Journal of King Saud University – Computer and Information Sciences 32 (2020) 543–552

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Android data storage security: A review

Haya Altuwaijri, Sanaa Ghouzali *

Department of Information Technology, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

ARTICLE INFO

Article history:
Received 3 February 2018
Revised 29 June 2018
Accepted 10 July 2018
Available online 19 July 2018

Keywords:
Android security
Android data storage
Physical threats
Software threats
Android encryption

ABSTRACT

The broad adoption of smartphones has superseded the desktop computers and laptops as a primary computing platform, due to mobility, constant connectivity and application diversity. Mobile devices encompass storage of extensive information including sensitive ones such as authentication credentials, pictures, videos, personal data, work information, and many more. Thus, securing data stored on mobile devices becomes a critical issue. In this review, we investigate the security of Android storage model between 2013 and 2018. Several threats are found in the literature that can be categorized as physical or software threats. Additionally, the existing solutions for each category are highlighted. Although Android provides valuable encryption systems including full disk encryption and keychain to enhance the data storage security, the encryption key, which is stored in the device, is still vulnerable to physical threats.

Contents

* Corresponding author.
E-mail address: sghouzali@ksu.edu.sa (S. Ghouzali).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2018.07.004
1319-1578/

1. Introduction

According to "Mobile Security Project" under "The Open Web Application Security Project (OWASP)" ("OWASP Mobile Security Project – OWASP", Owasp.org, 2016), insecure data storage is one of the leading top 10 security issues in smartphones since sensitive information can be revealed if it is not protected carefully. Unskilled Developers assume that access to the internal memory stores cannot be gained, but an attacker who accesses the device physically can attach the phone to a computer and retrieve sensitive personal information. Moreover, malware applications (apps) and legitimate vulnerable apps can be a threat to sensitive information leakage. A Trojan horse hidden behind legitimate app can perform imperceptible activities with the aim to steal user's information. Furthermore, attackers who want to access the data might exploit vulnerable apps, such as apps with over-privileged permissions.

Smartphone security has been subjected to intensive research work in the past years. Existing review papers presents the state of the art of popular smartphone operating systems (Khan et al., 2015) (Ahvanooey et al., 2017) (Zaidi et al., 2016). Khan et al. showed that mobile malware, similar to Personal Computer (PC) malware, can cause system corruption or reveal private user information (Khan et al., 2015). Also, they demonstrated vital points to ensure mobile security based on restricting malicious activities at several levels; application developer level, application's store level, and operating system level. Different types of mobile operating systems and software attacks have been discussed in (Ahvanooey et al., 2017). The authors provided a comprehensive overview of mobile threats, vulnerabilities, and countermeasures by reviewing published papers during the period 2011–2017. Authors in (Zaidi et al., 2016) studied smartphone security within 2010–2015. The authors categorized attacks into old and new attacks and presented possible solutions for each category. Moreover, they showed an estimation of mobile malware growth in 2020.

Android is the most used mobile operating system in the world that dominates the smartphone market with a share of 82.8% in 2015 ("IDC: Smartphone OS Market Share", www.idc.com, 2016). The literature exposes a set of threats to Android data storage that can be exploited to perform attacks. For example, an unreliable factory reset in some Android devices is a significant threat to the security and privacy of the stored data since the data is not erased correctly. Several techniques are used to protect data stored in a mobile device, mainly based on password-based data encryption. Authors in (Faruki et al., 2014) provided a survey of general Android security issues and defenses. They focused on Android malware growth within a specific timeline (2010–2013) and the suggested solution. They categorized the suggested solutions based on the goal, methodology to achieve this goal and deployment of the solution. They concluded that no single solution could efficiently address each issue. Rashidi et al. presented another survey that reviewed and discussed Android security threats and solutions between 2010 and 2015 (Faruki et al., 2014). They surveyed techniques (e.g., based on static and dynamic code analysis) to deal with malware on mobile devices and investigate strengths and weaknesses of each technique. Moreover, Sufatrio et al. introduced a survey on Android security and provided a taxonomy of mitigation solutions with five main categories based on app's deployment stages (app development, app availability in the market, app installation on a device, app execution on a device, and app security setting modification on a device) (Rashidi and Fung, 2015). They classified existing work into those five categories and thereby comparatively studied them and highlighted the limitation of each. Additionally, a general overview of Android security issues has been presented in (Sufatrio et al., 2015) without any categorization.

This paper aims to complement previous reviews about insecure data storage on Android smartphones by expanding the coverage of security threats and solutions. We believe an in-depth examination of Android data storage model is required. Therefore, we review Android attacks, threats, and their solutions over the period of 2013–2018. Additionally, we propose a distinctive categorization of Android data storage threats model based on physical and software threats and review a few works of each class. Moreover, mitigation solutions for each category are investigated.

The paper is organized as follows. Section 2 introduces the Android data storage model. Section 3 presents the Android data storage threat model. It contains two main subsections, which are, Physical threats and Software threats. Section 4 outlines the solutions found in the literature to enhance data storage security and emphasizes on Android encryption system as a critical feature for protecting data. Finally, the conclusion and future work are drawn in Section 5.

2. Android data storage model

Android is an open source mobile operating system that was initially developed by Android Inc. and financially backed and later bought by Google. In November 2007, the initial beta version of Android was released, then the first stable version 1.0 followed in September 2008. Android is primarily designed for touchscreen devices such as smartphones and tablets, and it is currently developed under Android Open Source Project (AOSP) that is promoted by the Open Handset Alliance (OHA), led by Google (Faruki et al., 2014). Android is based on the Linux Kernel, and its apps are written in Java. However, the native code and shared libraries are developed in C/C++ (Faruki et al., 2014). In the Android framework, there are unique storage alternatives, which as indicated by their access control components, can be divided into three classes: system, application-specific, and public as shown in Fig. 1. The system storage is the catalog where the entire Android OS is located and is secured by the Linux access control component. The application specific storage is a spot that is under control of a particular application and can only be read and written by that app, generally mounted on /data/. It hosts apps private directories that are commonly used to store sensitive information such as login credentials. The other storage alternative is shared public storage (internal primary SD card), that is mounted on /sdcard/ or /mnt/sdcard/, which

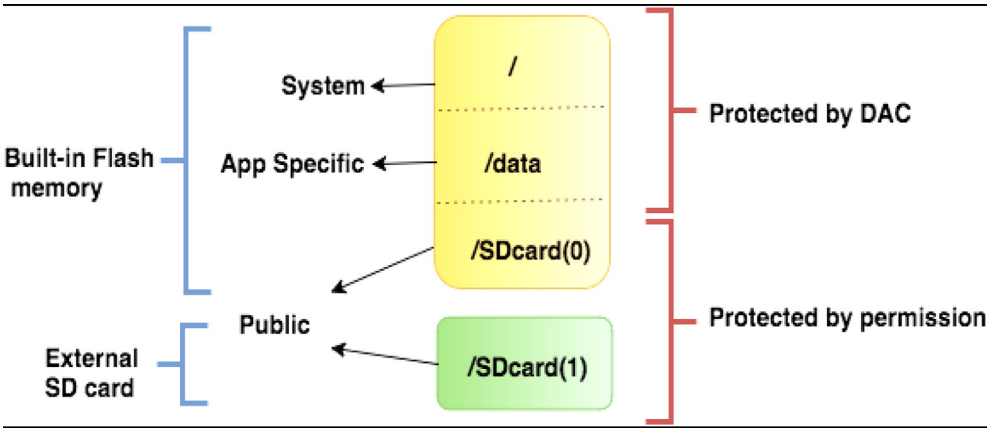

Fig. 1. Android storage model options and security.

