University of Amsterdam

MSc System and Network Engineering

Research Project 2

Using Sensitive Information on Android 6 Based Smartphones

Author:
Romke van Dijk
romke.vandijk@os3.nl

Supervisors:
Jordi van den Breekel
vandenbreekel.jordi@kpmg.nl
Xenia Economidou
economidou.xenia@kpmg.nl
Ruud Verbij
verbij.ruud@kpmg.nl

August 5, 2016

Abstract

Smartphones are used in today's enterprises for sending e-mails, storing documents and accessing data remotely. This exposes sensitive information on these smartphones. Android is the most implemented operating system for smartphones and sensitive information should be protected on those devices. In this research we investigated to what extent sensitive information is protected on Android based smartphones by studying already existing literature. We showed that multiple areas put sensitive information at risk. One of those areas is the data encryption, the encryption keys remain in memory when the device is locked. If an attacker is able bypass the lockscreen authentication, the attacker gets access to all the sensitive information. Also there are security concerns due to trade-off between usability and security for lockscreen authentication. If a weak authentication method is used, the sensitive information might be at risk. When a more complex method is used, the smartphone can become unusable. When selecting a smartphone, a device owner should make sure that the manufacturer is committed to distribute security updates in a reasonable time, otherwise the sensitive information might be at risk. As for now it is also impossible to detect a compromised kernel at run time. Finally a mobile device management solution should be used to manage the smartphones, used to enforce security policies on the smartphones and used to collect information about the smartphones. Although the current information collection is limited and can be significantly enhanced. Besides investigating possible weaknesses, this research also presents possible solutions.

Contents

1. Introduction

Smartphones have become an indispensable part of today's enterprises. Users bring their own smartphones inside corporate networks and enterprises distribute smartphones to their employees as business phones. Using smartphones in organisations introduces new security risks no matter which deployment schema is used. Employees use smartphones for sending and receiving e-mails, storing documents or remotely accessing data, thus exposing sensitive information.

Chief Information Security Officers (CISO) are challenged to mitigate those security risks. The risks are mitigated by designing and implementing security policies. Such policies include for example, password length or enforcing transmission of data over secure channels. Multiple organisations provide guidelines for CISOs to help them design these policies. Examples of those organisations are the National Institute of Standards and Technology (NIST) and the Communications Electronic Security Group (CESG). The NIST policies are generic, discussing the risks of using smartphones in organisations. The CESG guidelines explain which security settings can mitigate which risks on operating system level.

Operating System	Market share
Android	84,1%
iOS	14,8%
Windows Phone	0,7%
BlackBerry OS	0,2%
Others	0,2%

Table 1.1: Smartphone OS market share in Q1 of 2016 [Sta16]

It happens that technical requirements force an organisation to use a specific operating system. This operating system could be the Android Operating System (OS). Also based on the market share of smartphone operating systems [Sta16](as can be seen in Table 1.1) it is likely that an organisation comes into contact with the Android OS.

Every operating system has its own strengths and weaknesses. If an organisation wants to use sensitive information on an Android based smartphone, it is important that security features are configured to mitigate possible risks as much as possible. The already mentioned guidelines do not explain to what extent sensitive information is protected. The goal of the

CESG guidelines is only to inform the device owner about which security settings mitigate specific risks. The explanation of how, why and to what extent those security settings mitigate those risks is not the goal of those guidelines.

In this research we will investigate the requirements for using sensitive information on Android based smartphones, the security features of the Android OS, possible weaknesses of those security features and to what extent sensitive information is protected on an Android based smartphone. We also show how the security features can be improved. Already existing literature will be used to investigate the requirements, the security features or the possible weaknesses. After reading this research we want our reader to understand to what extent sensitive information is protected on Android based smartphones and which areas require improvements.

1.1 Research Questions

This research focusses on how effective the security mechanisms are for storing sensitive information on the Android OS, but not on the phone's hardware. This will be done by answering the following research question:

To what extent is sensitive information protected on Android 6 based smartphones?

To help answering the main research question the following subquestions need to be answered:

- RQ1: What are the requirements for storing sensitive information on Android based smartphones?
- RQ2: What security mechanisms does the Android OS implement to comply with those requirements?
- RQ3: Which new security features are implemented in the Android OS version 6?
- RQ4: How can the security features of the Android OS be improved?

The Android OS is constantly evolving and research is quickly outdated. Some of the available research does not clearly address which Android OS version has been researched. Our research does clarify this matter by stressing which new security features have been added in the most recent version, Android version 6. This is the reason why RQ3 has been added additionally to RQ2.

Sensitive information is a core concept of this research, but what exactly is meant with sensitive information? This will be discussed in the next section.

1.2 Sensitive Information

In this research sensitive information refers to the majority of information processed (or created) by large enterprises or public services that are used

in routine business operations and services and could have damaging consequences if lost, stolen or published in the media [CES14]. Examples are [Wik16e];

- personal and private information (such as medical records), which is required to be protected by law;
- confidential business information (such as information protected by a Non-Disclosure Agreement (NDA)).

The next section will describe to the attack landscape and thus explain in which cases sensitive information should be protected.

1.3 Attack landscape

Smartphones are highly mobile and therefore they introduce security risks. They lack physical security controls because they can be used outside of an organization's control, for example in restaurants or hotels. They are much more likely to get stolen than other devices because of their small form factor. They primarily use untrusted networks for internet access, for example the cellular network. Compared to laptops which use trusted networks more often, for example the corporate network. Mobile operating systems running on smartphones make it extremely easy to install and run applications. Also smartphones interact with a lot of different systems, for example synchronization with a personal computer or with cloud services [SS14]. Smartphone users tend to use weak authentication methods or disable authentication altogether, which also potentially puts sensitive information at risk [Dev16a]. Compared to for example a laptop, where a keyboard is available which making it more convenient to enter complex passwords. These examples show that mobile devices have a different attack landscape then for example a personal computer.

NIST [SS14] and USCert [RF11] helped translating those risks in a well defined attack landscape. Based on the definition of [Off14], sensitive information should be protected against attackers with bounded capabilities and resources. Examples of those attackers may include, investigative journalists, competent individual hackers, or the majority of criminal individuals and groups [Off14].

Examples of possible attacks performed by those attackers to which sensitive information should be protected [RF11][SS14]:

- device theft, when a smartphone gets stolen sensitive information stored on the smartphone should not be available to the attacker.
- malicious applications stealing sensitive information. Applications should be able to access information that they do not required. For example, a simple flash light application should not be able to access calendar data.

- exploiting vulnerabilities (both known and unknown) to execute code on the system. It should not be possible for the bounded attacker to make the phone do something which it is not supposed to do for example disabling a security mechanism.
- eavesdropping on communication, it should not be possible for the bounded attacker to access data that is sent to and from the smartphone. Even when the user of the smartphone is accessing sensitive information through an insecure WiFi network (for example a free insecure WiFi network in a hotel), the sensitive data should still be protected.

Both the attack landscape and the definition of sensitive information are important concepts used through this research. The next section will describe the structure of the rest of this report.

1.4 Structure

The rest of this paper is structured as follows: this research starts with its related work and methodology in Section 2 and 3. Section 4 will answer *RQ1* thus describes the requirements for storing sensitive information. Section 5 will answer *RQ2* and *RQ3*, thus elaborate on the security features of Android version 6 and describe its weaknesses. Section 6 will compare the answers of those of *RQ1* with *RQ2* and *RQ3* in order to answer the main research question. Using this information possible improvements were found. Those improvements are described in Section 7 (answering *RQ4*). Finally, we end this report with a discussion, conclusion and suggest future work in Sections 8, 9 and 10, respectively.

2. Related Work

As described in Section 1, there are guidelines available from CESG [CES15] [CES16] and NIST [JS08] [SS14] for processing sensitive information on mobile devices including Android based smartphones. The CESG guidelines describe which settings need to be enabled to protect sensitive information. The guidelines also mention non-mitigated risks.

The NIST guidelines address security measures for smartphones and PDAs, but do not address specific security features of operating systems. They focus on the risks of introducing smartphones in an organization. They also suggest mitigating measures for those risks. CESG also published a comparable document [CES13], both documents were used in this research to determine the requirements for storing sensitive information.

The security features of Android have been studied, but the Android OS is continuously being extended. Some examples of Android security research are [Cue+15], [Heu+14], [IG14], [Kha+12] and [Bug+11]. The most up-to-date description of the security features of Android is the Android project page [Pro16][Goo15b][Inc15a][Pro10][Goo15a][Inc15b]. This page describes the security features of the Android OS, but it is oriented towards developers. It does not provide guidelines for storing sensitive information. Some additional sources have been used when the project page was insufficient. One of those sources has been [IG14]. [IG14] provides a more detailed explanation of the security features of the Android OS.

An interesting case study is the Samsung KNOX secure container [Sam16]. The KNOX container can be used to store sensitive information inside an application. The KNOX container has been researched in [KW16]. This case study can be used to determine weaknesses in the security features of the Android OS and research possible mitigations to those weaknesses. Other sources that have been used for the same purpose are [MS13], [Ber12], [Uel+13], [DM12], [Rog13], [OGo03], [Tes16], [Dav+12] and [Fox15].

