Radboud Universiteit Nijmegen

Computer Science Master's thesis

Security Evaluation of Mobile Device Management Solutions

Deloitte.

Author:
Joost Kremers

Supervisor RU:
Professor Bart Jacobs
Supervisor Deloitte:
Thomas Bosboom MSc

June 4, 2014

1 Abstract

In this paper the interconnection between Android and Mobile Device Management (MDM) is analyzed. After a general introduction to Android, the security related areas in Android are introduced: permissions, sensitive settings, intent spoofing, encryption and rooting. These vulnerable areas specific for Android together with some more general threats, like insecure communication and injection attacks, are bundled into a framework that allows security experts to perform an evaluation of a MDM implementation on Android in a repeatable and verifiable fashion. This framework is based on Keunwoo Rhee's research and is thus dubbed the Extension of Rhee's Framework (ERF). ERF was discussed in depth with two senior security consultants that have performed similar tests in the past.

The current version of ERF consists of seven focus points: enrollment, policy, device settings and audit data, data protection, secure communication, application management and MDM implementation. This results in 23 test cases that include a roadmap on how to perform a security evaluation of a Mobile Device Management implementation on Android. Part of this framework is automated in Mobile Device Management Evaluator, MDM-E. MDM-E tests for exploitable settings, exported Inter-Process Communication and policy discrepancies. Performing a case study with MDM-E on a cloud-based MDM called Meraki, a bug was found that the engineers called "obnoxious". Tens of thousands users can be targeted by the stealthy exploit that is discussed in this paper. MDM-E has thus already proven itself useful when evaluating MDM systems and will be freely available for researchers and other enthusiasts [1].

A survey was filled out by 146 respondents and grants a general overview of the mobile device usage in a corporate environment. It reflects the restraints of companies towards Android, since the percentage of Android on personal devices is much higher than on company owned devices. A total of 23 percent of all respondents have a MDM installed on their device. Personal devices seem to be at more risk, since those devices are more often rooted and more devices allow Unknown sources.

Keywords: *Mobile Device Management, MDM, Android, Mobile Security, MDM-E, penetration test, security evaluation, Meraki.*

2 Acknowledgments

During the six months I have worked on the research leading to this paper I received help from many people. In this section I would like to extend my gratitude to everyone who has contributed to my research: I am in your debt.

[1] GitHub: `https://github.com/JKremers/MDM-E`

First I would like to thank Keunwoo Rhee for sharing his research with me. His work was a great starting-point for the Android specific framework presented in this paper.

My colleagues at Deloitte Cyber Risk Services have helped me to broaden my horizon and allowed me to focus on my thesis. If they did not share my survey with their connections through Twitter and LinkedIn, I would have never received the response I obtained now. Special thanks to senior consultants Werner Alsemgeest and Jochem van Kerkwijk for providing me with feedback on my framework.

The biggest gratitude has to be given to my two supervisors. The scientific foundation was guaranteed by professor Bart Jacobs, while the hands-on mentality and always up-to-date knowledge of Thomas Bosboom opened the door to my professional career at Deloitte.

My friends and family that have supported me during this research were indispensable. I want to thank multiple people that have proofread my paper and have given feedback on it. Paul van Dorst will probably be glad that my "I don't feel like writing this paper anymore"-rants are finally over. Thanks for bearing with me!

Before I begin with the actual content of my thesis, I would like to leave you with a quote from a highly respected computer scientist, professor Knuth. On my own I would have never been able to complete this research, but with the help of other researchers I did.

"People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on eachother, like a wall of mini stones. - Donald Knuth.

Disclaimer: Any possible statements in this document do not reflect official Deloitte statements or opinions. Statements made in this document are personal and are not, in any way, related to Deloitte. Information in this thesis is purely informational towards Deloitte. And although the research (internship) has been conducted at Deloitte, it does not contain any official Deloitte information.

Contents

3 Introduction

Mobile devices are booming. In every field of occupation there is an eruption of portable devices, both personal and work related. These laptops, smartphones, USB-drives and other machines pose new information security threats, especially when the individual or the company possesses sensitive information. The mixture of company owned devices and devices that are owned by an employee (Bring Your Own Device - BYOD) makes this issue even more difficult. To gain control over these mobile devices and the data on them, several companies market Mobile Device Management (MDM) software. These packages are designed to manage the mobile devices on the following levels [3]:

- Software management: configuration, updates, monitoring and backups.
- Network service management: billing, support and more.
- Hardware management: physical management over the devices for inventorying and (de-)activation.
- Security management: enforcement of security policies.

The main components of a MDM solution are a client and a server [19]. The client component is installed as an app on the mobile device that needs to be managed and the server is in charge of distribution of the policy and monitoring of the devices. This server component can either be on premise or online in the cloud. After the client is enrolled to its designated control server, the client regularly sends its device status. The control server reacts with instructions that the companies security policy demands. When the server requires the client to perform an action, it sends a push message to the device with the commands that need to be executed.

From the point of view of a security expert, this may sound like a bullet-proof solution: all the security policies are enforced by the MDM and due to the monitoring capabilities fast incidents response can be achieved. Unfortunately MDM-systems are target of two sorts of attacks. On one side the legitimate user of the device may want to circumvent policies that are enforced by the MDM, for instance: he wants to upload his personal photos to Dropbox, but the companies policy and thus the MDM implementation does not allow Cloud-based services. On the other side there are outside attackers that wants to get their hands on confidential data stored on the device. The MDM wants to make their life harder by enforcing a password-policy and encrypting the data on the device - and an attacker therefore tries to deceive the MDM.

In this paper I will present an overview of topics related to Mobile Device Management on Android, answering the following Research Question (RQ):

- *RQ: How do Mobile Device Management solutions operate on Android?*

This broad question allows us to investigate the interrelation between the MDM and Android and gives us focus on the most crucial and interesting areas regarding MDM. Once I have identified the vulnerable areas in Android I will be able to analyze existing frameworks that test MDM implementations and see whether they take these areas into account. Since executing these frameworks can be very time consuming it can be useful to automate (part of) these frameworks. In this paper the possibility to automate this process will be explored. The three sub-questions (SQ's) that are answered are:

- *SQ1: How are mobile devices used in a corporate environment? And what are the risks?*
- *SQ2: What method is there to structurally test a MDM implementation on Android? Is this method complete and correct?*
- *SQ3: Is it possible to automate (part of) this security-testing process?*

In this paper I will answer these questions.