is utilized to share information among applications. It is mainly used to store multimedia files made with the camera and microphone; besides, it is exposed to a computer connected via USB as downloaded documents, videos, images, and so on. Moreover, some Android devices contain external, removable SD card that is categorized under public shared storage (Mohini et al., 2013). It provides the same functionality as the internal SD card but can be physically removed and inserted by the user. It is called the secondary SD card. However, the primary and secondary SD cards are sometimes referred to as external storage (Mohini et al., 2013).

To protect these storage partitions, Android relies on Discretionary Access Control (DAC) mechanism provided by the underlying Linux file system to implement access control for the system and app-specific storage. For shared public storage, there is no fine-grained access control, it is protected by READ and WRITE permissions only, and therefore any app can read or write any folder in the shared public storage once it has acquired related permissions (Simon and Anderson, 2015). Android provides encryption algorithms that permit the developer to encrypt data in their applications to improve security. Android provides javax.-cipher package that offers the classes and interfaces for cryptographic applications implementing algorithms for encryption, decryption, and key agreement (Liu et al., 2015).

3. Android data storage threat model

This section demonstrates an assessment of Android Data storage threat model illustrated in Fig. 2. It identifies possible threats and vulnerabilities in Android Data storage, divided into physical threats and software threats. Flawed factory reset can be categorized in both software and physical threat.

3.1. Physical threats

Sets of researchers focus on Android smartphones physical threats since confidential data may exist in memory on mobile devices for a long time after being used. Thus, on the stolen device, the retrieval of sensitive information is possible and becomes a growing concern. The attacker who gains physical access to the device can quickly get the memory content ("javax.crypto—Android Developers," Developer.android.com, 2016). Researchers in (Xia et al., 2015) studied data exposure issue in Android to emphasize that it is a real problem. They analyzed 14 favorite Android apps and found that they could retrieve sensitive data (e.g., passwords) by dumping either Random Access Memory (RAM) or internal storage in 13 apps after 10 min.

More specifically, the literature exposes two types of attacks, Cold boot attack and Evil maid attack, which can be done against Android devices when the attacker gains physical access.

3.1.1. Cold boot attack

Cold boot attack depends on the fact that RAM retains data for a period after a computer is turned off. The amount of time relies on RAM chip's temperature, and it can be increased as much as the RAM is cold. Thus, the RAM can be re-plugged to another computer to get its content such as encryption keys. This attack can be made against all software-based encryption technologies. In (Tang et al., 2012) researchers examined if this type of attack can be done against Android Full disk encryption (FDE) (described in

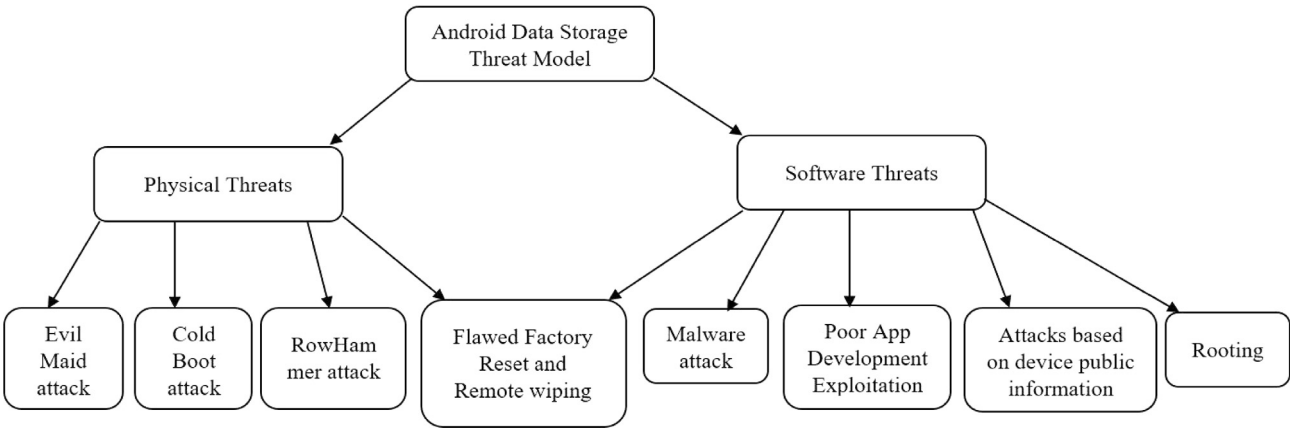

Fig. 2. Android data storage threat model.

Section 4.2.1.1). They built Frost tool, which is a recovery image tool. After gaining physical access to the smartphone, the tool is installed in the recovery partition. Using Frost, encryption keys from RAM can be retrieved and then break the FDE encryption. Frost requires an unlocked bootloader[1] to break the FDE since the unlocking process wipes all user data. Therefore, recovering encryption keys from RAM is not possible. However, Frost can also be used to take the memory of the phone and analyze it offline. Researchers were able to access recent emails, pictures, and browser's history from physical RAM dumps.

3.1.2. Evil maid attack

Any software-based encryption needs a part of the disk unencrypted; in Android, this is the entire system partition. A study in (Müller et al., 2013) proved that evil maid attacks could be made against smartphones because Android leaves the Master Boot Record and entire system partition unencrypted. In Evil maid attack, the attacker who has physical access can substitute the entire Android system with a modified Android that includes keystroke logging. Authors present an EvilDroid tool that shows that even with an encrypted device, the Android system partition can be altered with keylogging. The tool previews a modified PIN request interface that stores the PIN in unencrypted cache partition, thus, all data on the device can be accessed after acquiring the PIN. Researchers affirm that performing evil maid attacks against Android devices is possible. Additionally, the study shows that countermeasures are seemingly tricky. Manufacturers should consider this issue. The bootloaders need to be locked and wipe the user partition before unlocking.

3.1.3. RowHammer attack

RowHammer attacks rely on memory management weakness in the deep layers. It allows flipping memory bits by repeated access (hammers) memory cells causing unauthorized changes and corrupting sensitive memory regions (Götzfried and Müller, 2014). This complex vulnerability has been exploited in Android by implementing *Drammer*, a deterministic RowHammer-based attack (Götzfried and Müller, 2014). Authors showed that *Drammer* could obtain a privilege escalation to acquire root privileges and can be launched by any Android app with no special permission and without relying on any software vulnerability. Moreover, the authors highlighted that this type of attack could not be mitigated by current defenses.