Another interesting case study is Apple's iOS, a technical description of its security can be found in [App15]. The security features of iOS and Android are compared in [MP15] and [Qer+14]. These papers indicate that iOS offers more advanced security mechanisms than Android, but they do not suggest improvements for Android. Both papers were published before the release of version 6 of the Android OS.

3. Methodology

This section describes the method used to answer the research questions.

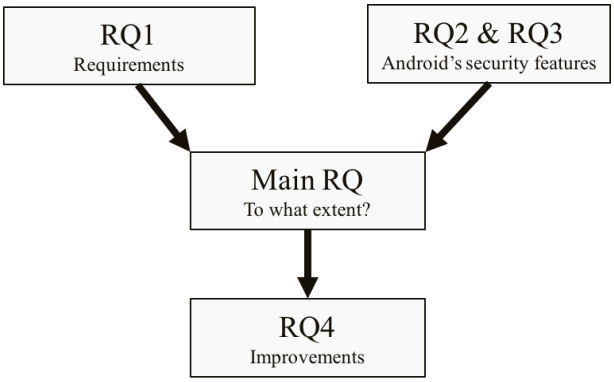

Figure 3.1: Methodology used in this research

RQ1 - requirements for storing sensitive information First the requirements for storing sensitive information were researched. To do so, the related work as described in Section 2 was researched. This includes the guidelines of NIST and CESG.

RQ2 & RQ3 - security mechanisms in version 6 of the Android OS For researching the security features of the Android OS, the main resource has been the Android Project page [Pro16]. Besides the previously existing security features of the Android OS, extra attention was given to new security features introduced in version 6 of the Android OS.

Next, the requirements were compared with the implemented security features to determine if the security was sufficient for storing sensitive information.

RQ4 - improvements of the security features For the areas where the security features were not sufficient, possible improvements where investigated by researching the related work. This research did not implemented any of the improvements, but only investigated them on a theoretical level.

4. Requirements

This section will describe the security requirements for storing sensitive information on an Android based smartphone and thus answers *RQ1*. The requirements are based on the guidelines described in [SS14] by NIST and [CES13] by CESG.

Before diving into the requirements, it is important to understand the Confidentiality Integrity and Availability (CIA) triad.

- The *confidentiality* of sensitive data means that unauthorized users are not able to read the data [Wik16d].
- *Integrity* means that the data cannot be changed by unauthorized users [Wik16d].
- *Availability* means that if a user authenticates successfully sensitive data should be available.

To translate this to data used on smartphones: a sensitive e-mail (or any data) stored on a smartphone (also referred to *data at-rest*) or a sensitive e-mail transmitted from or to smartphones (also referred to as *in-transit*) should not be readable or modifiable by unauthorized users. It should also not be possible for those unauthorized users to modify the contents of the sensitive e-mail. However, the sensitive e-mail should be available when an authorized user authenticates himself successfully.

Application or user data have different requirements compared to the operating system files. The confidentiality of the operating system files is not as important as the integrity of the operating system files. However, if the integrity can not be guaranteed, the confidentiality of sensitive data is at risk. If an attacker can modify system files, he would be able to upload sensitive information remotely. For example a modified system service sending all the received e-mails to a remote server. Because the source code is available for the Android OS project, it does not make sense to ensure the confidentiality of the operating system files.

To ensure the confidentiality and integrity of sensitive data the following subcategories are required:

- data protection, all requirements to protect the confidentially (and integrity) of data;
- platform protection, all requirements to protect the integrity of the platform.

4.1 Data protection requirements

Examples of data stored on smartphones can be amongst others, sensitive e-mails, location data or text messages. This section describes the requirements to protect this data.

4.1.1 Data at-rest

Data at-rest protection refers to data being stored on a smartphone. Data at-rest should be protected sufficiently so that an attacker is required to authenticate himself to get access to the data stored on a smartphone. This applies for example when a device gets stolen. An attacker should not be able to access any sensitive data stored on the smartphone.

4.1.2 Data in-transit

Data being transmitted through untrusted networks should be protected. Strong in-transit protection should be used to protect the confidentiality and the integrity of this data. Thus it should not be possible for the bounded attacker to listen in on communication.

4.1.3 Authentication

To protect the data at-rest and data in-transit, strong authentication should be used. The smartphone should be able to verify the identity of the user, and any services should be able to verify the smartphone and the smartphone verifying services. A possible solution to authenticate the user is through for example a password. Only the user should know this password allowing him to access his smartphone.

4.2 Platform protection requirements

The platform requirements refer to everything for ensuring the integrity of the platform (operating system).

4.2.1 Application protection requirements

To reduce the impact of a compromised application or a compromised component of the platform, there should be segregation between applications. Application permissions should be restrictive and distributed on a 'need-to-know' basis. Applications should only be able to (directly) modify or access their own files.

As described earlier, a flash light application should not be able to access sensitive calendar data. There should be a mechanism in place to ensure which data is accessible by applications.

4.2.2 Secured boot chain

Secure boot chain is a technology in which every step in the boot chain is verified by the next step. Thus when a user powers on his or her smartphone a 'trust anchor' is used to verify the next step in the boot chain. The next step verifies the step after that. This continues until the smartphone is booted. This technology is important because if an attacker would be able to modify anything during the boot sequence he could potentially hide himself. For example, a virus hiding itself from a virus scanner. Thus, it should be impossible for unauthorized processes or users to modify the boot sequence.

4.2.3 Malicious code execution detection and prevention

Vulnerabilities potentially make it possible to execute malicious code on a smartphone. Researchers around the world investigate new methods of preventing malicious code execution. The operating system running on a smartphone should implement the current best practices when it comes to malicious code execution prevention.

Detection of malicious code execution is a more complex matter, the smartphone should provide methods to make it possible to detect malicious code execution. A frequently used solution on desktop computers is an anti-virus solution. An anti-virus solution scans applications before and while they are running on a platform for malicious behaviour.

Also the origin of an application should be verifiable to ensure that an application originated from a specific developer.

It also should be possible to restrict applications from running on a smartphone and only allow authorized applications to run, to reduce the risk of running a malicious application. If there is a defined set of applications that is used in an enterprise, why not only allow those applications to run on the smartphone? This reduces the risk of for example running the potentially malicious flash light application.

Vulnerabilities potentially make it possible to bypass any protection mechanism that might be implemented. Security updates can fix vulnerabilities. An enterprise wants to install those updates a quickly as possible to reduce the risks of those vulnerabilities as much as possible. Thus security updates should be available within reasonable time for distribution to smartphones. Distribution to smartphones should be enforced and automated to reduce the impact of vulnerabilities. This does not only refer to security updates of the operating system, but also refers to security updates of applications.

4.2.4 Security policies

It should also be possible to define and enforce a security policy for the smartphone. This policy should define which protection mechanisms are enabled. It should also be possible to monitor the state of this policy and

the state of the platform. For example through gathering and analysing log files or by monitoring installed applications. An example of a security policy can be to ensure a strong form of user authentication (strong password).

Designed policies should easily be updated, distributed and enforced on smartphones. It should not be possible for users to modify the critical security settings. For example a user should not be able to disable the user authentication.

5. Security Features of the Android OS

The previous section described the requirements for storing sensitive information. This section will describe the implementation of the security features of the Android OS, specifically version 6, answering *RQ2* and *RQ3* and will use the same subsections as section 4.

5.1 Data protection mechanisms

This section will describe the security features ensuring the data protection on an Android based smartphone.

5.1.1 Data at-rest

Android offers data at-rest protection by means of full disk encryption. Full disk encryption is a technology which is used to protect information, making the information unreadable to unauthorized users. Someone trying to access to data needs to know some kind of secret before getting access to the data [Wik16b]. In the case of Android the full flash memory containing all the application data (and thus user data) is being encrypted, thus becoming unreadable for an unauthorized user. There are different encryption algorithms available of which Android uses the Advanced Encryption Standard (AES) encryption scheme and it is required to have 128-bit keys. Also, starting with Android version 5, disk encryption is enabled by default [Pro16] [Inc15a]. According to NIST those keys should be sufficient for protecting data till at least 2030 [Bar16].

The first time the Android OS boots, a key called the *Disk Encryption Key (DEK)* is generated. The *DEK* is used to encrypt data on the smartphone. Android uses another key, called the *Key Encryption Key (KEK)*. The *KEK* is generated through a custom algorithm (see [Pro16] for more details). The *KEK* is based on the user's passcode (password, PIN or pattern) and the *Hardware-bound private Key (HBK)* (a key bound to the phones hardware) [Pro16]. The result of this is that the *DEK* is encrypted with a derivative of the user's passcode. When the user changes the passcode, it does not require re-encryption of the whole file-system with the new

key material. Only the DEK needs to be re-encrypted with the new KEK. Figure 5.1 is a graphical representation of the full disk encryption scheme.