3.1 Scope

To prevent ambiguity, a proper terminology has to be used. In this paper I will use *solution* or *implementation* when talking about a specific Mobile Device Management solution. *Client* stands for the client-side MDM software - or app - that is installed on the mobile device that is managed. Unless otherwise written the MDM control server is meant when using the term *server*.

In this paper the focus lies on the security evaluation of MDM solutions on the Android platform. The choice for Android was made due to the large market share: 64,2% of all smartphone sales in the second quarter of 2013 were Android-based [9]. Other OS's or modified versions of Android, like the Blackphone or Samsung's KNOX may provide a more secure platform for corporations, but will not be discussed in this paper. Both rooted and unrooted devices are taken into account, especially due to the risks of rooting which will be investigated later in this paper.

Containerization is a technique that can be combined with MDM packages. While MDM focuses on protecting the device, containerization is a technique to secure the data on the device. This is a complete different approach and will therefore not be included.

Due to legal obstructions the vulnerability analysis part of this paper only focuses on the client side, since new MDM solutions tend to run their server in the cloud. Most cloud providers do not allow pentesting on their servers, because it may obstruct normal operations.

The mixture of personal and corporate data on a device makes it hard to determine who is the owner of the device and its data. The legal implications of this mixture pose a complete new field which will not be addressed in this thesis.

3.2 Methods

To answer the research questions taking the scope into account, multiple steps were taken. The basis of this research is literature that was found via libraries accessible via Radboud University Nijmegen and Deloitte, including white-papers and dissertations. With this literature the main research question will be answered. The literature also provides information on the vulnerable areas of Android that should be regarded with extra attention.

To give some insight into what role mobile devices play in a professional environment, a survey was constructed. This survey focuses on the presence of an MDM and gauges the risk that these mobile devices users are exposed to. The literature review also provided one framework that specializes in the security testing of MDM implementations. This framework was used to analyze a MDM implementation (i.e. Cisco's Meraki). From this case study I have created a list of imperfections found in the existing framework. Based on the existing framework, the discovered imperfections and the literature a revised version of the framework is created. A discussion with two senior security professionals at Deloitte about the revised framework yields feedback on the completeness and applicability of the framework. Adjustments to the framework are made in line with this feedback.

Once the framework was completed, part of this framework is implemented in the Mobile Device Management Evaluator (MDM-E) application. This application is scalable and easy to execute. In Figure 1 the total structure of this research is depicted. The complete approach and all substantiations can be found in section 4 of this paper.

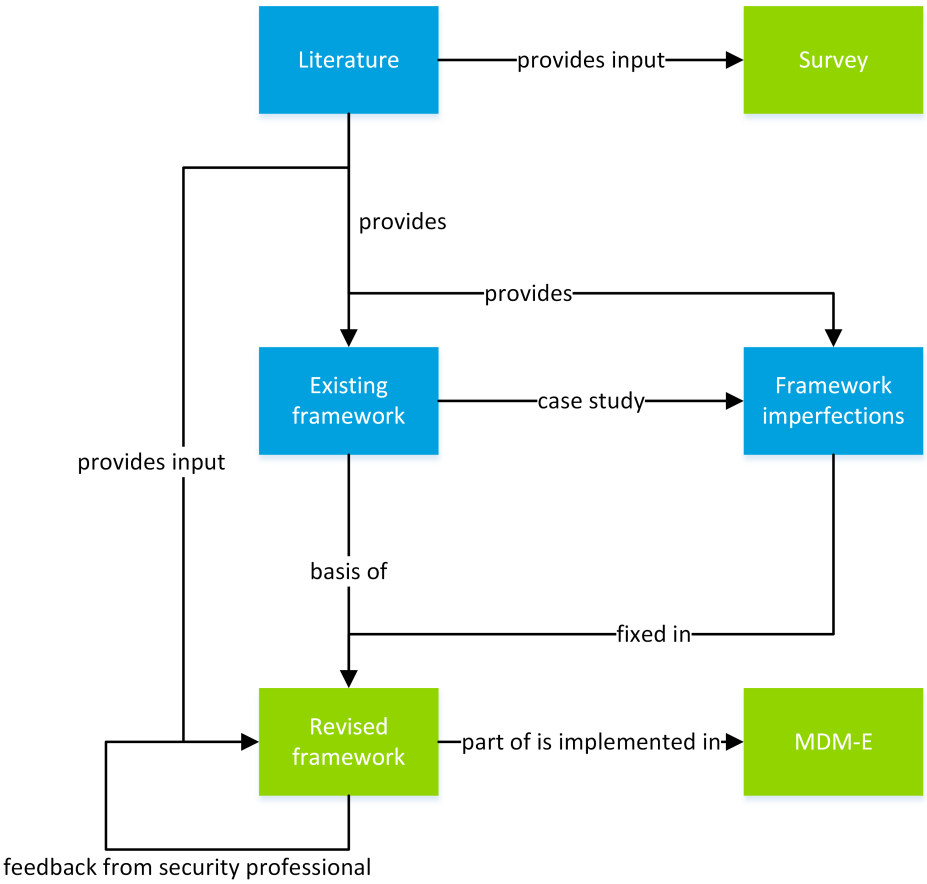

Figure 1: An overview of the research setup.

3.3 Research overview

The next section will discuss the Android Operating System and how MDM vendors can use the functionality that Android has to offer. To get an understanding of the vulnerable areas in Android, the attack vectors will be presented. Special points of focus in this section are rooting, encryption and sensitive settings. The framework that is available for testing MDM implementations is discussed, followed by other research in the field of MDM. In the subsequent section I will discuss the approach taken to answer the other research questions. This includes explanation of the survey-questions and the idea behind the automated testing application. In the *Results*-section, the new framework will be presented together with justification for the alterations. A vulnerability found by MDM-E is also explained in depth, together with a PoC of an exploit for it. This paper ends with a summary of the findings including the conclusions, future work and recommendations.

4 Technical background

In this section the literature that is the basis of this research is presented. The idea is to give an insight on the areas of Android that an attacker may want to target, especially in regards to an MDM implementation. First I will give a look under the hood of the Android operating system, with special focus on permissions, intents, encryption and rooting. The next part of this section zooms in on the interconnection between Android and a Mobile Device Manager, explaining the functionality and threats to MDM. Following this I will go into detail on several generic vulnerabilities that are also applicable for Android, but not Android specific. This ranges from unnecessary logging up to code injection. Subsequent to this I dive into the existing MDM evaluation framework proposed by Rhee: I discuss his vision and list the points of interests that are relevant for Android. Finalizing the section I summarize the list of attack vector an attacker can utilize to subvert a MDM. This list, together with Rhee's existing framework, is the basis of the revised framework that is one of the deliverables of this paper.