3.2. Software threats

This section illustrates some software-based threats that include malware attack, poor application development, and rooting the device.

3.2.1. Malware attack

A study in (van der Veen et al., 2016) underlined the problem of stealing personal information from memory in Android devices. They analyzed 26 Android applications to check how these applications handle data in memory that is currently or has recently been manipulated. In the analysis, a full memory dump of the system is done both while using the application and after its termination, and the result showed that most of the applications analyzed keep sensitive and personal data in clear text within the memory. This analysis motivated the researchers to build Trojan horse that would hide behind a legitimate app that just let the user take pictures and share them, while in the background, it will monitor active processes, and dump the memory sectors when the process is running. The malware app was able to exploit vulnerabilities in the targeted apps and retrieve login credentials. The user behavior and the poor practices when developing mobile applications are the causes of introducing vulnerabilities. Moreover, the user did not notice this attack, both regarding visibility and mobile phone performance. This study presents a memory attack that doesn't require physical access.

3.2.2. Poor application development Exploitation

Developers influence the security of the data by building vulnerable apps such as building over-privileged applications, misuse of cryptographic application programming interface (API), and saving non-sensitive but non-shared data on public storage.

3.2.2.1. Over-privileged applications. Android permissions system is a way of ensuring a level of security and privacy by restricting application access to resources unless the user accepts the access permission. The app developers identify the required permission in the code, and when installing the app, there is no control if the permission is related to the app or not (Stirparo et al., 2013). Android permissions are criticized as coarse-grained (Tiwari et al., 2015), many applications gain more permissions than necessary; for example, if Flipkart shopping app requests internet permission, it can send and receive files from any other websites, not just Flipkart.com (Stirparo et al., 2013). Because of coarse-grained permissions, the poor practices of Android developers, incompetent permission administration, and insufficient permission documentation provided by Google, there are many apps with unnecessary permissions. These apps violate the principle of least privilege (Stirparo et al., 2013) and may be used to access sensitive resources. A systematic review of Android permissions issues and countermeasures have been done in (Tiwari et al., 2015). The study stated that the over-privileged applications is probably the most severe threat to Android security, which violates user privacy and security, and leads to personal information disclosure. The main reasons for building over-privileged apps are as following: developers may copy the code and apply it without understanding it, may request permissions that they thought it is mandatory and related to the functionalities they design but it is not (Tiwari et al., 2015).

3.2.2.2. Misuse of cryptographic API. Android Provides Cryptographic APIs that allow developers to secure sensitive data such as passwords and personal information on mobile devices. Researchers in (Fang et al., 2014) investigated whether developers correctly use these cryptographic APIs to achieve the required level of security. Following static analysis approach, researchers build a lightweight tool called "CryptoLint" to measure cryptographic misuse. This tool analyzes the application's code and checks for common flaws. A large-scale experiment was done by analyzing 11,748 Android applications and found that 88% of them misuse cryptography. The evaluation of applications was based on a set of defined rules that affect the required security, and if any application violates one of the rules, it is then considered as insecure. In the end, the final result shows that there are more than 10,000 vulnerable apps that are considered as secure apps since they are using cryptographic API, but actually, they are not using it appropriately.

3.2.2.3. Saving non-sensitive but non-shared data on public storage. General data can be classified into three groups: sensitive data such as passwords, public data to be shared with other apps such as date and time, and non-sensitive data not to be shared with others such as user's virtual-world identifiers, including the account name and profile photos. In Android, the first two types

[1] A bootloader is the first part that runs when booting Android device, and it handles instructions to boot the operating system.

gain significant care in its storage model. However, for the third type, Android encourages the developer to save this type of data in the public storage partition. This practice can be considered an enormous problem in Android's current storage model since all non-sensitive data will be saved in shared storage that any app can access. Moreover, the problem occurs if the developer could not identify all sensitive data, which leads to information leakage (Simon and Anderson, 2015). A study in (Simon and Anderson, 2015) investigated the occurrence of the mentioned problem in real applications, by choosing 17 favorite Android apps and checked whether they store the data correctly. The result showed that 13 apps leave user's private information on shared storage. Thus, it is possible to extract much information about the users that violate their privacy. The developer influences the security of user's data by identifying data group and then handling it correctly. The study suggests two solutions; first, use cryptographic API properly to encrypt data before saving it on public storage. Second, save data in the app-specific partition (/data) in the internal storage, since each app can access its partition only.

3.2.3. Attacks based on device public information

Device public information in Android refers to any information that can be accessed by apps after obtaining the required permission. This information includes Android_ID, Google account email, phone number, Wi-Fi MAC address, etc. and is used by the app to identify users (Egele et al., 2013). Currently, many apps rely on communication with a remote backend service to perform functions, which require authentication mechanism. A study in (Egele et al., 2013) investigated vulnerabilities of using device public information in apps for user authentication. Authors successfully exploited this vulnerability and performed identity-transfer attack since all the information needed to authenticate the user was available and could be accessed by any app. The offense consists of a malicious app which steals all device public information from victim's device and then sends it to the attacker device without any user interaction. Afterward, the attacker installs the vulnerable app, in which it will log into the victim's account directly since the needed information is available. This type of attack can be applied to any app that depends on device public information to authenticate users. Besides, the authors presented a comprehensive dynamic analysis to assess 1000 of the most favorite apps available in Google Play. The result shows that 41 of them was categorized as vulnerable. Two of them was the top-rated messaging apps, WhatsApp and Viber.

3.2.4. Rooting the device

The risk of insecure data storage increases more if the device has been "rooted". Rooting the device bypasses the confinements and limitations that are placed by the producer and gives the Android user privileged access to the device's subsystems to install unapproved (by Google) apps, to update the OS to the latest version of Android if the device is outdated and no longer updated by the manufacturer, to use custom themes, and so on (Bianchi et al., 2017). The result is that the device is left vulnerable to attacks and other security issues such as data leakage flaws (Kaspersky Lab, 2017). The process of gaining root access breaks most of the Android's security layers and requires to switch the device from Security-On to Security-Off (Casati and Visconti, 2018).

3.3. Flawed factory reset and remote wiping

Android provides built-in "Factory Reset" to erase data on the device; this can be done either remotely or by accessing the device directly. In the case of lost or stolen device, users can use "Android device manager" to find the device location, lock the device with passcode remotely and erase the data by performing a remote factory reset, which removes all data from the device except what is saved in the external storage (Poonguzhali et al., 2016).