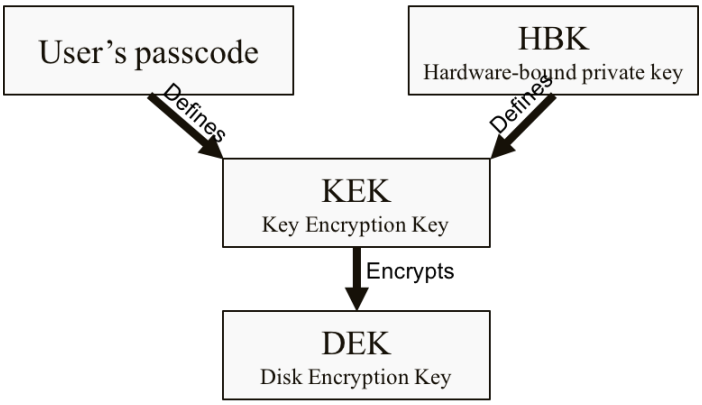

Figure 5.1: Relation between elements for full disk encryption

When the Android smartphone boots, the user inputs his passcode to generate the *KEK*. With the *KEK*, the *DEK* can be decrypted. From that moment on the *DEK* is placed in memory to allow reading and writing data to the disk. Once the phone is turned off, the *DEK* is removed from memory.

The keys are protected by the Trusted Execution Environment (TEE). The TEE is a secure part of the main processor (or a separate microcontroller) of a smartphone. It is an isolated execution environment which provides security features like running Trusted Applications and protecting the confidentiality of Trusted Applications their assets [Wik16f] (like encryption keys). All security sensitive operations should be executed in the TEE. The main processors on smartphones are considered untrusted and cannot access certain areas of Random Access Memory (RAM), hardware registers and fuses where secret data (for example the HBK) is stored. Software running on the main processor delegates any operation that require the use of secret data to the TEE processor via an API [Pro16].

With version 6 of the Android OS the Keystore has been enhanced by adding additional features to the Keymaster Hardware Abstraction Layer (HAL) [Pro16]. For example the addition of *user-authentication-gated* encryption keys, which will be further discussed in Section 5.1.3. The Android Keystore provides the possibility to store encryption keys in some kind of container to increase the difficulty of extracting them from the device. This technology is for example used in storing the *DEK*.

A weakness of the data at-rest protection in the Android OS is that the encryption keys remain in memory when the device is locked [CES16]. The Android project strongly recommends that encryption keys should never be stored on the device unprotected, but this is not required when keys are being used [Pro16]. However, full disk encryption requires to have the keys in memory. As a result, if an attacker steals the device, the attacker would

(theoretically) be able to retrieve the encryption keys from RAM. [MS13] froze the RAM of an Android based smartphone and used a cold boot attack to retrieve the disk encryption keys from RAM [MS13]. A cold boot attack is an attack where the device is turned off and then booted again. In this case not the regular Android operating system will be booted by a custom prepared operating system which is designed to extract the encryption keys from the 'frozen' RAM. This attack relies on the property of RAM to hold its data for a short period of time when frozen to a very low temperature. This attack is not possible on all smartphones, only on smartphones where the bootloader has been unlocked. Section 5.2.2 will elaborate more on this.

[MS13] also theoretically described the possibility of retrieving the encryption keys from RAM through a Joint Test Action Group (JTAG) interface. A JTAG interface is a low level debug interface mostly used by the Original Equipment Manufacturer (OEM) during the development of smartphones or for performing maintenance on broken smartphones. It can potentially give direct access to the contents of the RAM. These interfaces can be protected by requiring authentication, but it is left up to the OEMs to protect their interfaces.

Besides by getting direct access to the RAM, sensitive information would be at risk if the attacker would be able to bypass the lockscreen. The attacker would then have access to all the user's data without knowing the secret (passcode). *CVE-2015-3860*[CER15] is a vulnerability where a buffer overflow caused the lock screen to crash giving the attacker access to all the sensitive information. This vulnerability has been patched, but it illustrates that the encryption scheme could be improved. If the encryption key would not be stored in memory, bypassing the lockscreen would not give access to sensitive information. Such an encryption scheme is used by Apple's iOS [App15].

As already mentioned, the *DEK* is protected by the TEE. Implementing a TEE is strongly recommended by the Android Compatibility Definition Document (CDD), but it is not required [Inc15a]. Not having the hardware backed key protection results in the encryption key stored on filesystem only being protected by a key based on the users passcode (the KEK). This makes the encryption key vulnerable to an off-the-box attack [Pro16]. An off-the-box attack is an attack where the key is moved to another device and then the user's passcode is bruteforced there. A bruteforce attack is an attack where all possible combinations are tried to unlock the phone [Wik16a]. When a TEE is implemented, the KEK is derived from the HBK (together with the user's passcode), preventing an off-the-box attack.

The TEE sounds very promising, but every vendor has to design and create their own implementations. The security of all those implementations should go through extensive security testing. During this research Google announced that they increased the rewards for submitting a vulnerability to the Google Vulnerability Rewards Program. The highest reward stands for finding a vulnerability in either the TrustZone (a TEE implementation) or in Verified Boot [To16] (see Section 5.2.2 for more information on Verified boot). The increased rewards sounds promising, but every different implementation of both the TEE and any security feature of Android should go

through extensive security testing.

There are only a few Android based smartphones which have gone through government certification processes (for example the Federal Information Processing Standards (FIPS) 140-2). An example of such a smartphone is the Samsung S7 [Sam16]. Untested implementations increase the potential risk of security incidents. One of the most used implementatiosn of a TEE is ARM's TrustZone on the Qualcomm processors, this implementation has had multiple vulnerabilities (for example examples [MIT13], [MIT16a], [MIT16c] and [MIT16b]). Two of those vulnerabilities resulted in a fully compromised device, which illustrates that the TEE should be tested extensively because it is a critical part of the security ecosystem of the Android OS.

With the release of version 6 of the Android OS an additional feature was introduced, which allowed the adoption of the SD-card memory into the main storage. This enables the possibility to also use full disk encryption on the SD-card. In previous versions of Android, the SD-card was not encrypted even if full disk encryption was used. However, this feature is not required to be implemented by the Android CDD [Inc15a]. The Android CDD defined which conditions OEMs have to meet before they are allowed to distribute their smartphone with the Android OS.

5.1.2 Data in-transit

To prevent eavesdropping on communication, a Virtual Private Network (VPN) solution can be used. VPN is a technology where data is encrypted over a secure tunnel between the server and the smartphone. Android supports a couple of VPN protocols Layer 2 Tunneling Protocol (L2TP), IPSEC or Point-to-Point Tunneling Protocol (PPTP).

It also offers a feature called Always-On VPN. The Always-On VPN guarantees that all data will always be routed through the VPN. The Always-On feature only works with the native VPN protocols (the protocols listed above). So third party applications using different protocols do not have this functionality.

5.1.3 Authentication

Important is the authentication mechanism used to verify the identity of the user. Native Android offers PIN, pattern and password authentication. Fingerprint authentication is introduced in version 6 of the Android OS.

Another new feature is *user-authentication-gated* cryptographic keys, where access to a cryptographic key is granted if a user is able to successfully authenticate [Pro16]. For example: an application requests a cryptographic key from the keystore application in the TEE, the TEE requires the user to authenticate before releasing the key to the application.

To authenticate a server, x509 certificates can be used. X509 certificates are digital certificates verifying the identity of an entity. To be able to verify the identity of a server, a trust anchor is required. This can be done either through a built-in list of certificates or through a technology called

certificate pinning. With certificate pinning the server's certificate is placed inside an application. When the application connects to a server it check if the certificate offered by the server is identical to the one provided inside the application.

X509 certificates can also be used to verify the identity of a smartphone. The challenging part of this is the distribution of certificates to the smartphone, because this is not provided in the native Android OS. Version 6 of the Android OS introduced the possibility of adding certificate enrolment protocols through third party applications. This allows certificate enrolment protocols to be available on an Android based smartphone through for example a Mobile Device Management (MDM) solution. More information about MDM solutions is provided in Section 5.2.4.

Whether authentication on smartphones is sufficient is arguable. Google stated at the 2016 developer summit that 50% of their Nexus phones are not using authentication (prior to the release of the fingerprint authentication)[Dev16a].

A PIN number could also be considered a weak form of authentication. [Ber12] states that 26,83% of users' PINs can be guessed within 20 tries. The total key space in case of a 4 digit PIN is only 10^4. Attacking this keyspace through a bruteforce attack is doable through the earlier described off-the-box attack. The encrypted *DEK* would be removed from the smartphone and all combinations of the PIN would be tried to reconstruction a *KEK* that decrypt the *DEK*. This attack would only be possible if no TEE would be implemented, because the TEE prevents against off-the-box-attacks as described in Section 5.1.1.

Users tend to prefer something convenient for authenticating to their smartphone. Because of this, a different authentication method was introduced, which is based on a recall-based authentication scheme. A recall-based authentication scheme requires the user to redraw a secret. Recall-based authentication is supposed to give a better user experience and offer higher security because humans are better at remembering patterns than a number sequence. In the Android OS the recall-based authentication scheme is implemented by requiring the user to draw a pattern on a 3x3 grid [Uel+13].