4.1 Android Architecture

The Android operating system was originally designed by the company Android, before it was bought by Google in 2005. It is build on top of a Linux kernel [12], which controls the device resources (e.g. camera and network connections). Applications (apps) are run in the Dalvik virtual machine (which will be replaced by ART over time). These applications run in an Application sandbox. This sandbox makes sure an app can only use the system-resources that the app is given permission to access. Next to the Dalvik VM there are native C-libraries that can be accessed. When a mobile device with Android is started, a so called *bootloader* loads the OS. The OS normally starts in Android-mode, while it is possible to start it in Recovery-mode. Recovery-mode is used to apply software updates or to restore the device to factory-defaults.

The application sandbox makes use of the Linux user-permission-model. The difference in this model between Linux and Android is that in Linux every user has it's own ID, while in Android a unique user ID (UID) is provided to an app. If an app wants to access data that is not within the same UID, it has to have the same group ID: this can only be the case when the apps are developed (and thus signed) by the same developer.

The folder structure used in Android is the same as in Linux. An important folder is the /data folder which contains application settings and data. The folder is split into sub-folders for every app with the full application name as folder-title (e.g. com.application.example). Without making changes to the OS, this folder is not accessible for other applications besides the app itself and system apps. If malicious applications can access an application specific /data-folder, they may change settings or content for that specific application. If an

attacker is able to alter the setting file for an MDM implementation and these settings are then enforced by the MDM, he is thus able to circumvent the MDM.

Figure 2: An overview of the Android system Architecture. Based on work from [12].

To protect the user against unwanted access to system resources, apps have to request permissions. During the production of an application the developers must list the permissions the application requires in the Android Manifest. This manifest is packed into the applications installation file, which has the .apk extension. There are permissions for most of the system resources: vibration of the phone; access to GPS data; sending an SMS and many more [14]. On installation of the application the user is prompted whether he agrees or disagrees granting the permissions to that specific application. In the latter case he waives the installation: it is not possible to deny individual permissions of an app without a rooted device. Unfortunately research has shown that only 17 percent of the users paid attention to the permission-requests [11]. An attacker can misuse this knowledge by adding permissions that can give him certain gain. He can send premium texts [36], forward confidential files [2] and much more.

Since every manufacturer uses its own hardware, they are responsible for updating the OS to a newer version. This leads to slow adaption of new Android versions and a splintered version spread. The Google Dashboard [16] shows that versions like 2.3.3-2.3.7 (Codename Gingerbread) and 4.0.3-4.0.4 (Ice Cream Sandwich) still have a solid 15-20% market-share as of March 2014. The newest version, 4.4 KitKat only has a 2.5% share - but is on the rise: on the first of April this percentage has risen to 5.3% [16]. These slow updates can result in various issues, including errors with app compatibility and bugs remain present in older versions of Android for a long time.

In the Android settings menu there are two settings that should be considered in regards to security: the Unknown sources setting and the USB-debugging option. The Unknown sources setting allows the owner of the device to install applications outside of the Android Play store. This is risky since these applications have not been tested by Google and may contain malware. The other setting, USB-debugging is hidden in the Developers settings-menu. Enabling USB-debugging allows the device to communicate with a computer. You can for instance open a shell to the device from this computer and push apps to the device. The program mostly used to do this is called ADB (Android Debug Bridge) and is part of the Android SDK. If this setting is enabled an attacker may be able to push applications (with malware) to the device without the owner knowing. From Android 4.2.2 the device owner is prompted whether he wants to communicate with a connected computer or the connection will be blocked.

4.1.1 Inter-process communication

Applications on Android consist of a combination of three different components. Activities represent the visual side of the apps and are the main entrance point of an application. Services run in the background, mostly for a long time. The third component is a broadcast receiver that listens to certain commands that are send through the system. Inter-process communication (IPC) between

different components and even other apps can be done via *intents*. Like the name implies an app can send an intent to send an email, for instance. If there is a broadcast receiver listening to that intent, Android can start an activity with an email-program. This broadcast receiver is set up to listen to a certain *intent filter*. It is possible to let a receiver only listen to internal components by adding permissions or simply stating that the receiver can only be triggered from internal components. If this is not done, other applications may trigger sensitive components of the applications: In [5] possible attacks on IPCs are presented, which are displayed in Table 1. In a blogpost by Palominolabs it is shown that unprotected receivers in the Paypal app allow an attacker to make a legitimate payment-screen [8], which falls in the Exported Activities - Activity Launch (with data) category.

Unauthorized Intent Receipt	
Intent type	**Potential vulnerability**
Send Broadcasts	Broadcast Theft (without data)
Send Broadcasts	Broadcast Theft (with data)
Send Activity requests	Activity Hijacking (without data)
Send Activity requests	Activity Hijacking (with data)
Send Service requests	Service Hijacking (without data)
Send Service requests	Service Hijacking (with data)
Intent Spoofing	
Component type	**Potential vulnerability**
Exported Broadcast Receivers	Broadcast Injection (without data)
Exported Broadcast Receivers	Broadcast Injection (with data)
Exported Broadcast Receivers	System Broadcast without Action Check
Exported Activities	Activity Launch (without data)
Exported Activities	Activity Launch (with data)
Exported Services	Service Launch (without data)
Exported Services	Service Launch (with data)

Table 1: Intent attacks on Android components. Courtesy of [5].

For Mobile Device Management applications it is essential that important intents are protected. When intents can for instance be intercepted, the attacker may learn more about the internal operations of the MDM or he may be able to prevent the intent from reaching the right component. In case of intent spoofing the attacker may be able to spoof the *UpdatePolicy* intent, which allows him to change the protection enforced by the MDM. He may also perform a DoS attack on an activity that is not protected, which will mainly annoy the user of the device.

4.1.2 Encryption

As of Android 3.0 the user is able to encrypt the /data partition. Encryption is performed by the dm-crypt layer in the kernel [15]. When activating encryption of the device, the user is required to have a password or PIN code. The users password/PIN and a salt obtain from /dev/urandom are used as input in Password-Based Key Derivation Function 2 (PBKDF2). After 2000 rounds, this function outputs a 32-byte value, which is split in a 16-byte AES-key and a 16-bit Initialization Vector (IV).