In (Mohini et al., 2013) researchers analyzed the effectiveness of the factory reset in Android by studying 21 Android smartphones from 5 vendors with Android versions from v2.3.x to v4.3. They found that wiping the external storage can only be done if the user chooses the additional option "External storage" in factory reset setting in Android graphical user interface (GUI). However, if a user resets his device with Recovery/Bootloader instead of using the setting in GUI, external storage is not sanitized (see Fig. 3. Additionally, they discovered that all devices did not sanitize the external storage properly. The study concludes with a set of critical issues regarding factory reset, because of the flawed factory reset, Android v2.3.x does not provide proper deletion of the data partition. Besides, Android does not provide appropriate removal of the internal and external SD card in all versions. Furthermore, vendors push incomplete upgrades to the devices, and newer devices lack driver support for proper deletion that should be shipped by vendors, which could affect the factory reset process. Android full disk encryption has the potential to mitigate flawed factory reset problem. However, researchers found that encryption keys can be recovered because of flawed factory reset. To mitigate this issue, researchers recommend filling up the partition of interest with random-byte files, to overwrite all unallocated space or overwrite the entire partition "bit-by-bit" (Mohini et al., 2013).

To protect data in stolen devices, a user can use mobile antivirus (MAV) that provides "remote wipe" of the device and "remote lock". MAV requires Admin permission that should be enabled by the user on the setting, if it is not enabled, MAV cannot use the built-in wipe and lock features, also, cannot do reliable partition overwriting bit-by-bit to sanitize data storage. A study in ("Remotely ring, lock, or erase a lost device - Accounts Help", Support.google.com, 2016) examined top 10 MAV apps that provide remote wipe and lock functionalities to check their security practices and implementations. Starting by installing these apps on Android 2.3.5 device and then reviewing the apps' code and conducting a simple run-time analysis. The result showed that MAV might be defective due to poor implementation skills, Android API limitations and incorrect documentation that lead to misuse the API, also, vendor customizations that allow the attacker to bypass MAV protection even if its functions are appropriately implemented.

There are several issues regarding using MAV; there is incorrect information given to users after a remote wipe and lock, for example, even when the admin permission is disabled, MAV display "successful wiping" when the user requests to wipe. Additionally, using MAV web interface, users can access their data and perform remote lock and wipe; however, the authentication process is weak since there is no restriction on the password in most of MAV, for example, four letters as a password is accepted. Moreover, MAV relies on carrier network that might be insecure. The study deduces that MAV remote wipe functions cannot be used as an alternative to a flawed built-in Factory Reset.

4. Solutions for Android data storage threats

This section presents proposed solutions for the previously mentioned threats which can also be divided into physical and software solutions.

4.1. Physical threat solutions

A set of research papers present solutions to tackle physical threat problem in Android devices. CleanOS, TinMan, Sentry,

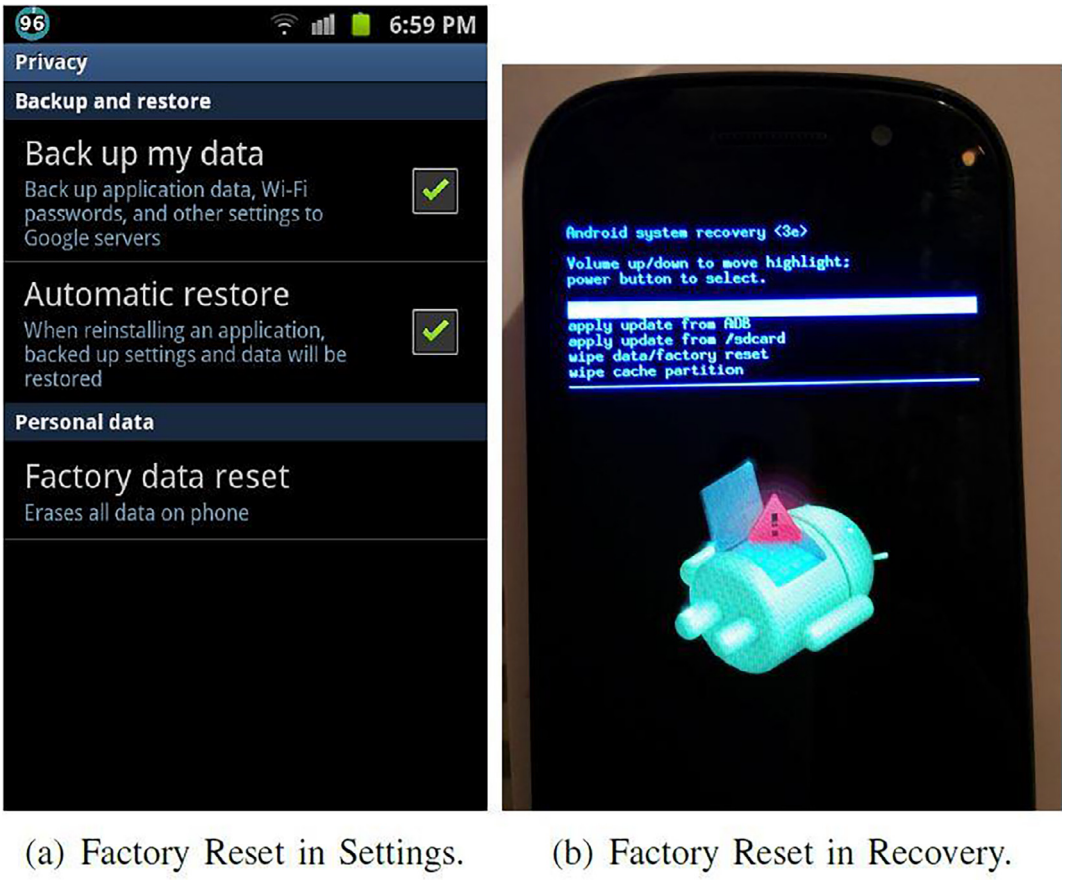

Fig. 3. Factory reset provided by most devices (Mohini et al., 2013).

Armored, Deadbolt, DroidVault, Replay Protected Memory Block (RPMB) partition, CATT, and ARMOR are different countermeasures in several papers. A summary of these countermeasures is given in Table 1.

4.1.1. CleanOS

In (Xia et al., 2015), researchers proposed a prototype of "CleanOS", which is an Android-based operating system that identifies sensitive data in RAM and internal storage that are unused for a specific amount of time, encrypt them, and then save the encrypted keys in the cloud. Thus, keep a clean environment in case of device theft. However, CleanOS does not protect data in use; it intends to minimize data exposure by protecting data that are not used for a period. Besides, it is vulnerable to network attack, where the attacker can sniff keys when they are sent to the cloud.

4.1.2. TinMan

In ("javax.crypto—Android Developers," Developer.android.com, 2016) researchers tried to mitigate data exposure issue and protect in-memory confidential data by proposing "TinMan" prototype system that uses offloading mechanism. It separates credentials access from the rest of the functionalities of the app and provides a trusted node to store those credentials. They focus on confidential data such as password, bank account, social security number, and credit card number, which are named confidential record (*cor*). TinMan separates *cor* from regular private data and enforces its protection (see Fig. 4). This mechanism aims to avoid storing sensitive data on the device, so when the device is lost or stolen, there is nothing to lose. The trusted node can be a server inside a company or a virtual machine on a trusted cloud, other than the device itself, so the applications need to access the trusted node to get the confidential data. However, this solution has significant challenges since confidential data residue in many places, and it is hard to identify them all. Also, it might degrade app's performance.