[Uel+13] studied the patterns used in a real world scenario to determine their real strength. Potentially the keyspace of the Android based pattern authentication scheme is 2^{19}. [Uel+13] showed humans are biased when it comes to choosing patterns. For example 38% of the participants in the study used the top left as a starting point for their pattern. This and other predictable characteristics caused the keyspace to be reduced to $2^{10,90}$. Making the key space close to the key space of a randomly picked 3 digit PIN [Uel+13]. Figure 5.1.3 displays the percentages per starting points in Android screen unlock pattern.

If a strong PIN, pattern or password and a TEE are used, Android can put (and increase) a time-out between failed authentication attempt making it infeasible to brute force a PIN code for our bounded attacker. However, the Android CDD does not require an exponential backoff algorithm (the Android CDD however highly recommends implementing it) [Inc15a]. It

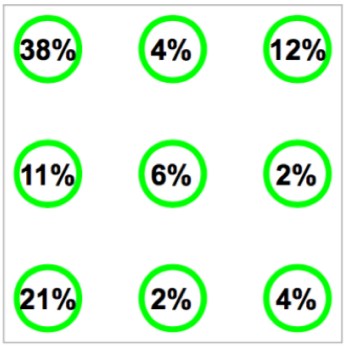

Figure 5.2: Starting point for unlock patterns in percentages [Uel+13]

only requires that rate limiting is enabled for authentication to the lock screen. It also requires to have at least 30 seconds of rate limiting after 5 failed fingerprint authentications. [DM12] showed that the 30 second timeout was implemented on a Galaxy Nexus running Android 4.1. In version 6 of the Android OS this issue has been solved. The exponential backoff algorithm starts after 20 failed authentications and the time out is doubled after 40 failed authentication attempts. After a total of 50 failed authentication attempts, the time out is doubled every 10 attempts.

Android version 6 introduced a new form of authentication: fingerprint authentication. Google states that on their Nexus devices, the amount of users who enabled authentication increased from 50% to 90% [Dev16a] due to the introduction of fingerprint authentication. This kind of authentication scheme has also been implemented in Apple's iOS [App15]. [Rog13] bypassed the authentication of iOS using 'artificial gummy fingers'. The Gummy Finger attack requires the attacker to lift the fingerprint of the user of for example a beer glass. The lifted fingerprint is then digitalised by photographing it. This fingerprint is then printed on gummy tissue using laser equipment, resulting in an artificial gummy finger. [Rog13] states that it was a lengthy process and requires equipment worth thousands of dollars (like a high resolution camera and a laser printer)[Rog13]. This attack might be too complex to be performed by our bounded attacker, still it is a possible risk that the lockscreen could be bypassed.

$$k_b = \frac{1}{FMR(1)} \tag{5.1}$$

For fingerprint authentication, the k_b the effective keyspace of biometric authentication) is based on the False Match Rate (FMR) [OGo03]. The formula to calculate the k_b is shown in equation 5.1. The Android CDD requires at least a FMR of $0,002\%$ resulting in an effective key space of 50000. Making it halve the keyspace of a 5-digit PIN, because a 5-digit PIN has a keyspace equal to 10^5 which is equal to 100000.

According to [OGo03] it is a bad practise to solely rely on biometric authentication, which is the case for fingerprint authentication in Android.

Even if passcode authentication is enabled for full disk encryption, the Android CDD does not require to re-authenticate the user after a specific interval [Inc15a]. This could result in Android solely relying on fingerprint authentication because smartphones are always-on devices (smartphones are not restarted verys often).

CESG[CES16] recommends to use a strong 9 character password for authentication. This would indeed be a stronger authentication scheme than fingerprint authentication, but it comes at the cost of user experience. However, passwords, PINs and patterns are vulnerable to over-the-shoulder attacks where an attacker looks over the shoulder of the victim and captures the secret. This illustrates it is not a trivial task to decide on the required authentication scheme.

5.2 Platform protection mechanisms

5.2.1 Application protection mechanisms

To protect the platform (and applications) from malicious applications, Android offers application segregation by implementing sandboxing capabilities. Sandboxing means that every application runs in a complete separated environment, creating segregation between applications.

Since Android version 4.3 Security-Enhanced Linux (SELinux) has been used for defining the boundaries of Android applications and system processes. In Android version 4.3, SELinux was configured in permissive mode. SELinux provides mechanisms for defining the boundaries of applications and restricting them. Through SELinux it is possible to restrict an application from making a specific system call or prevent it from writing to a specific file or folder.

Permissive mode is a reporting mode, where illegal actions are only logged. Since Android version 5 SELinux has been changed from permissive to enforcing mode. Enforcing mode does not only log illegal actions, but also prevents them. Android version 6 further tightens the boundaries, improving the segregation between applications and thus reduces the threat of exposed services [Pro16].

Besides SELinux, every process runs with its own Linux defined User ID (UID), which takes advantage of the Linux user-based protection [IG14]. Thus when a new application is installed a new user is created together with a separate application folder, which is only read- and writeable by that new user. Other applications are not able to read or write in the newly created folder.

For application permissions, Android 6 introduces the concept of 'Runtime Permissions' where permissions are requested only when they are needed, providing a 'need to know' permission scheme. For pre-Android 6 applications, application permissions are still requested at installation time, but can be revoked at a later time. Application permissions allow applications to perform specific API calls. For example, access cellular information, location information or read the user's contacts.

The segregation and sandboxing have proven to be insufficient in the past. Recently found vulnerabilities, for example CVE-2016-2463 [MIT16d] showed that it is possible to bypass the application sandbox, in case of the example, by sending a specially crafted media file to a smartphone. Of course, found vulnerabilities can be patched with security updates.

5.2.2 Secured boot chain

To protect against an attacker modifying the boot chain, Android implements *Verified Boot*. This secures the boot chain through the device-mapper-verity (dm-verity) kernel feature. A hash table is used to verify (at a block level) the integrity and authenticity of the boot chain. With the introduction of Android version 6 this is now required by Android CDD [Inc15a]. The boot chain starts from an immutable hardware key supplied by the OEM and ends at the booted kernel. Each stage should verify the integrity and authenticity of all the bytes of the next stage before executing the code of the next stage [Pro16].

The *Verified Boot* feature makes sure that the kernel is not compromised at boot time, thus malware will not be able to place itself in the boot sequence without being detected. However, if malware is able to exploit the system at runtime, it is impossible to detect that the system is compromised, because there is no method to detect if the kernel has been compromised at runtime.

There is another problem with *Verified Boot*. Users can disable the *Verified Boot*. One such method is through the process of unlocking the bootloader. Sometimes the OEM provides a method to unlock the bootloader. If that is not the case, users sometimes use exploits to unlock the bootloader. Unlocking the bootloader makes it possible to install different Android distributions, also referred to as CustomRoms. After unlocking the bootloader *Verified Boot* is not enabled anymore and thus the bootchain is not being verified [Pro16]. This makes the user vulnerable to the previously described cold boot attack [MS13].

It is required by the Android CDD to wipe all data after the bootloader has been unlocked. If this would not be the case, sensitive information might be at risk. Once *Verified Boot* has been disabled, any part of the operating system can be changed and thus any protection mechanism protecting sensitive data can be disabled.

5.2.3 Malicious code execution detection and prevention

To protect against malicious code execution Android implements Address Space Layout Randomization (ASLR) [Pro16] and Data Execution Prevention (DEP) [Wik16c]. ASLR and DEP complicate the exploitation of vulnerabilities based on buffer overflows. Buffer overflows are very unlikely in most Android applications because they are developed in Java,. However, Android does support C / C++ applications (called native applications).

Some of the core functionalities of the Android OS are written as native applications.

Another protection mechanism for mitigating buffer overflows is Control Flow Integrity (CFI). CFI is a technology where applications have strict flow control. This control flow describes which function may be called by which function (how does the application flows through its code). For example: only the shutdown button should be able to shutdown a smartphone. Without CFI there is no flow control that prevents the volume button from shutting down the smartphone. [Dav+12] showed a CFI implementation on smartphones, but this framework is not implemented in any version of the Android OS.

It could be argued that CFI does not increase the security, because most of the applications are written in Java. This is partially true, but it is possible to write native Android applications (in C or C++). An example of a native application is the media server, used for playing media files. The previously discussed vulnerability [MIT16d] used a buffer overflow in the media server to execute malicious code. CFI could potentially reduce the risk or increase the complexity of exploiting this vulnerability.

Besides implementing buffer overflow protection mechanisms, Google also actively scans (and rescans when new vulnerabilities are found) all applications uploaded to the Google Play store with the Google Bouncer. It scans for (but is not limited to) coding mistakes, known vulnerabilities and malicious behaviour. Developers who upload malicious applications will receive a time line for 'fixing' flagged applications [Dav16]. [Fox15] described that a group of hackers was able to bypass the Google Bouncer by using complex methods. Eventually, those methods were discovered and patched.