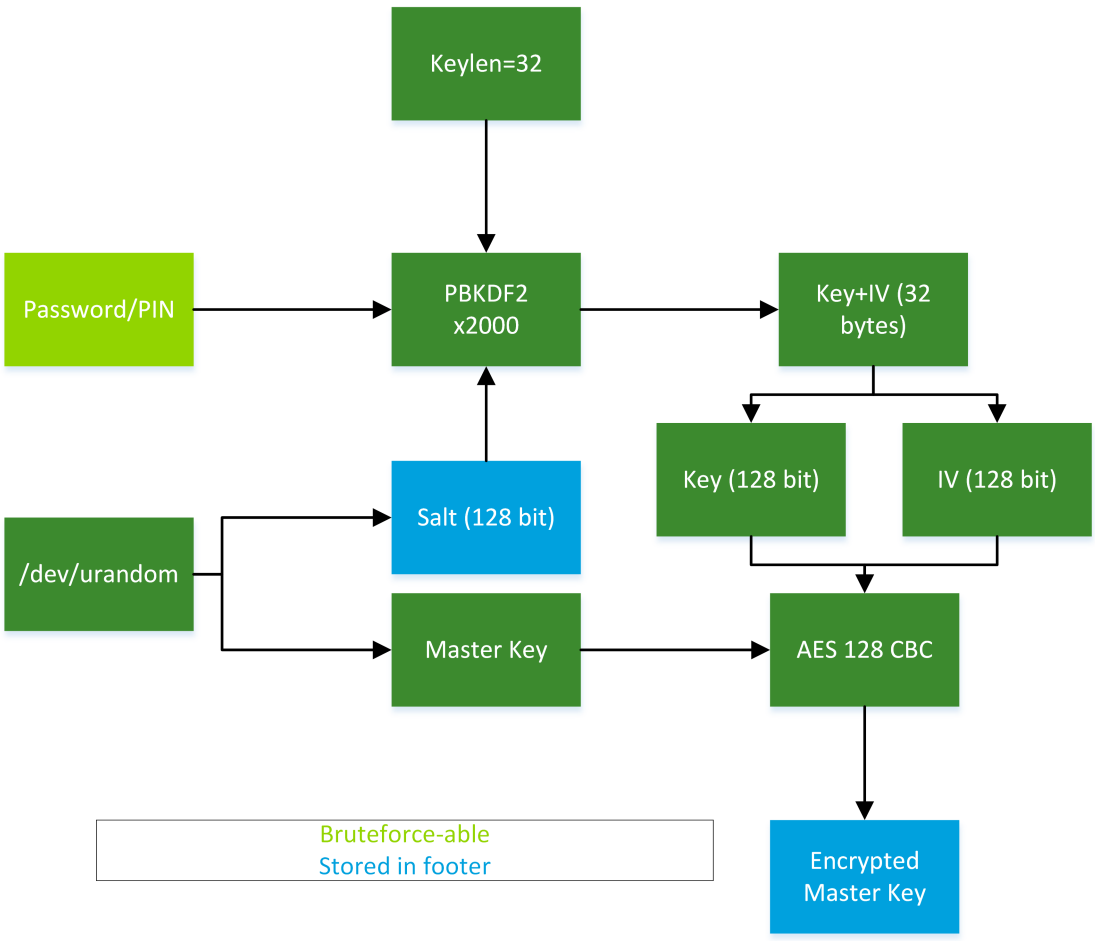

Figure 3: An overview of the Android Encryption key creation. Based on work from [4].

The actual encryption uses AES-128 in CBC-mode together with an ESSIV:SHA256 IV. The encrypted master key and the used salt are saved unen-

crypted at the end of the encrypted folder. The decision to use the user supplied password has been criticized, especially since this allows offline attacks to obtain the users password: if you can make an image of the filesystem, you obtain the encrypted master key and the used salt of the encryption from the footer. You can perform a bruteforce attack on the password and reverse the AES-operation to obtain a possible Master key. Since the /data partition always starts with the same header, you can verify if you have the correct Master key and thus the correct user password [4]. This means that for a 4-number pincode, you only have to check for a maximum of 10000 (= 10x10x10x10) PINs. Even when a more complex password is chosen you can perform this attack offline, circumventing the maximum amount of password tries enforced by the MDM. In a blogpost Elenkov recommends users to change the encryption PIN to a different value by directly calling the cryptfs library from a command-line [10]. The drawback of this is that the user has to remember an additional key that he needs to enter at boot-up.

4.1.3 Rooting

Rooting (iOS: jailbreaking) refers to gaining root access on the Linux kernel. This is the highest user on the system, which means you are allowed to perform higher privileged actions. On one hand this can provide the user of the system with extra functionality, but on the other hand it may also cause him or his company a lot of harm.

Rooting is achieved by privilege escalation due to a security vulnerability mostly specific to the hardware and the operating system [33]. An example of such a vulnerability was found in the Exynos 4210 and 4412 processor, which power the Samsung Galaxy SII, SIII and other popular smartphones and tablets. An exploit using this vulnerability can "bypasses system permissions on the kernel level, taking advantage of read/write permissions within the kernel" [29]. Granting the user root access.

Following the lecture by A [21], rooting allows the user to "load custom software (ROMs), install custom themes, increase performance, increase battery life, and the ability to install software that would otherwise cost extra money (ex: WiFi tethering)". He continues by explaining that users normally only have a *guest*-account.

On the downside of the slope rooting may cause severe harm, since it circumvents security features enforced by the Android OS. The process of rooting may *brick* the telephone, which means that the alternations to the OS caused the telephone to no longer be operable. Furthermore the security of your device can be compromised. Since every application can request root permissions, every application potentially has access to all confidential data on the device. In the case of the previous mentioned Exynos vulnerability, the attacker is even able to read confidential RAM data.

In an article by the Open Web Application Security Project (OWASP) on rooting, it is mentioned that "many users do not understand that jailbreaking or rooting can potentially allow malware to bypass many of the device's built in security features" [25]. Even the controls enforced by a MDM implementation can be circumvented. It is therefore a good practice for these packages to implement root detection mechanisms. In [17] and [25] the following root-detection techniques are discussed:

- Finding a third party market.
- Attempt to write outside of the sandboxed directory.
- Verify if the ROM is signed by a release-key. If a custom ROM is installed, it was signed by a different key.
- Search for an app that may provide root access. Examples of these are: com.noshufou.android.su; com.thirdparty.superuser; eu.chainfire.supersu; com.koushikdutta.superuser; com.zachspong.temprootremovejb; com.ramdroid.appquarantine and many more.
- Check whether the command-line command "su" can be invoked.