4.1.3. Sentry

Another solution to overcome the problem of data lodging in RAM was proposed in (Colp et al., 2015). The study focuses on securing the data only when the screen is locked because in unlocked state attacker can access the data using the user interface. The problem is when the device is locked, data remains in RAM for a while, which makes it vulnerable to physical attack. Hence for the proposed solution, is to encrypt memory pages of the sensitive application when the screen is locked and decrypt when the screen is unlocked. However, the main contribution is to avoid storing encryption/decryption keys in RAM and use ARM system-on-chip (SoC) instead. An ARM is an abbreviation for Advanced RISC (reduced instruction set computer) Machines Company which develops ARM processor based on the RISC architecture ("What is ARM processor – Definition from WhatIs.com", WhatIs.com, 2016). ARM SoC architecture is used by recent smartphones and tablets; it contains low capacity storage next to the CPU. The paper presents "Sentry" prototype system that uses SoC storage mechanisms to secure the cryptography keys, making physical attacks more difficult to mount because they must target the SoC to retrieve secrets, which is much more expensive. The main limitation of Sentry is that it leads to lower performance.

4.1.4. Armored

In (Müller et al., 2013) researchers provided a countermeasure against physical attacks where they built "Armored" that stores

Table 1
Summary of physical threat solutions.

Solution	Pros	Cons
CleanOS	Encrypt sensitive data and save the keys in the cloud.	Do not protect data in use. Vulnerable to network attack.
TinMan	Separate confidential data and save them in a trusted node.	Hard to identify confidential data which reside in many places. App's performance degradation.
Sentry	Secure the encryption keys by preventing saving them in the RAM.	Lead to lower performance.
Armored	Store the encryption keys and intermediate values of AES inside the CPU registers of the ARM microprocessor.	Impractical for end-users.
Deadbolt	Protect the FDE keys by securely overwriting the RAM.	Vulnerable to offline password guessing attack on the stored volume key.
Droidvault	Utilize TrustZone to manipulate the unencrypted data.	Difficult for applications to have a clear-cut line between secure and insecure data. TrustZone is processor-dependent and requires a firmware update in the existing devices.
RPMB	Enforce authentication of all read and write commands issued to the RPMB secure storage partition. Therefore, the memory replay attack can be prevented.	–
CATT	Protect existing vulnerable legacy systems against RowHammer attack without computational overhead.	–
ARMOR	Detect all the possible RowHammer errors with a high level of confidence and accuracy.	Impose performance overhead in case of a hammered-row in the system.

encryption keys and intermediate values of Advanced Encryption Standard (AES) inside CPU registers of the ARM microprocessor, without involving main memory. A proof-of-concept implementation was done as follows, 1) connect the phone to the computer via USB, 2) login as root, so it requires rooting the device, 3) write a sequence of key bits into the CPU registers, and 4) Run a cleanup procedure to erase all key residues from RAM. This implementation is impractical for end-users; researchers suggest making Android's password prompt repaired in a way that the encryption key is written directly into CPU registers.

4.1.5. Deadbolt

"Deadbolt" prototype application has been developed in (Skillen and Barrera, 2013) as another countermeasure against physical attack, and as a complement to Android lock-screen and FDE features. Since mobile phone is always-on and rarely shutdown, the FDE keys remain in memory even when the device is in locked screen state. Deadbolt protects the FDE keys while still providing essential mobile functionality by (upon lock screen) unmounting the encrypted data partition, and securely overwriting key stored RAM. The main limitation of Deadbolt is that it is still vulnerable to offline password guessing attack on the stored volume key.

4.1.6. Droidvault

DroidVault (Li et al., 2014) provides a secure data vault on Android devices using the TrustZone. TrustZone technology provides trusted hardware that allows for developing a diverse set of security services. Parallel to the OS, TrustZone is a separate environment that can run security function isolated from kernel OS by a hardware barrier (Zhao et al., 2014). Utilizing TrustZone, DroidVault presents a secure storage platform on Android since the data is stored in encrypted form on the filesystem and the unencrypted data is manipulated only in the TrustZone. DroidVault is a promising solution based on (Li et al., 2014) evaluation. However, it encompasses limitations. First, it is difficult for applications to have a clear-cut line between secure and insecure data. Second, TrustZone is processor-dependent, so it does not work for all the devices. Third, it requires a firmware update in the existing devices. Finally, the system has not been implemented on smartphones or tablets, but on a programmable board running the Android operative system.

4.1.7. Replay protected memory Block (RPMB) partition

In rooted Android, user area partition can be quickly erased or hacked using read/write system calls. Therefore, hackers can take backup of this partition and restore it to another phone to access data (Reddy et al., 2015). RPMB is a specific memory area in Embedded MultiMediaCard (eMMC). It is a secure storage partition, in which, all read and write commands issued to the RPMB must be authenticated. Authors in (Reddy et al., 2015) implemented this hardware-based mechanism and developed

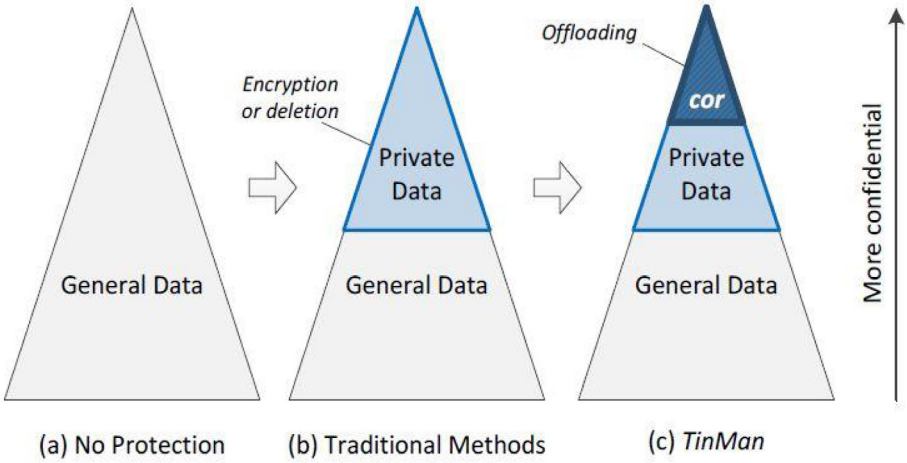

Fig. 4. Tinman separates cor and protects it by applying offloading mechanism ("javax.crypto—Android Developers," Developer.android.com, 2016).

RPMB driver interface to provide secure data storage against physical attacks.

4.1.8. CATT

CATT is the first practical software-based defense against RowHammer attack proposed in (Brasser et al., 2017). The researchers presented the design and implementation of this mitigation scheme. The main idea of CATT is not to prevent bit flips, but rather to limit the bit flips to memory pages that are already in the address space of the malicious application. Hence, strong isolation of user and kernel space is required to ensure that the attacker cannot exploit RowHammer to flip the bit in kernel memory. CATT solution has been successfully and efficiently applied to the Android operating system as it does not affect the run-time performance nor the stability of the system.