It is possible to install an anti-virus application on Android, which could analyse applications and potentially flag them as malicious. By default Google enabled (from Android version 4.2 and up) the "Verify App" feature [Pro16], where applications are scanned for malware at installation time (remotely by Google). Users also have the possibility to install an additional anti-virus solution. [Tes16] showed that those scanners detect known malware, but the antivirus products could easily be evaded by a minor alteration. It did not require advanced code obfuscation or any similar technologies. [Tes16] also showed that *droppers* are a major problem. *Droppers* are applications that do not contain malicious code, but download the malicious code after the *dropper* is installed. The malicious code never touches the disk, it only remains in memory. None of the anti-virus applications were able to detect *droppers*. Finally [Tes16] showed that none of the anti-virus solutions were able to detect unknown threats.

Applications running on Android are required to be signed. When the application is installed, the package manager verifies if the application is signed by the public key distributed together with the application. Applications do not have to be signed by a trusted party (there does not need to be a trust anchor), signing certificates can be self signed [Pro16]. This feature protects against a malicious user distributing a fake upgrade of a legit application. Based on the certificate, the Android OS is able to verify

if an application is signed by the same developer.

Through the Android for Work API MDM solutions can white-list applications. White-listing can be applied both to allow applications to run or allow applications to be installed [Inc15b]. The next section will describe MDM solutions in more detail.

Most security risks of the scenarios described above can be reduced if the Android OS is fully up-to-date on security updates. It would require our bounded attacker to find a vulnerability and create an exploit.

One of the biggest issues regarding Android security is the distribution of security updates. Every OEM has to implement the security patches themselves in their own Android implementation. The Android Open Source Project will provide updates for at least the three most recent versions of Android and this is a security issue. This could result in 22,9% of the deployed Android devices do not receive security updates any more [Goo16]. Of the other 77,1% it is unsure whether or not the OEM will distribute security updates. Samsung, LG and Google have promised to distribute security updates on a monthly basis for their support devices[Dre15]. Table 5.1 shows the distribution of the different Android OS versions at the moment of writing. Sensitive data might be at risk if the OEM does not provide security updates in a reasonable time (or does not provide security updates at all).

Version	Distribution
2.2	0.1%
2.3	2.0%
4	1.9%
4.1 - 4.3	18.9%
4.4	31.6%
5	35.4%
6	10.1%

Table 5.1: Android version distribution as of June 6th 2016 [Goo16]

[CES16] describes that Android 6 introduces the possibility for device owners to specify an update policy (for example automatically install updates as soon as they are available or during the night). It is not possible to force application updates. Automatic installing updates for applications can be enabled through Google Play, but cannot be enforced (a user can still disable this setting) [CES16].

5.2.4 Security policies

An MDM solution plays an important part in the protection of sensitive information on Android based smartphones. In most cases an MDM solution is an application installed on the smartphone, this application communicates with the MDM server. It sends system information (for example the Android version, phone number or installed applications) to the MDM server. The server on its part sends a policy to the client which speci-

fies for example which authentication schemes are available or the required passcode complexity. The client retrieves the policy and enforces it on the smartphone and finally reports back to the server. This process is shown in Figure 5.2.4. It should be noted that some security mechanisms are only available through an MDM solution, for example the feature to wipe the phone after a maximum of failed authentication attempts. This feature reduces the risk of an attacker guessing (or bruteforcing) the passcode.

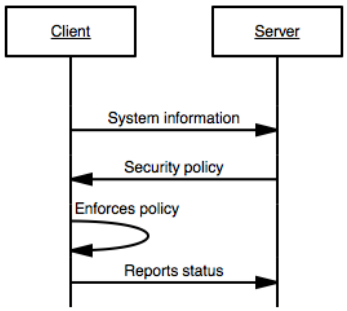

Figure 5.3: Process of applying security policy using an MDM solution

[CES16] described that some settings cannot be enforced, for example the automatic updating of applications through the Google Play store.

[CES16] also described that it is not possible to remotely retrieve system logs from an Android based smartphone. Android simply does not offer any APIs for retrieving system logs. There are some APIs available that provide information on installed applications, running Android version, last contact time with MDM, failed authentication attempts, network state, compliance state, enrolment status, location information or roaming status [CES16]. Most MDM solutions offer the option to collect and store the gather information for analysis.

6. Result

This section will compare Section 4 with Section 5 to determine if the security mechanisms implemented by the Android OS are sufficient to guarantee the security of sensitive information to our bounded attacker and to what extent this sensitive information is protected. This section will answer the main research question.

6.1 Data protection

6.1.1 Data at-rest

Data at-rest might be at risk, because an attacker would be able to retrieve the encryption keys directly from memory with a cold boot attack or through other means (for example crashing the lockscreen). This attack could be too complex to be performed by our bounded attacker and would either require an unpublished vulnerability, an unlocked bootloader or using the JTAG interface.

To reduce this risk, a device owner should make sure that the smartphone has implemented a TEE, the bootloader is locked and the device is up-to-date on security patches. Also if a SD-card slot is available, the device owner should make sure that the Android OS adopts it into the main storage to ensure the data at-rest protection.

6.1.2 Data in-transit

If the native VPN implementation is configured properly, the data in-transit protection meets the requirements. This is because the Always-On functionality guarantees that network connections are only allowed when the VPN is enabled. The Always-On VPN can be disabled by the user. Users could be tricked to disable the Always-On option. To reduce this risk, users should be educated on the risk of disabling the Always-On VPN.

6.1.3 Authentication

Authentication can be considered insufficient when non-random and weak PINs or patterns are used or when random strong PINs or patterns are used

without having a TEE implemented. Bypassing the fingerprint authentication is complicated, but still feasible for our bounded attacker.

A device owner should use an MDM solution to enforce an authentication policy. Which authentication scheme should be used depends on the required protection. A trade-off has to be made between security (requiring a strong password) and usability (fingerprint authentication). The authentication policy should also enforce wiping data after a maximum failed attempts.

Authentication of smartphones and servers can be based on x509 certificates and is currently considered a strong form of authentication, thus this is sufficient [CES16]. An MDM solution should be used which implements certificate enrolment protocols.

6.2 Platform integrity

6.2.1 Application protection

Based on the requirements, Android offers sandboxing, segregation between applications and a 'need-to-know' based permission scheme. The possibility of being exploited through a weakness in the application sandbox can be mitigated by deploying security updates in a reasonable time.

6.2.2 Secured Boot chain

Based on the requirements, Android offers a sufficient secure boot mechanism, but only when the bootloader is locked. Thus a device owner should make sure that users do not unlock the bootloader. This could be done through user awareness.

6.2.3 Malicious code execution detection and prevention

The past has shown that the malicious code execution detection and prevention have not been sufficient. It could be argued that it is too complex for our bounded attacker to find new exploits in version 6 of the Android OS. Applying security updates in a reasonable time could again mitigate this risk, but there is no method available to detect that a kernel is compromised during run time potentially putting sensitive information at risk.

For malware, anti-virus solutions fail to detect small modifications to known malware and hackers have shown that the Google Bouncer can be bypassed. To prevent malware installation, there is an easier solution: only install trusted applications and only allow trusted applications to run.

As described in Section 5.2.3, whitelisting can be used through an MDM solution to further reduce the risk of exposing the platform to a malicious application.

If an Android phone is used that receives security updates after every monthly security patch round, it is protected against known vulnerabilities

that could be used by our bounded attacker. For device owners, security updates should be scheduled to be installed in a reasonable time. Also a device owner should make sure that OEMs distribute security updates for used devices in a reasonable time.

6.2.4 Security policies

An MDM can be used to enforce the security policies. APIs exist to enforce mentioned security features described in this research, apart from some limitations: users can disable automatic application updating and change VPN settings (as already mentioned in Section 6.1.2).

The platform monitoring is very limited and could be improved to allow a more detailed event collection. For example a user disabling the auto update of applications can not be detected by MDMs because Android does not provide an API call to monitor this. However, the current platform monitoring provides the most essential information. The ability not to gather log files does not directly impact the confidentially or the integrity of an Android based smartphone, it, however, does reduce the detection rate of compromised devices.

6.3 Attack landscape

This next section will compare the results with the attack scenarios described in Section 1.3 and determine to what extent sensitive information is protected in the case of the attack scenarios.

6.3.1 Stolen Device

An Android smartphone is still adequately protected when it gets stolen if a TEE is implemented, strong authentication is used, the bootloader is locked and the Android OS is up-to-date on security updates. The possibility remains that data could be retrieved if the attacker has an unknown (or non-patched) vulnerability that either bypasses the lockscreen, unlocks the bootloader without wiping the user data partition or discovers a method to retrieve the content of the memory directly (for example through the JTAG interface).

An MDM solution can be used to enforce the security policies including the authentication policy. However, device owners should make sure that their smartphones implement a TEE, the bootloader is locked and the OEM distributes security updates in a reasonable time.

6.3.2 Malicious Applications

To protect against malicious applications applications should only be installed through the Google Play store, only trusted applications should be able to be installed or be able to run using whitelisting and the device should be up-to-date on security updates. Users should also be made aware

to not disable the automatic updating of the applications. It is only possible through an MDM to implement white-listing of applications.

6.3.3 Exploits

The detection of exploits is rather weak, it is possible to verify the kernel at boot time but it is not possible to detect a compromised kernel at run time. To mitigate this risk, the device should be up-to-date on security updates and the bootloader should be locked.