There is a growing interest in root-detection evasion methods on forums like XDA-Developers. In [1] it is for instance explained how to surpass the root detection techniques that MobileIron has implemented. AirWatches root detection ability was also subject of investigation and in [18] it is shown how to circumvent these methods. Both the MobileIron and the AirWatch workarounds are probably not usable anymore, due to patching. To make sure the MDM correctly detects the modifications made to the device, it is recommended to implement all root detection methods known at that time and keep them up to date.

4.1.4 Non-Android specific threats

An important measure for a company to manage its information flows is the security policy. In this policy it is stated what is allowed and what is disallowed. It is crucial that there is no discrepancy between the policy and the implementation enforced by the MDM on the device. These discrepancies may for instance arise when a system administrator makes a typographic error when setting up the MDM.

A potential information leakage can be found in the form of unnecessary logging. These logs may contain sensitive information about the device and the MDM. On Android you can access the logs via ADB, by executing the command *adb.exe logcat*. Other logging may be stored on the SD-card or in the applications /data-folder. An attacker may try to capture these sensitive logs for information gain. Developers of an MDM should therefore make sure that

only crucial issues are logged and these do not contain confidential data.

Communication between the MDM and the server should be done in a secure manner. Using a certificate rolled out via SCEP can be used to set up a SSL-connection between the application and the MDM. If this connection is not secured an attacker is able to extrapolate sensitive data that is transmitted, for instance GPS-location of the device or WiFi credentials pushed to the device.

In the security policy a company has, there may be a section on permitted and forbidden applications. Most MDMs use a blacklist or a whitelist to adhere to these rules. It is recommended to use a whitelist instead of a blacklist for this purpose. The blacklist - enumerating disallowed applications - may not be complete and may thus allow the device owner to install applications that the policy prohibits. A whitelist - enumerating allowed applications - although less user-friendly, makes sure no disallowed applications can be installed on the device.

Just like websites, web-pages on Android may also be vulnerable to SQL injection and other forms of injection. The application's activity may use a WebView to display information to the user - and allow the user to input data. This data, if not properly sanitized, can be used to execute database commands that allow an attacker to reflect sensitive data. In [31] it is shown that modern MDM implementations can also be vulnerable to these attacks.

4.2 Mobile Device Management on Android

Since Android 2.2 (Froyo) developers are able to use the Android Device Administration API. This API can be used to develop MDMs on Google's mobile operating system by allowing specific security system calls. The amount of control the MDM can enforce by using this API, depends on the version of Android that is running on the target mobile device. Table 2 visualizes the controls that the MDM can manage by using the Android Device Administration API [13].

When the system administrator has pushed the MDM software to the device, the OS prompts the user if he wants to acknowledge the solution as Device Manager. The user sees all of the permissions that the MDM requests (dependent on the previously mentioned version restrictions). Once enabled it does not necessarily mean that all of these permissions are used by the MDM: this entirely depends on the implementation and the security policy in place. There is a clear distinction between the app part and the Device Manager part: the app part has control over the GPS, the network connection and the file system; while the Device Manager part has control over the security features.

	≤2.1	≥2.2	≥3.0	≥4.0
Password required		x	x	x
Minimum password length		x	x	x
Alphanumeric password		x	x	x
Complex password			x	x
Minimum letters in password			x	x
Minimum lowercase in password			x	x
Minimum non-letter in password			x	x
Minimum digits in password			x	x
Minimum symbols in password			x	x
Minimum uppercase in password			x	x
Password expiration			x	x
Password history restriction			x	x
Maximum failed password			x	x
Maximum inactivity lock			x	x
Require storage encryption			x	x
Disable camera				x
Set new password		x	x	x
Lock device now		x	x	x
Restore to factory defaults		x	x	x

Table 2: The Device Administration API availability [13]. The first 16 are restrictions, while the last three are direct actions that the Device Administrator may call.

Having access to this API allows MDM solution developers to customize their package with more complex rules: the app can request the GPS data; see whether the mobile devices is located within the company walls and then let the Device Manager call the setCameraDisabled() function to make sure no on site company secrets can be photographed. Although this sounds as a good protection, various methods can be devised that circumvent this feature, for instance: a dummy location can be redirected to the mobile device to make it think it is outside of company walls [7] or the phone can be turned off until it is in a room inside of company walls that does not have GPS coverage.

4.2.1 Threats for MDM

Rhee et al. designed Mobile Device Management requirements by first emulating a threat and then applying methodology based on Common Criteria [35]. This allows customers to structurally define the requirements they want from an MDM solution; developers can improve their system by applying the framework and evaluators of MDM systems can use it as a guideline. Together with the

vulnerable areas of Android, this is a good starting point to keep in mind when evaluating a MDM solution. They identify 13 threats:

#	**Threat**	**Description**
1	Disclosure	Any information that leaks from the MDM system. This disclosure can be in the form of unnecessary logging, an attacker listening to an unprotected communication or access to the /data-folder.
2	Software	The MDM or OS can be altered by an attacker. This includes the pushing of applications to an ADB-enabled device and the rooting of devices.
3	Bypass	The attacker can bypass protections enforced by the MDM.
4	Data authenticity	An attacker alters data in the MDM without authorization. Multiple attack vectors are present on Android for this threat, for instance altering data in the /data-partition and injection.
5	Data transfer	An attacker alters communication from or to the MDM without authorization, for instance when not applying SSL.
6	Traffic	An attacker can capture and analyze data on the channel. The passive variant of the named above threat.
7	Spoof	An attacker is able to spoof the identify of an authorized user.
8	Malware	The MDM system can be infected by malware. Having a malware-scanner may provide the user with a more secure device, especially since 99 percent of the malware is targeting Android devices [6]. From Android 4.2 and higher Android has a built-in malware scanner.
9	Denial of Service	Normal operation is disturbed. An example of such a threat is given in the IPC-section, where an activity can be started continually.
10	Leakage	An attacker is able to extract sensitive data from the MDM.
11	Record	The storage can be flooded so that new security events can no longer be saved.
12	Disaster	An unforeseen natural disaster can interfere with the normal operations of the MDM. Which is not within the scope of this paper.
13	Zero-day	A new security vulnerability is found that can be used to attack the MDM. It is infeasible to prevent a zero-day attack. It is crucial to detect such an attack and to be able to respond to it.