4.1.9. ARMOR

ARMOR, A Run-time Memory Hot-Row Detector, is a hardware defense against RowHammer attack developed in the University of Manchester – The school of Computer Science – (Ghasempour et al., 2015). ARMOR solution is based on monitoring number of activations for each row in the DRAM and detects which specific rows are at risk of being "hammered" at run-time. It can detect all the possible RowHammer errors with a high level of confidence and accuracy. Additionally, it is scalable and technology independent (Ghasempour et al., 2015).

4.2. Software threat solutions

Software threat solutions are mainly based on encryption schemes, which are one of the key security features for protecting data on a device. Android provides several encryption systems that can be used as a countermeasure for software-based threats.

4.2.1. Android encryption systems

There are two primary encryption systems provided by Android; Full Disk Encryption (FDE) and KeyChain.

4.2.1.1. Full disk encryption. Since Android 4 (released in October 2011), Android developers provide FDE that encrypts Android device to prevent data exposure and to protect users against data theft (Müller et al., 2013). However, Version 4.x does not necessarily indicate that the device supports encryption, there are specific 4.0 devices, which do not support encryption (Hruska, 2016).

FDE uses the user's password or PIN- lock screen to derive the encryption key through password-based key derivation function 2 (PBKDF2) to encrypt the device. Other lock screens, such as the pattern lock functionality cannot be used except in the latest version of Android 5 where PIN, Passwords, and Patterns can be used as encryption keys ("Full Disk Encryption | Android Open Source Project", Source.android.com, 2016). Thus, the power of the encryption depends on the strength of the passcode (Teufl et al., 2014). Moreover, requiring the user to type the encryption password every time he wants to unlock the screen, makes him even less likely to choose a secure password. Also, there is an increased chance that an attacker is shoulder surfing while user unlocks screen, which will disclose encryption password as well. Also, FDE is a lengthy process that makes the device unusable during that time, depending on the device capabilities; FDE may degrade the device performance. Once it is enabled, it cannot be reversed without a factory reset of the device. If the user forgets the key, there is no way to get the data back (Bianchi et al., 2017).

This encryption feature is disabled by default, and the user must activate it manually, which raises a security risk. Furthermore, it can only be activated if PIN- locks or passwords are in use (Müller et al., 2013). FDE does not encrypt whole disks, it encrypts user partitions mounted at (/data) only, which host app's private directory. Each app has its own specific (/data) that it is hidden from users and cannot be read or written by any other app. It is usually used to store sensitive data such as configuration files, saved games data, or any other types of files that the user should not tamper with ("Remotely ring, lock, or erase a lost device – Accounts Help", Support.google.com, 2016); ("Storing Application Data", Google Developers, , 2016). Moreover, FDE never applies for external storage (Bianchi et al., 2017).

However, in a recent release of Android 6.0 (October 2015), the full disk encryption is mandatory by default (Hruska, 2016). It encrypts the application's private data (/data partition), and the application's shared storage partition (/sdcard partition) also, which is the internal non-removable SD card ("Android 6.0 Compatibility Definition", 2016). It is stated in Android 6 Compatibility Definition that the encryption must use AES with a key of 128-bits (or greater) and the key should be encrypted with lock screen passcode before being stored. "The encryption key SHOULD be AES encrypted with the lock screen passcode stretched using a slow stretching algorithm (e.g., PBKDF2 or scrypt). If the user has not specified a lock screen passcode or has disabled use of the passcode for encryption, the system SHOULD use a default passcode to wrap the encryption key" ("Android 6.0 Compatibility Definition", 2016).

Overall, FDE depends on the Android version and device manufacturer, since each manufacturer supplies their devices with customized versions of the Android OS (Teufl et al., 2014). Table 2 demonstrates different FDE features based on Android version and possible threats. None of these versions apply FDE to external storage.

4.2.1.2. KeyChain. Android version 4.0 introduces KeyChain API that allows developers to store user's credentials used in an application securely. It is separate from FDE since it can be used even when the FDE system is not activated. This method of encryption depends on the developer's decision in using it for their applications to store user's credentials. Poor developer choices can lead to serious security problems. Moreover, if the Android system is already encrypted via FDE, there is no additional protection offered by the Android KeyChain. As in FDE, the user's passcode is utilized to derive the cryptographic key that is used to encrypt a master key, as this means that the user influences the security of the KeyChain system by choosing the passcode for locking the Android device. KeyChain uses the same passcode used in screen lock as in FDE, and thus, if the passcode for the FDE system has already been determined by applying brute-force attack, there is no need to apply another attack for the KeyChain (Teufl et al., 2014).

4.2.2. Other solutions

Based on Android encryption systems issues discussed above, Android secure storage application has been presented in (Xia et al., 2015). It prompts the user to store their data locally on the device in a secure way without an extra requirement. The proposed system provides an option to the user for encrypting specific files rather than full encryption. Moreover, the files can be saved in the unencrypted format without doing the factory reset as in FDE. Users can choose to store files in the secure storage; then these files will be encrypted using Password-Based Encryption Standard mechanism. The mechanism generates two keys; a master key that is generated from the user's password and content protection key that is used to encrypt sensitive data, and it is encrypted by the master key. The main drawback of the proposed app is that it depends on the user's password that can be stolen or forgotten. In contrast to passwords, user's biometrics cannot be easily forged, duplicated, or shared. Google published Android 6.0 (released in October 2015) that includes Fingerprint API, which

Table 2
FDE features and threats.

Android version - Release date	Encrypted partition	Encryption Key	Optional or Mandatory	Limitations and Threats
Android 4 –Oct. 2011	Application partition (/data)	PIN and Password	Optional	- Encryption must be activated manually. - The security of the system depends on the strength of the passcode. - Susceptible to external brute-force attacks.
Android 5 –Oct. 2014	Application partition (/data)	PIN, Password, and Patterns	Optional	- Encryption must be activated manually. - The security of the system depends on the strength of the passcode. - Susceptible to external brute-force attacks.
Android 6 –Oct. 2015	Application partition (/data) and internal non-removable SD card partition (/sdcard)	PIN, Password, and Patterns	Mandatory	- The security of the system depends on the strength of the passcode. - Susceptible to external brute-force attacks.

can be employed by developers to include fingerprint recognition in their apps. Goode Intelligence predicts that 3.4 billion users will use biometric systems on their mobile devices by 2018 (Yldrm and Varol, 2014). Mobile fingerprint technology has been used in different mobile applications that mainly focus on authenticating users, such as phone lock, mobile payment, and mobile banking (Yldrm and Varol, 2014) (Gao et al., 2014). Fingerprint in mobile devices encompasses several advantages; it is a fast authentication model, it is entirely accepted in mobile devices, and it provides competitive performance rate (Avila et al., 2014). However, deploying biometrics in mobile devices faces several challenges such as limited computation power, small storage capacity, and finite battery life. The algorithm used in traditional biometrics systems needs to be simplified to adapt to small CPU processing power of mobile phones. The essential simplicity might reduce the accuracy and security level, which will affect the performance of mobile-enabled biometric techniques (Wang et al., 2011). Besides, authors in (Bianchi et al., 2018) have shown how the inappropriate usage patterns of the fingerprint API in most Android apps such as Google Play Store and Square Cash can make these apps vulnerable to several attacks. Another challenge is how to maintain safe storage of biometrics data in the device. Growing biometrics databases carry with them expanding worries about the likelihood of theft or misuse of individuals' biometrics. To preserve the privacy of biometric technology users, biometric data must be efficiently secured during storage in a database (Krivokuca, 2015).