6.3.4 Eavesdropping

Accessing unsecured networks can be made secure if a proper VPN implementation is used. The build-in VPN should be used, because this allows for the Always-On functionality. It guarantees that all connections will always go through the VPN, only users are able to disable this settings this could potentially make an Android based smartphone vulnerable to eavesdropping on communication. Users should be made aware not to disable the Always-On functionality, because an MDM solution can not enforce this.

7. Improvements to the Android Security Features

As suggested in previous sections, Android security could be improved in a couple of areas. This section will suggest improvements on the security features of the Android operating system. The section will thus answer *RQ4*.

7.1 Data protection

7.1.1 Data at-rest protection

To solve the problem of having the encryption keys in memory when the device is locked required a different encryption scheme than the one currently used. The problem of leaving the encryption keys in memory is that an attacker is not required to know the user's passcode to get access to the data. If he is able to bypass the lockscreen, he gets full access the the user's data without knowing the passcode.

An encryption scheme that does remove the encryption keys from memory when the device is locked is Apple's iOS encryption scheme.

7.1.1.1 Apple's iOS encryption scheme

[App15] describes the encryption scheme used in Apple's iOS. This section is based on this whitepaper.

Every iOS device has a unique key and a device group key placed on the device during manufacturing. No software or firmware can read them directly, they can only be seen as the result of an AES operation. The unique device key allows data to be tight to a device. Besides those two keys, every file has its own encryption key, a 'per-file' key. 'Per-file' keys are protected by so called 'class' keys. Class keys determine *when* the 'per-file' keys are available. Thus to unlock a file, first the hardware key is required to unlock the class key. For some class keys the user's passcode is also required to fully unlock the class key. Which class key is required to unlock a 'per-file' key is stored in the file meta data. This file meta data is protected by the file system key. The file system key is generated when iOS is first installed. Figure 7.1 illustrates this encryption scheme.

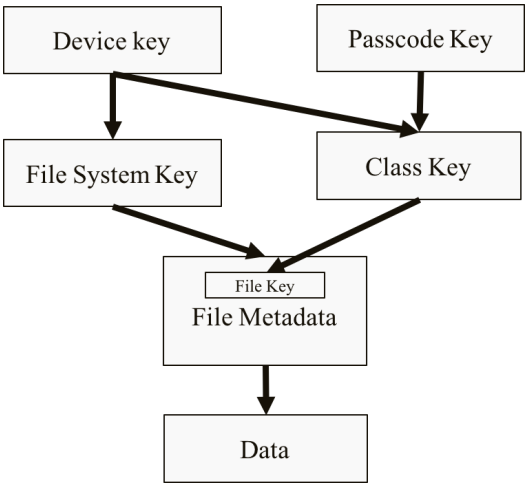

Figure 7.1: Encryption scheme used in Apple's iOS

There are different types of class keys:

- Protected by the passcode, requires user's passcode to unlock the class key, class key is discarded after the device is locked. It is not possible to read or write without the passcode.
- Read protected, makes it is possible to write new files, an example could be an email attachment downloaded in the background, still the passcode is required to unlock the class key.
- Protection after first login, the class key remains in memory after the user authenticates for the first time (at boot time).
- No 'per-file' key protection, files protected by this class are only protected by the device's unique key.

7.1.1.2 eCryptfs

As the Android OS uses the Linux kernel, a good solution would be to use a file based encryption scheme that is already implemented in the Linux kernel. eCryptfs is such an encryption scheme.

In eCryptfs, every file is encrypted with a File Encryption Key (FEK) and the FEK is encrypted with the master key (called File Encryption Key, Encryption Key (FEKEK)). This encrypted FEK is called the Encrypted File Encryption Key (EKEK). The EKEK is stored in the meta-data of the header of the encrypted file. The user's passcode can be used to define the FEKEK [DeF12]. Figure 7.3 illustrates the eCryptfs encryption scheme.

7.1.1.3 Improving eCryptfs

eCryptfs still has the same problem as Android's full disk encryption, the FEKEK has to remain in memory to operate. [DeF12] suggested an im-

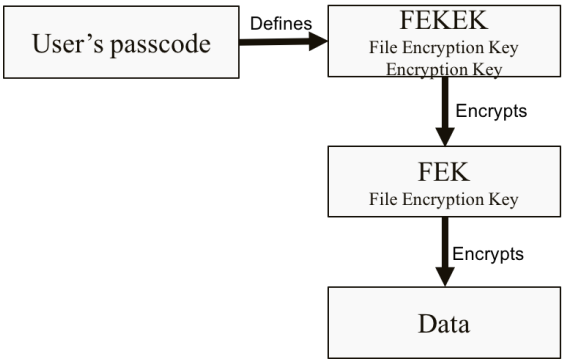

Figure 7.2: Encryption scheme used in eCryptfs

provement to eCryptfs by using boundary keys per application. Boundary keys are generated by concatenating the FEKEK with the UID of the application and then hashing the result. For hashing, the MD5 algorithm was used because the result matched the corresponding 128-bit key length which was required at the time of writing. The boundary keys of application should be removed from memory when the phone is locked.

The described encryption scheme by [DeF12] is similar to the encryption scheme used by Apple's iOS, but its primary weakness is using the UID (which is a predictable number and is not a secret). Because of this, the entropy of the boundary key reduces. Also using MD5 as a hashing algorithm is not recommended anymore [Bar16].

To improve on the eCryptfs scheme, the Apple's iOS encryption scheme can be used. The FEK will be encrypted with a class key. For the sake of brevity, not all the iOS class keys will be described. The 'protected by the pass code' class key is encrypted with KEK. To recall, the KEK is a key based on the user's passcode and the HBK which is a hardware-bound private key. In the case of this class key, when the device is locked the KEK and class key are removed from memory. When the device is unlocked, the passcode can be used to reconstruct the KEK, which allows to decrypt the class key. For more information about the KEK generation process please read [Pro16].

The 'no 'per-file' key protection' class should still use full disk encryption, for this a new key, the HBEK (Hardware Bound Encryption Key), should be used. This key is an AES key bound to the device, should not be retrievable and should either be generated during manufacturing or the first time the Android OS boots. When 'no 'per-file' key protection' class protection is used for a file, it is thus only protected by the Hardware Bound Encryption Key. Figure 7.3 illustrates this encryption scheme.

7.1.2 Data in-transit

The data in-transit protection could be enhanced if it would be possible to manage the native VPN implementation though an MDM solution. Addi-

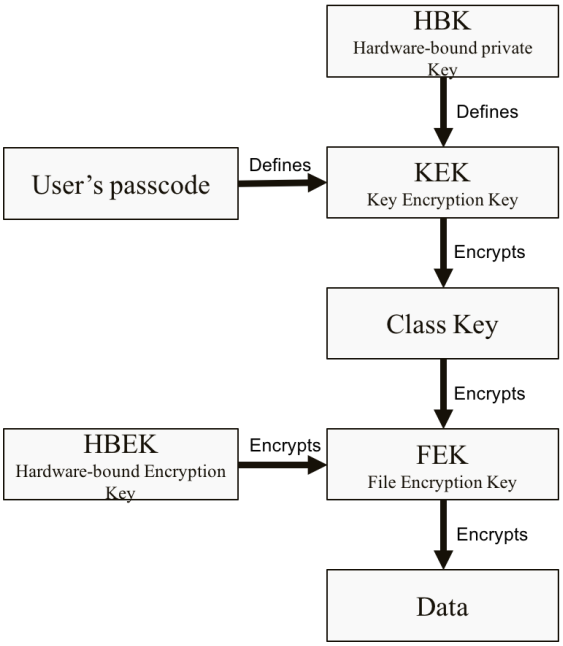

Figure 7.3: File based encryption scheme based on eCryptfs, dm-crypt and Apple iOS encryption scheme

tionally, the Always-On VPN functionality should also be available to third party VPN implementations.

7.1.3 Authentication

As described in section 6, authentication is important for smartphones. Fingerprint authentication offers sufficient authentication, but might not protect against a more capable attacker. Forcing users to use strong PINs or passwords might not be convenient and will reduce the user experience.

A better solution might be to combine authentication schemes. For example require fingerprint authentication and a 4 digit-PIN.

Another solution could be to use fingerprint authentication and require re-authentication for accessing security sensitive information.

Also Android could add the support for wiping data after a maximum failed authentication attempts (which is only available through an MDM). Also increasing the backoff timeout exponentially should be required by the Android CDD. Both mechanisms will prevent the possibility of brute forcing the authentication of the lock screen.

7.2 Platform integrity

7.2.1 Application protection

Based on the requirements, the application protection is sufficient, thus no improvements are required.

7.2.2 Secured Boot chain

Based on the requirements, the secure boot chain implementation is sufficient, thus no improvements are required.

7.2.3 Malicious code execution detection and prevention

Currently there is no method available in Android to detect a compromised kernel at run time. This introduces a security risk, because smartphones are always-on devices, leaving them exposed for a long time. A solution would be to detect the compromised kernel at run time. An interesting approach to solve this issue, also presented in [KW16] is using an application in the TEE to detect the compromised kernel. This both increases the prevention and detection rate of malicious code execution.

