/abstract/document/7345261/

[14] "F.A.Q." [Online]. https://www.kingoapp.com/faq.htm . Accessed: February 20, 2017.

[15] "HTCdev - Unlock Bootloader." [Online]. http://www.htcdev.com/bootloader/ . Accessed: February 20, 2017.