Table 3: Threats to a MDM. Courtesy of [35].

4.3 Rhee's MDM framework

To make the life of a security consultant more easy and to make sure all security audits are performed in a structured way, multiple frameworks for testing mobile applications are available. OWASP splits its framework [26] up in three areas: Information gathering; static analysis and dynamic analysis. The first area speaks for its self: this includes the permissions, used protocols and finding out connected entities. The static analysis involves analyzing the source-code, received from the developers or obtained by reverse-engineering. The last area requires the security expert to perform more in depth tests, including the exploration of exported IPCs and SSL checks.

A more specific framework for testing MDM implementations has been developed by Rhee for his Ph.D. thesis on the Security Evaluation of a MDM solution [27]. In this thesis he proposes a new threat modeling method based on Common Criteria, followed by defining the security requirements for MDMs. Rhee creates a mapping between the threats and the security objectives. His main contribution in this thesis is the evaluation criteria and process "to evaluate an agent of the mobile device management system." This framework consists of 17 cases that test the security controls of a MDM solution. In his paper Rhee applies his framework on a (not mentioned) MDM package. His cases consist of an explanation of the case, the tools needed to perform the test and the criteria that is needed to pass the test. For every tool he gives a manual, but not an actual path to follow when performing a security evaluation.

Category	**Evaluation item**
Audit	Audit data generation
Authentication	User authentication
Authentication	Device locking before user authentication
Authentication	Authentication failure
Authentication	Session locking
Data protection	Data encryption
Data protection	Data integrity
Data protection	Device locking
Data protection	Device wiping
Data protection	Encryption key and cryptographic data management
Secure communication	Transfered data encryption
Secure communication	Connectivity
Hardware management	Device control
Application management	Application installation, removal, execution, termination
Self protection	Protection of an MDM agent
Anti-malware	Anti-malware interworking
Detection of modification	Modification detection

Table 4: Test-cases for evaluating a MDM. Courtesy of [27].

This framework from 2012, although built very structured, leaves room for interpretation. His work also does only look at the technical implementation of the MDM, while policy discrepancies are not taken into account. Platform-specific vulnerabilities, as described in this paper for Android - are not incorporated in the framework. He also presented a list of improvements for the tested MDM implementation, but like mentioned before this is not useful since he has not disclosed what MDM he tested.

4.4 Other research

Besides all the relevant research shown above there is a large collection of MDM related papers that are interesting but not usable in this paper. These are processed below.

In Wiewiora's whitepaper [34] he acknowledges that MDM "is quickly becoming a critical necessity for organizations", due to the large increase in employee owned devices that are also used for work and the IT department trying to keep control over the sensitive corporate data. He mentions that companies often apply MDMs that are platform specific or that they only specialize into one function (e.g. wiping). He investigated elements that are part of an effective MDM Strategy, that manages "the entire life-cycle across multiple platforms and devices". The main elements of this strategy are:

- a holistic mobile framework: manage devices and applications with respect to the user.
- strong security policies: to minimize security threats from leaking information onto a system out of reach of the company, strong security policies can help. Companies are advised to implement strong password policies, data encryption should be applied and devices should be tracked, locked and wiped when needed. Passwords should be multi factor-based. Wiewiora advises a combination of something you have (a smartcard), something that you know (a password) and something that you are (biometrics).
- full lifecycle device management: following Gartner, Wiewiora says that enterprises should make their system suitable for multiple platforms and OS ("managed diversity"). Not all versions should be allowed, for instance an enterprise should not allow someone to use Android 2.1, since this does not allow any device manager; or when storage encryption is one of the requirements Android 3.0 or higher has to be used. He also recommends to use app black- and whitelisting,

In Joey Janssen's Masters thesis he introduces Mobile Risk Assessment Method (M-RAM) [20]. M-RAM is made based on existing industry risk assessment methods and interviews with 22 mobile security experts. The method consists of three components: a risk assessment process; entities involved in the system; attention areas that are weak spots in MDM systems. This allows companies or evaluators to evaluate the mobile risks a company has and M-RAM

gives possible solutions to minimize the risk. He names four vulnerable areas: 1) lost and stolen devices; 2) unauthorized data access; 3) personal and enterprise data mix-up; 4) inability to enforce the security policies.

Rhee wrote a paper in which he and his colleagues designed a MDM based on Common Criteria [28]. This is supposed to be platform independent, improves the security of current solutions and is still highly usable. The client side app consists of the following entities:

Application management module manages the apps installed on the device (e.g. blacklist, updates, execution).

Audit and report module collects information from the device (e.g. IMEI and connection status) and talks to the communication module.

Communication module provides the actual communication between the agent and the server.

Device control module is able to control the actual hardware (e.g. NFC reader, camera and USB debugging).

Policy management module is in control of the configuration of the other modules. It also makes sure unauthorized modifications are not possible.

Security management module is in charge of authentication policies (e.g. password protection and the detecting if the MDM is deleted).

The server side consists five entities:

Application management module: is in control of the whitelisted and blacklisted apps.

Audit and report module: saves the gathered data.

Communication module: provides the actual communication between the agent and the server.

Identification and authentication module: is in charge of enrolling and authenticating users.

Policy management module: allows you to configure the policies.

In [30] the discussions about mobile development between multiple Chief Information Security Officers (CISO) were summarized. They talked about the upcoming issues regarding BYOD and enterprise supplied mobile devices. Also potential solutions (MDM and containerization) are presented. They raise a lot of questions on potential policy issues that have to be taken into account when implementing one. The counsel recommends looking ahead for other solutions besides MDM packages, but lists some clear questions that should be answered when evaluating a MDM system, for instance: does it support strong authentication; does it affect battery life; is root-access detected? The last recommendation they do is to include mobile development in the long term vision of the company.

4.5 Summary

In this section the Android operating system has been presented including the interconnection with the MDM. Most of the MDM functionality is utilized by calls to the Device Management API. It is explained that discrepancies between the policy and the implementation can lead to problems as well as multiple sensitive security settings (USB-debug and Unknown Sources). Furthermore proper care has to be taken when using Inter-Process Communication in an app. The dangerous of using the basic Android encryption mechanism and the vulnerabilities introduced by rooting are listed. These attack vectors are crucial to take into accounts when evaluating a MDM implementation. The framework made by Rhee in [27] tries to achieve this, but in my opinion leaves some gaps when it comes to clearness and Android specific cases. These issues are combined and used as input for the (revised) framework that is proposed in this paper.