The ability to collect device logging might also increase the detection rate, of course when an attacker has gained full root access he is able to falsify log files. In case of a compromised application (for example logging crashed applications like the lockscreen) collecting device logging might help detection. Also just collecting log files is not enough to increase the detection rate, the collected log files should also be analysed to increase the detection rate.

OEMs should make sure that updates are available to phones that are still actively used and that those updates are released in a reasonable time.

7.2.4 Security policies

The APIs used by MDM solutions can be improved by adding additional features like platform monitoring to collect detailed events or to add the possibility of enforcing automatic updating of applications.

8. Discussion

When comparing this research with its primary related work, the CESG guidelines [CES16] [CES15], the NIST guidelines [JS08] [SS14] and the Android Project page [Pro16], this research answers a question that the related work does not answer: to what extent sensitive information is being protected. It also gives an overview of the technical details of the security mechanisms and clearly defines the impact of those security mechanisms. Its primary weakness might be that this research contains too much details for less technical readers, but it fills a information gap which was not previously addressed before.

This research showed that it is possible to protect sensitive data against a bounded attacker. Android offers security mechanisms that are enabled by default. For example the application segregation or malicious code execution prevention. Other security mechanisms need to be enabled or checked by the device owner whether or not the OEM has implemented them.

Some of the security mechanisms are critical for protecting sensitive data. For example not having a TEE puts sensitive information at risk. [Ben16] showed that the TrustZone TEE might not be as strong as expected. The researchers also showed that the HBK was generated in software and was available throughout the TEE making it not *"strongly"* bound to the hardware. A method was described to extract the DEK and breaking the full disk encryption. This attack would still require to find an exploitable vulnerability in the TEE, which might be too complex for our bounded attacker. Still the research showed that the full disk encryption might not be strong enough in the case of storing *extremely* sensitive information that should be protected against a state actor. This would be interesting future research because our research is focussed on protecting sensitive information against a bounded attacker.

Being up-to-date is one of the most important measures to protect sensitive information. Not being up-to-date has an impact on all of the security mechanisms. A interesting case study is the recent 'GODLESS' malware. This malware was distributed through multiple applications (for example in a flash light application, see Figure 8.1). The malicious applications where distributed through the Google Play Store, but also through other means. For example distributing fake applications (for example a legitimate looking social media application) through forums or websites. Inside those malicious applications a couple of known exploits were used to get full control of the smartphone. The malware targeted older, non updated Android versions

(Android version 5.1 or earlier). In the case of this malware, a device would have been protected if it had been up-to-date on security updates [Zha16].

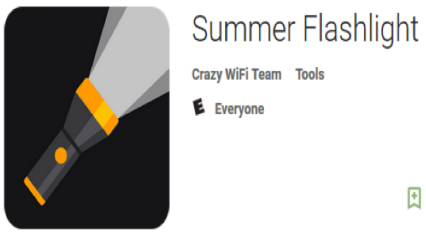

Figure 8.1: Flashlight application which contained the GODLESS malware [Zha16]

In both case studies sensitive information was protected against a bounded attacker if the security mechanisms were in play. Still there are areas that can be improved. For example the data at-rest protection can be improved, as discussed in Section 7.1.1. Google also stated that they see room for improvement and have announced that in the next release of the Android OS (Android version N) a file based encryption scheme will be used [Dev16b]. Implementation details are not available as for now. It would however be interesting to research if the upcoming enhancement implements a file based encryption scheme where the encryption keys do not remain in memory.

Detecting a compromised smartphone is something that is complicated. The suggested solution to detect a compromised kernel through the TEE might cause a significant performance penalty. Every so often (or during every system call) the TEE would have to check if the kernel is not compromised. A solution to this problem might be to implement those check for security sensitive operations only. Summarizing, more research is required to develop a sophisticated solution for this problem. Google is also working on a possible solution for this problem. In Android version N they are working on improving the 'Verified Boot' feature to be able to repair system files, but if this is also possible during run time was not discussed [Dev16b].

Combining authentication methods to strengthen the authentication might protect against simple attackers, but more persistent attackers might still be able to bypass the stronger forms of authentication. It is important that any form of authentication does not decrease the usability of the mobile device to a point where the user is not able to use the device efficiently anymore. Other security mechanisms might reduce the risks that arise when a device is stolen to an acceptable level, reducing the need for a stronger form of authentication.

It should also not be forgotten to educate users about some of the security concerns of a smartphone. For example to not try to install unwanted applications, disable the 'Always-On' VPN option or unlock the bootloader. This same issue is also addressed by [CES16].

9. Conclusion

This research investigated to what extent sensitive information can be protected on an Android based smartphone. First the requirements of storing sensitive information together with the attack landscape were investigated. Sensitive information should be at least protected from attackers with bounded capabilities (investigative journalist, capable individual hackers and the majority of criminals) in the case of device theft, malicious applications, exploits and eavesdropping on communication.

This research also investigated the security features which should protect the sensitive information by researching already existing literature. Android offers data at-rest protection in the form of full disk encryption and data in-transit protection using VPN or other forms of transport layer security. It offers a secured boot chain, application sand-boxing and offers APIs to MDM solutions to further tighten the security of the Android OS.

There are some areas that can weaken the protection of sensitive information. Firstly, encryption keys remain in memory when the device is locked. If attackers are able to bypass the lockscreen, they would be able to access the sensitive data without knowing the user's passcode. This situation could happen when the device is stolen. This research suggested an improvement encryption scheme based on eCryptfs and Apple's iOS encryption scheme.

Secondly, lockscreen authentication is also an area that could weaken sensitive information. Users tend to use passcodes that are easy to remember (either passwords, patterns or PINs). The user experience of smartphone is reduced when users have to enter a long complex passcode when they want to use the device. A solution introduced by Android version 6 is fingerprint authentication. Fingerprint authentication is (by itself) not a strong form of authentication. An improvement would be to incorporate more convenient authentication methods and stack them with fingerprint authentication or require to re-authenticate using strong passcodes only for security sensitive tasks.

Thirdly, Android does not offer any method to detect a comprised kernel during runtime. A solution could be to use the TEE to detect this.

Also any device owner should make sure that the Android based smartphone includes a TEE, has the OEM committed to supply monthly security updates for the used smartphone and install them in a reasonable time, an MDM is used to enforce further tightening of the security features, a trade-off should be made between potential security risks and user comfort

regarding lockscreen authentication, boot loader is locked and no unknown applications can be installed.

10. Future Work

Future work can be to research how extremely sensitive information is protected on Android based smartphones against state actors for example. This would require researching possible methods of state actors and investigate if sensitive information could be protected.

This research has suggested an improved eCryptfs based encryption scheme. This encryption scheme still requires more research before it is implementable.

In 2016 Google will release a new version of the Android operating system. This new version will introduce new security features. It would be interesting to see if the new version further improves the protection of sensitive data.

The 'artificial gummy fingers' attack has not been performed for the Android OS. Future work could be to perform this attack. During this research [Kop16] showed that it is possible to falsify fingerprints for authenticating the smartphones (including Android based smartphones). This shows that the 'artificial gummy fingers' attack should work for Android based smartphones.

More research should be carried out on the security of the different TEEs, because the TEE is a critical part of the security of the Android OS.

Acknowledgements

I would like to thank Jordi van den Breekel, Xenia Economidou and Ruud Verbij from KPMG for giving me extensive feedback on my report. I also would like to thank Loek Sangers from sparring on ideas and providing feedback on my report. Without them this research would not have been on the level it is now.

Bibliography

[App15] Apple. *iOS Security*. Sept. 1, 2015. URL: https://www.apple.com/euro/privacy/d/generic/docs/iOS_Security_Guide.pdf (visited on 05/31/2016).

[Bar16] Elaine Barker. "Recommendation for Key Management". In: *NIST Special publication* 800 (2016), p. 57. URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

[Ben16] Gal Beniamini. *Full TrustZone exploit for MSM8974*. Mar. 30, 2016. URL: http://bits-please.blogspot.nl/2016/06/extracting-qualcomms-keymaster-keys.html (visited on 08/02/2016).

[Ber12] Nick Berry. *Is your password too predictable?* Sept. 2012. URL: http://datagenetics.com/blog/september32012/index.html (visited on 06/14/2016).

[Bug+11] Sven Bugiel et al. "Practical and lightweight domain isolation on android". In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM. 2011, pp. 51–62.

[CER15] US-CERT. *CVE-2015-3860*. Sept. 30, 2015. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3860 (visited on 06/13/2016).

[CES13] CESG. *End user device strategy: security framework and controls*. Mar. 25, 2013. URL: https://www.gov.uk/government/publications/end-user-device-strategy-security-framework-and-controls.

[CES14] CESG. *Government Security Classifications*. Mar. 5, 2014. URL: https://www.gov.uk/government/publications/government-security-classifications (visited on 06/06/2016).

[CES15] CESG. *End User Devices Security Guidance: Android 5.x*. Oct. 30, 2015. URL: https://www.gov.uk/government/publications/end-user-devices-security-guidance-android-5x.