Other security solutions for saving sensitive data on Android mobiles are cloud-based or remote server-based which require a working network, as this makes the accessibility of the stored contents depends on the network. Also, there is a possibility of user data getting disclosed, if the cloud storage or the remote server database gets compromised (Bianchi et al., 2017).

4.3. Summary

Table 3 summarizes the Android data storage threats and solutions presented in this paper.

Table 3
Summary of Android Data Storage Threats and Solutions.

	Attack	Solution
Physical threats	Cold boot attack	Clean OS, TinMan, Sentry, Armored, Deadbolt, DroidVault.
	Evil maid attack	–
	RowHammer attack	CATT, ARMOR
Software threats	Flawed Factory Reset and Remote Wiping	–
	Malware attack	Android Encryption
	Attacks based on device public information	Android Encryption
	Rooting the Device	–
	Poor Application Development Exploitation	Android Encryption

5. Conclusion

The literature exposes a set of identified threats on Android data storage, along with solutions to mitigate the risk and improve the security. We can deduce that users can influence the security of the data either by deciding to root the device, which is a risky process, or by disabling the FDE option. Besides, the developer influences the security of the data by building legitimate vulnerable apps, or by building malware apps that can access sensitive data. Cryptography is the primary defense against data disclosure; it is highly recommended to encrypt data on the device. Android provides two types of encryption systems, FDE and KeyChain that use password-based encryption method and depend on user's passcode. Choosing a strong lock screen passcode is a critical issue, but requiring the user to type encryption password every time to unlock the screen, renders it unlikely to choose a secure password. Moreover, security of conventional cryptographic techniques relies on the assumption that only a legitimate user knows the cryptographic keys, hence, maintaining the secrecy of keys is a big challenge. In the password-based encryption method, the key is vulnerable to offline brute force attack when the encrypted key stored in the device (Kaspersky Lab, 2017). An attacker who gains physical possession of the smartphone can extract the data and mount an offline attack, trying passcodes until one is found that produces a key, which decrypts the data successfully (Bianchi et al., 2017). Moreover, it is dangerous to depend on user's passwords since it can be easily lost, stolen, forgotten, or guessed. Thus, how to protect data on mobile devices against software and physical attacks, is still a significant and urgent problem. A promising solution would be to use biometric cryptosystem to protect the storage of the encryption key in the device. Biometric cryptosystem substitutes password-based encryption method; the user is not prompting to enter a password, s/he is requested to present his/her biometric template instead. It can be used to secure data against software attacks since data is encrypted, and against physical attacks because the cryptographic key is never stored in the device, only an auxiliary data will be stored (Kanade et al., 2012).

As a future work, we will investigate the threats and vulnerabilities of using biometrics for authentication and data encryption on Android. Besides, Android is continuously updated, and new versions are published endlessly. New versions might prevent some of the vulnerabilities discussed in this survey and open others. Thus, Android security and its threats should continuously be investigated.

References

"Android 6.0 Compatibility Definition", 2016. [Online]. Available: https://static.googleusercontent.com/media/source.android.com/en//compatibility/android-cdd.pdf (Accessed: 11.11.16).

"Full Disk Encryption | Android Open Source Project", Source.android.com, 2016. [Online]. Available: https://source.android.com/security/encryption/ (accessed: 13.11.15).