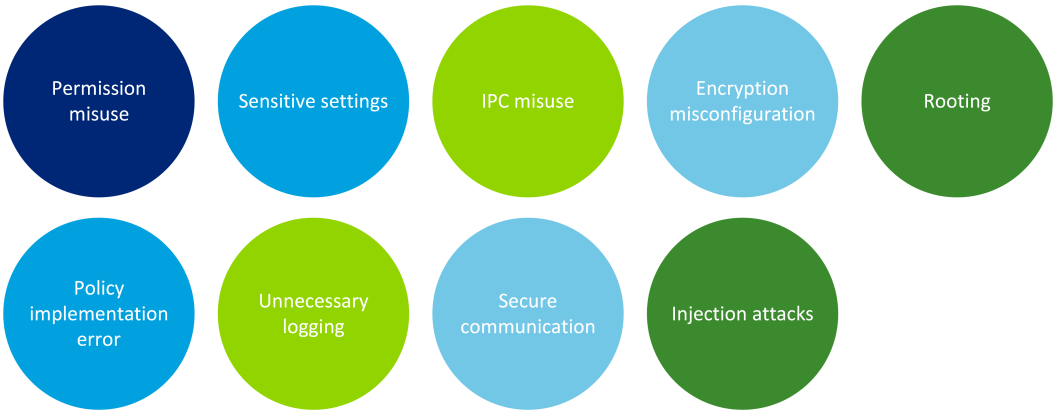

Figure 4: The identified weak spots in Android that should be regarded when testing a MDM implementation.

5 Approach

5.1 Survey Mobile Devices in a Corporate Environment

In section 3.2 I explained the need for a survey. Now the setup of this survey is presented. The purpose of the survey is to obtain an overview of the way mobile devices are used in a professional working environment, hence answering SQ1. After the survey I want to be able to make statements about this usage globally. To achieve this I need to specify our population. The following formula is used to determine the sample size [32]:

$$n \geq \frac{N*z^2*p(1-p)}{z^2*p(1-p)+(N-1)*F^2}$$

Variable	Meaning	Value used
n	Sample size	-
N	Sample population	20000
p	Response probability	0,50
F	Error-margin	0,05-0,10
z	Standard deviation	1,96

In this formula the sample size does not grow very much above a population over 20.000, therefore I will use N = 20.000. This makes sure that, based on the outcome of this survey, I can make valid statements about the global mobile device usage in a corporate environment. p is the response probability which means how big the chance is a certain answer is picked. Since this is very hard to predict for such a survey, the standard value of 50% will be used. The standard deviation is 1,96 which is used frequently for comparable surveys. The margin for error F is set between five and ten percent - the final error-margin will be calculated based on the final number of respondents. An error-margin of 5% means that if the average value of a question is 50%, the adjusted result lies between 45% (50-5) and 55% (50+5). To get a broad overview of the mobile device usage, this error-margin is sufficient.

5% error margin:

$$n \geq \frac{20000*1{,}96^2*0{,}50(1-0{,}50)}{1{,}96^2*0{,}50(1-0{,}50)+(20000-1)*0{,}05^2} = 375$$

10% error margin:

$$n \geq \frac{20000*1{,}96^2*0{,}50(1-0{,}50)}{1{,}96^2*0{,}50(1-0{,}50)+(20000-1)*0{,}10^2} = 95$$

The goal is to reach more than 95 respondents. In a similar research that was performed early 2013, 148 mobile device users completed their survey [22]. Given the same population this gives them a error margin of 8%. This is within of the margins of 5-10% I set, validating my setup. Ideally the amount of respondents

is much larger, resulting in a smaller error margin. Besides the size of the sample, we also want to have an even spread of respondents (aselect survey). A total aselect survey is very hard to achieve with more than 95 respondents, therefore I will verify that from all fields of occupation and size of the company there are roughly the same amount of respondents. The survey is distributed by my colleagues, friends and me via social media (Facebook, Twitter and LinkedIn) and through three forums (Tweakers.net, XDA-developers and Fok.nl). The advantage of social media, especially Facebook - is that you probably will receive lots of respondents because these people know you. The advantage of posting it to large forums is that you reach a large audience and you can ask additional questions on these forums that allow other insights that may be relevant for this research. The main disadvantage is that the users of a forum mostly fit in a specific group - so validating the respondents should be done with care - and that the response percentage will probably be low. The Radboud University of Nijmegen has provided me with a license for Qualtrics, that allows the survey to be published online for free.

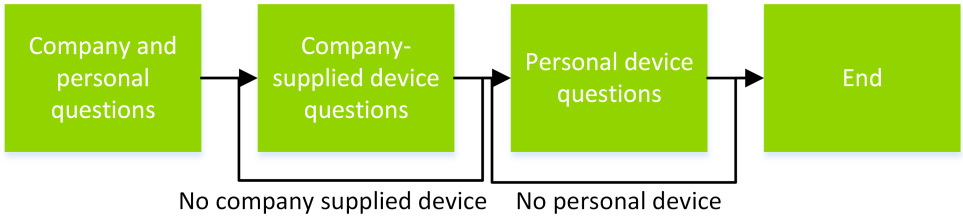

Figure 5: An overview of the survey setup.

In the figure above the structure of the survey is displayed. It starts with several questions related to their occupation and their own (perceived) computer knowledge. These questions are used to validate that the sample population is a sectional of the total population. The (perceived) knowledge is also used to correlate the knowledge to security related risks (i.e. rooting). A Likert scale is used to ask the respondent about this knowledge.

The respondent is then asked whether his employer provides a mobile device. If the respondent answers "Yes", he enters a question-path related to this device. The questions range from ownership of the device; the type of OS; security and password policies. Following this there are some questions related to mobile device risks (rooting and sensitive settings). These questions allow me to make statements about the devices used and the risk the population is exposed to. After these questions the path ends and the user is prompted if he uses a personal device for work purposes. If he answers "Yes" an identical path as before is started. The answers of the two paths can be compared and I expect the personally owned devices to be exposed to more risk than the corporate devices.