[CES16] CESG. *End User Devices Security Guidance: Android 6*. May 20, 2016. URL: https://www.cesg.gov.uk/guidance/end-user-devices-security-guidance-android-6.

[Cue+15] P de las Cuevas et al. "Corporate security solutions for BYOD: A novel user-centric and self-adaptive system". In: *Computer Communications* 68 (2015), pp. 83–95.

[Dav+12] Lucas Davi et al. "MoCFI: A Framework to Mitigate Control-Flow Attacks on Smartphones." In: *NDSS*. 2012.

[Dav16] Eric Davis. *Enhancing App Security on Google Play*. Apr. 28, 2016. URL: http://android-developers.blogspot.nl/2016/04/enhancing-app-security-on-google-play.html (visited on 06/08/2016).

[DeF12] Daniel DeFreez. "Android privacy through encryption". PhD thesis. Master's thesis, Southern Oregon University, 2012.

[Dev16a] Google Developers. *3rd Annual Google Security Update - Google I/O 2016*. May 20, 2016. URL: https://www.youtube.com/watch?v=gwbmGvur5VE (visited on 06/07/2016).

[Dev16b] Google Developers. *What's new in Android security (M and N Version)*. May 19, 2016. URL: https://www.youtube.com/watch?v=XZzLjllizYs (visited on 08/02/2016).

[DM12] Kitchen Darren and Shannon Morse. *Android Hacking with the USB Rubber Ducky*. 2012. URL: http://revision3.com/hak5/update-with-the-rubber-duck (visited on 06/24/2016).

[Dre15] Emily Dreyfuss. *Big Android Makers Will Now Push Montly Security Updates*. June 8, 2015. URL: https://www.wired.com/2015/08/google-samsung-lg-roll-regular-android-security-updates/ (visited on 06/14/2016).

[Fox15] Thomas Fox-Brewster. *Chinese Cybercriminals Breached Google Play To Infect 'Up To 1 Million' Androids*. Sept. 21, 2015. URL: http://www.forbes.com/sites/thomasbrewster/2015/09/21/chinese-hackers-beat-google-bouncer (visited on 06/14/2016).

[Goo15a] Google. *Device Administration*. 2015. URL: https://developer.android.com/guide/topics/admin/device-admin.html (visited on 06/21/2016).

[Goo15b] Google. *DevicePolicyManager*. 2015. URL: https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html (visited on 06/15/2016).

[Goo16] Google. *Dashboards*. June 14, 2016. URL: https://developer.android.com/about/dashboards/index.html (visited on 06/14/2016).

[Heu+14] Stephan Heuser et al. "Asm: A programmable interface for extending android security". In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, pp. 1005–1019.

[IG14] Sumedh Ingale and Sunil Gupta. "Security in Android Based Smartphone". In: *International Journal of Application or Innovation in Engineering Management* 3 (2014). ISSN: 2319-4847.

[Inc15a] Google Inc. *Android 6.0 Compatibility Definition*. Oct. 16, 2015. URL: https://source.android.com/compatibility/cdd.html (visited on 06/08/2016).

[Inc15b] Google Inc. *Android Security White paper*. May 1, 2015. URL: https://www.google.com/work/android/features/ (visited on 06/09/2016).

[JS08] Wayne Jansen and Karen Scarfone. "Guidelines on cell phone and PDA security". In: *NIST Special publication* 800 (2008), p. 124.

[Kha+12] Sohail Khan et al. "How secure is your smartphone: An analysis of smartphone security mechanisms". In: *Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on*. IEEE. 2012, pp. 76–81.

[Kop16] Joshua Kopstein. *Fake fingerprints: The latest tactic for protecting privacy*. June 27, 2016. URL: http://www.csmonitor.com/World/Passcode/Security-culture/2016/0627/Fake-fingerprints-The-latest-tactic-for-protecting-privacy (visited on 06/29/2016).

[KW16] Uri Kanonov and Avishai Wool. "Secure Containers in Android: the Samsung KNOX Case Study". In: *arXiv preprint arXiv:1605.08567* (2016).

[MIT13] MITRE. *CVE-2013-3051*. Apr. 13, 2013. URL: http://www.cvedetails.com/cve/CVE-2013-3051/ (visited on 06/14/2016).

[MIT16a] MITRE. *CVE-2015-6647*. Jan. 6, 2016. URL: http://www.cvedetails.com/cve/CVE-2015-6647/ (visited on 06/14/2016).

[MIT16b] MITRE. *CVE-2016-0825*. Jan. 6, 2016. URL: http://www.cvedetails.com/cve/CVE-2016-0825/ (visited on 03/12/2016).

[MIT16c] MITRE. *CVE-2016-2432*. May 9, 2016. URL: http://www.cvedetails.com/cve/CVE-2016-2432/ (visited on 06/14/2016).

[MIT16d] MITRE. *CVE-2016-2463*. June 12, 2016. URL: https://www.cvedetails.com/cve/CVE-2016-2487/ (visited on 03/12/2016).

[MP15] Ibtisam Mohamed and Dhiren Patel. "Android vs iOS Security: A Comparative Study". In: *Information Technology-New Generations (ITNG), 2015 12th International Conference on*. IEEE. 2015, pp. 725–730.

[MS13] Tilo Muller and Michael Spreitzenbarth. "FROST". In: *Applied Cryptography and Network Security: 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*. 2013, pp. 373–388. URL: http://dx.doi.org/10.1007/978-3-642-38980-1_23.

[Off14] Cabinet Office. *Government Security Classifications*. Apr. 1, 2014. URL: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/251480/Government-Security-Classifications-April-2014.pdf (visited on 05/31/2016).

[OGo03] Lawrence O'Gorman. "Comparing passwords, tokens, and biometrics for user authentication". In: *Proceedings of the IEEE* 91.12 (2003), pp. 2021–2040.

[Pro10] Android Open Source Project. *Android 1.6 Platform Highlights*. 2010. URL: https://developer.android.com/about/versions/android-1.6-highlights.html (visited on 06/08/2016).

[Pro16] Android Open Source Project. *Security*. June 1, 2016. URL: http://source.android.com/security/index.html (visited on 06/01/2016).

[Qer+14] F. Al-Qershi et al. "Android vs. iOS: The security battle". In: *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*. Jan. 2014, pp. 1–8.

[RF11] Paul Ruggiero and Jon Foote. *Cyber Threats to Mobile Phones*. 2011. URL: https://www.us-cert.gov/sites/default/files/publications/cyber_threats-to_mobile_phones.pdf (visited on 06/07/2016).

[Rog13] Marc Rogers. *Why I Hacked Apple's TouchID, And Still Think It Is Awesome*. Sept. 23, 2013. URL: https://blog.lookout.com/blog/2013/09/23/why-i-hacked-apples-touchid-and-still-think-it-is-awesome/ (visited on 06/14/2016).

[Sam16] Samsung. *KNOX Security Certifications and Validations*. 2016. URL: https://www.samsungknox.com/en/security-certifications (visited on 06/14/2016).

[SS14] Murugiah Souppaya and Karen Scarfone. "Guidelines for Managing the Security of Mobile Devices in the Enterprise". In: *NIST Special publication* (2014). URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-124r1.pdf.

[Sta16] Statista. *Global market share held by the leading smartphone operating systems*. 2016. URL: http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ (visited on 06/29/2016).

[Tes16] AV Test. *The best antivirus software for Android*. Mar. 1, 2016. URL: https://www.av-test.org/en/antivirus/mobile-devices/ (visited on 06/08/2016).

[To16] Quan To. *One Year of Android Security Rewards*. June 16, 2016. URL: https://security.googleblog.com/2016/06/one-year-of-android-security-rewards.html (visited on 06/21/2016).

[Uel+13] Sebastian Uellenbeck et al. "Quantifying the security of graphical passwords: The case of android unlock patterns". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM. 2013, pp. 161–172.

[Wik16a] Wikipedia. *Brute-force attack*. July 27, 2016. URL: https://en.wikipedia.org/wiki/Brute-force_attack (visited on 08/03/2016).

[Wik16b] Wikipedia. *Disk Encryption*. Aug. 1, 2016. URL: https://en.wikipedia.org/wiki/Disk_encryption.

[Wik16c] Wikipedia. *Executable Space Protection*. May 18, 2016. URL: https://en.wikipedia.org/wiki/Executable_space_protection (visited on 06/08/2016).

[Wik16d] Wikipedia. *Information Security*. June 5, 2016. URL: https://en.wikipedia.org/wiki/Information_security (visited on 06/07/2016).

[Wik16e] Wikipedia. *Information sensitivity*. May 14, 2016. URL: https://en.wikipedia.org/wiki/Information_sensitivity.

[Wik16f] Wikipedia. *Trusted execution environment*. May 18, 2016. URL: https://en.wikipedia.org/wiki/Trusted_execution_environment (visited on 06/08/2016).

[Zha16] Veo Zhang. *'GODLESS' Mobile Malware Uses Multiple Exploits to Root Devices*. June 21, 2016. URL: http://blog.trendmicro.com/trendlabs-security-intelligence/godless-mobile-malware-uses-multiple-exploits-root-devices/ (visited on 08/02/2016).