"IDC: Smartphone OS Market Share", www.idc.com, 2016. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp (accessed: 11.11.15).
"OWASP Mobile Security Project - OWASP", Owasp.org, 2016. [Online]. Available: https://www.owasp.org/index.php/Mobile#tab=Top_10_Mobile_Risks (accessed: 02.02.16).
"Remotely ring, lock, or erase a lost device - Accounts Help", Support.google.com, 2016. [Online]. Available: https://support.google.com/accounts/answer/6160500?hl=en (Accessed: 21.01.16).
"Storing Application Data", Google Developers, Online Available: https://developers.google.com/drive/web/appdata 2016 (accessed: 13.11.16).
"What is ARM processor? - Definition from WhatIs.com", WhatIs.com, 2016. [Online]. Available: http://whatis.techtarget.com/definition/ARM-processor (accessed: 17.02.16).
"javax.crypto—Android Developers," Developer.android.com, 2016. [Online]. Available: http://developer.android.com/reference/javax/crypto/packagesummary.html (accessed: 23.02.16).
Ahvanooey, M., Li, Q., Rabbani, M., Rajput, A., 2017. A survey on smartphones security: software vulnerabilities, malware, and attacks. Int. J. Adv. Comp. Sci. Appl. 8 (10), 30–45.
Avila, C., Casanova, J., Baallestros, F., Garcia, L., Gomex, M., Sierra, D., Pozo, G., 2014. State of the art of mobile biometrics, liveness and non-coercion detection. In: Project FP7-610713 (PCAS), Public Deliverable, The European Union's Seventh Framework Programme for research, technological development and demonstration.
Bianchi, A., Gustafson, E., Fratantonio, Y., Kruegel, C., Vigna, G., 2017 Exploitation and mitigation of authentication schemes based on device-public information. In: Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC'17), Orlando, FL, USA, pp. 16–27.
Bianchi, A., Fratantonio, Y., Machiry, A., Kruegel, C., Vigna, G., Chung, P., Lee, W., Broken fingers: on the usage of the fingerprint API in android. In: Proceedings of Network and Distributed System Security Symposium (NDSS'18), San Diego, CA, USA.
Brasser, F., Davi, L., Gens, D., Liebchen, C., Sadeghi, A., 2017. CAn't touch this: software-only mitigation against Rowhammer attacks targeting kernel memory. In: Proceedings of the 26th USENIX Security Symposium, Vancouver, BC, Canada.
Casati, L., Visconti, A., 2018. The dangers of rooting: data leakage detection in android applications. Mobile Inf. Syst. 2018, 6020461.
Colp, P., Zhang, J., Gleeson, J., Suneja, S., de Lara, E., Raj, H., Saroiu, S., Wolman, A., 2015. Protecting data on smartphones and tablets from memory attacks. In: Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems, Istanbul, Turkey, pp. 177–189.
Egele, M., Brumley, D., Fratantonio, Y., Kruegel, C., 2013. An empirical study of cryptographic misuse in android applications. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13), Berlin, Germany, pp. 73–84.
Fang, Z., Han, W., Li, Y., 2014. Permission based Android security: issues and countermeasures. Comput. Secur. 43, 205–218.
Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M., Conti, M., Rajarajan, M., 2014. Android security: a survey of issues, malware penetration and defenses. Commun. Surveys Tutorials 17 (2), 998–1022.
Gao, M., Hu, X., Cao, B., Li, D., 2014. Fingerprint sensors in mobile devices. In: Proceedings of the 9th IEEE Conference on Industrial Electronics and Applications, Hangzhou, China, pp. 1437–1440.
M. Ghasempour, M. Lujan, J. Garside, "ARMOR: A Run-time Memory Hot-Row Detector," 2015. [Online]. Available: http://apt.cs.manchester.ac.uk/projects/ARMOR/RowHammer/ (accessed: 29.06.18).
Götzfried, J., Müller, T., 2014. Analysing android's full disk encryption feature. J. Wireless Mobile Networks, Ubiquitous Comput. Dependable Appl. (JoWUA) 5 (1), 84–100.
J. Hruska, "Android 6.0 Marshmallow makes full-disk encryption mandatory for most new devices | ExtremeTech", ExtremeTech, 2016. [Online]. Available: http://www.extremetech.com/mobile/216560-android-6-0-marshmallow-makes-full-disk-encryption-mandatory-for-most-new-devices (accessed: 28.01.16).
Kanade, S., Petrovska-Delacrtaz, D., Dorizzi, B., 2012. Enhancing information security and privacy by combining biometrics with cryptography. In: Synthesis Lectures on Information Security, Privacy, and Trust, pp. 1–140.
Kaspersky Lab "Rooting your Android: advantages, disadvantages, and snags" 2017, Online Available: https://www.kaspersky.com/blog/android-root-faq/17135/ (accessed: 25.04.18).
Khan, J., Abbas, H., AlMuhtadi, J., 2015. Survey on mobile user's data privacy threats and defense mechanisms. Procedia Comp. Sci. 56, 376–383.
Krivokuca, V., 2015. Fingerprint Template Protection using Compact Minutiae Patterns Ph.D. thesis. The University of Auckland, New Zealand.
Li, X., Hu, H., Bai, G., Jia, Y., Liang Z., Saxena, P., 2014. DroidVault: A trusted data vault for android devices. In: Proceedings of the 19th International Conference on Engineering of Complex Computer Systems, Tianjin, China, pp. 29–38.
Liu, X., Zhou, Z., Diao, W., Li, Z., Zhang, K., 2015. An empirical study on android for saving non-shared data on public storage. In: Federrath, H., Gollmann, D. (Eds.), ICT Systems Security and Privacy Protection, IFIP Advances in Information and Communication Technology. Springer, Cham, pp. 542–556.
Mohini, T., Kumar, S., Nitesh, G., 2013. Review on Android and Smartphone Security. Res. J. Comput. Inf. Technol. Sci. 1 (6), 12–19.
Müller, T., Spreitzenbarth, M., 2013. FROST. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (Eds.), Applied Cryptography and Network Security (ACNS 2013), Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 373–388.
Poonguzhali, P., Dhanokar, P., Chaithanya, M., Patil, M.U., 2016. Secure storage of data on android based devices. Int. J. Eng. Technol. (IJET) 8, 177–182.
Rashidi, B., Fung, C., 2015. A survey of android security threats and defenses. J. Wireless Mobile Networks, Ubiquitous Comput. Dependable Appl. (JoWUA) 6 (3), 3–35.
Reddy, A.K., Paramasivam, P., Vemula, P.B., 2015. Mobile secure data protection using eMMC RPMB partition, In: Proceedings of the International Conference on Computing and Network Communications (CoCoNet), Trivandrum, India, pp. 946–950.
Simon, L., Anderson, R., 2015. Security analysis of android factory resets. In: Proceedings of the 3rd Mobile Security Technologies Workshop (MoST), San Jose, CA, USA.
Skillen, A., Barrera, D., van Oorschot, P., 2013. Deadbolt: locking down android disk encryption. In: Proceedings of the 3rd ACM workshop on Security and privacy in smartphones & mobile devices (SPSM'13), Berlin, Germany, pp. 3–14.
Stirparo, P., Fovino, I.N., Taddeo, M., Kounelis, I., 2013. In-memory credentials robbery on android phones. In: Proceedings of 2013 World Congress on Internet Security (WorldCIS), London, UK, pp. 88–93.
Sufatrio, D.J.J., Tan, T.-W., Chua, V.L.L., 2015. Thing, "Securing android: a survey, taxonomy, and challenges. ACM Comput. Surveys 47, (4) 58.
Tang, Y., Ames, P., Bhamidipati, S., Bijlani, A., Geambasu, R., Sarda, N., 2012. Cleanos: Limiting Mobile Data Exposure with Idle Eviction. Hollywood, CA, USA, pp. 77–91.
Teufl, P., Fitzek, A., Hein, D., Marsalek, A., Oprisnik, A., Zefferer, T., 2014. Android encryption systems. In: Proceedings of the International Conference on Privacy and Security in Mobile Systems (PRISMS), Aalborg, Denmark, pp. 1–8.
Tiwari, P., Singh, U., 2015. Android users security via permission based analysis. In: Abawajy, J., Mukherjea, S., Thampi, S., Ruiz-Martínez, A. (Eds.), Security in Computing and Communications (SSCC 2015), Communications in Computer and Information Science. Springer, Cham, pp. 496–505.
van der Veen, V., Fratantonio, Y., Lindorfer, M., Gruss, D., Maurice, C., Vigna, G., Bos, H., Razavi, K., Giuffrida, C., 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. Vienna, Austria, pp. 1675–1689.
Wang, S., Liu, J., 2011. Biometrics on mobile phone. In: Yang, J. (Ed.), Recent Application in Biometrics. IntechOpen, Rijeka, Croatia, pp. 3–22.
Xia, Y., Liu, Y., Tan, C., Ma, M., Guan, H., Zang, B., Chen, H., 2015. TinMan: Eliminating Confidential Mobile Data Exposure with Security Oriented Offloading. Bordeaux, France.
Yldrm, N., Varol, A., 2014. Mobile biometric security systems for today and future. In: Proceedings of the 2nd International Symposium on Digital Forensics and Security (ISDFS'14), Houston, TX, USA, pp. 86–91.
Zaidi, S., Shah, M., Kamran, M., Javiad, Q., Zhang, S., 2016. A survey on security for smartphone device. Int. J. Adv. Comp. Sci. Appl. 7 (4), 206–219.
Zhao, S., Zhang, Q., Hu, G., Qin, Y., Feng, D., 2014. Providing root of trust for ARM TrustZone using on-chip SRAM. In: Proceedings of the 4th International Workshop on Trustworthy Embedded Devices (TrustED'14), Scottsdale, AZ, USA, pp. 25–36.