Many questions in this survey rely on previously answered questions. It is for instance irrelevant to ask for the version of Android the respondent has if he answered that he has an iOS device. The full survey can be found in Appendix A. The conditions for displaying a question are listed right before the question in the "Answer if..." format. All of the questions are simple to make sure that they are clear and cannot be interpreted in a different way. If a different (or non expected) answer has to be given, the respondent can add it in the Other text-box.

5.2 Extension of Rhee's Framework (ERF)

From the literature review the only framework specific for MDM implementation testing was the framework by Rhee [27]. To test whether this framework is complete and sound a case study on Cisco Meraki has been performed. A special focus is the applicability of the framework to Android. The actual case study is not relevant for this research, but the feedback on it is. Therefore the imperfections in the framework and the missing cases are documented and reported in the Results section. These imperfections and missing cases are corrected in the revised version of the framework and dubbed as Extension of Rhee's Framework (ERF). To improve the workability and validity of ERF, it has twice been discussed with senior security consultants at Deloitte. The comments received from these professionals, that perform similar security tests on a regular basis, make sure the framework can be used in practice. Once a workable version is released Rhee will be contacted to discuss ERF with him, if he is interested. The final cases will be linked to the threats for MDM, as introduced in section 4.2.1. The final version of ERF can be used to test any MDM implementation on Android and includes a detailed roadmap on how to perform the tests. The result can be used to answer SQ2.

5.3 Mobile Device Management Evaluator (MDM-E)

To answer SQ3 I propose MDM-E: Mobile Device Management Evaluator. This Android application automates part of the Extension of Rhee's Framework. To show the potential of this project, a Proof of Concept (PoC) of MDM-E is created. The main structure of the application can be seen in Figure 6.

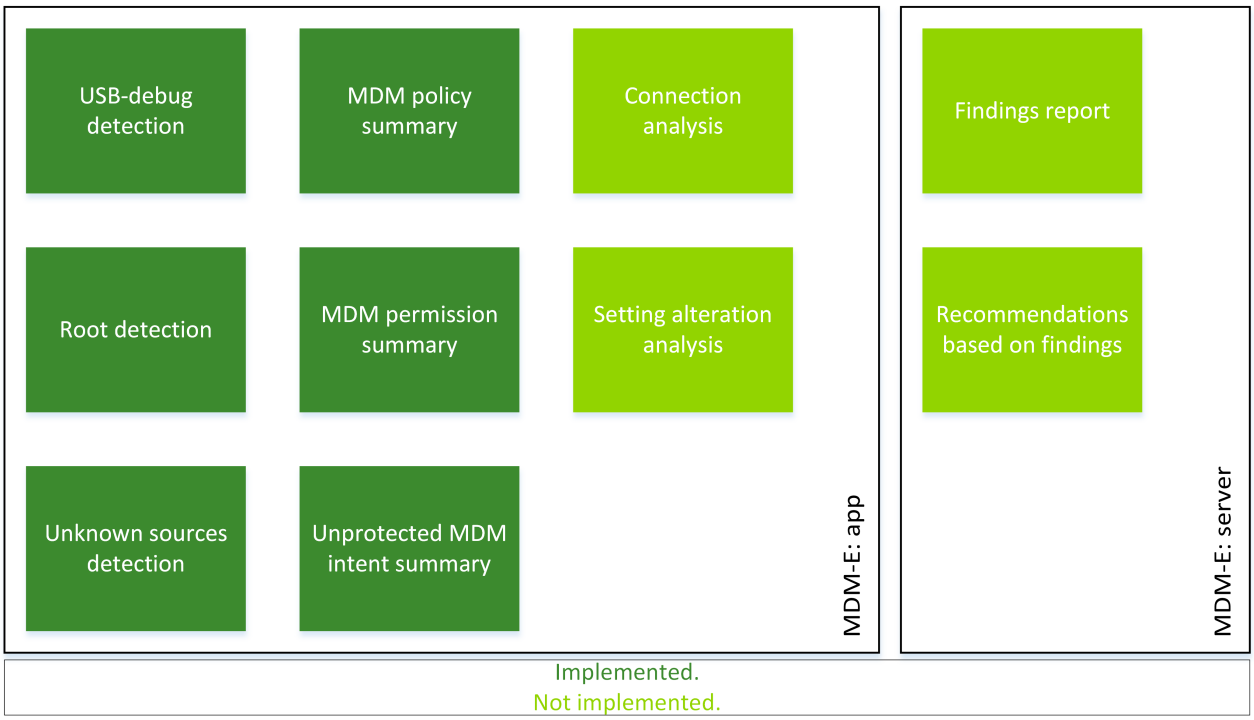

Figure 6: An overview of Mobile Device Management Evaluator.

The PoC is created using the standard Android SDK and should be scalable to include more test-cases in the future. The idea of MDM-E is that instead of manual testing, the application can be downloaded to and executed on the device. MDM-E has several points of interest on the device:

- Policy overview.
- Setting overview.
- Exported intents.
- Connection analysis.
- Setting alteration analysis.

The result of the tests is pushed to an MDM-E server that is used to visualize the findings. On the dashboard an executive summary is presented, but also detailed descriptions of the findings can be requested. These detailed comments can include the Best Practices that should be implemented for a certain case. In the following subsections the individual components of MDM-E are explained.

5.3.1 Policy overview

Since every MDM application works differently and takes a while to get used to, MDM-E contains a series of API-calls that displays the policies enforced by the MDM. This is done by first asking the OS who the current device policy manager is. The goal of this component is to be able to quickly verify that the companies information security policy matches the policy implemented by the MDM. The companies security policy may for instance require a password of 6 characters, but due to a typographic error only 4 characters are enforced by the MDM.

5.3.2 Setting overview

On Android there are multiple settings that should not be enabled, because they expose the user to much risk. Like mentioned in the Technical background, Rooting is dangerous for the security of the device. In that section also several detection mechanisms were iterated. MDM-E has most of these mechanisms implemented in it and reports the result to the server. Since the modular structure of Android applications, additional root-detection methods can be added easily. This test is implemented in MDM-E to verify if the MDM correctly reports rooting of a device. As discussed earlier, the root-detection evasion methods are very popular and this method can thus not be a hundred percent accurate.

Another potentially harmful setting is USB-debugging, as described in section 4.1. An attacker can force the user to connect his device to a computer (possibly a fake charger) and read data on the device, installing and deleting data and more. This setting is also checked by MDM-E.

The (for now) last setting that is tested for is Unknown Sources. Enabling this setting gives the user the possibility to install .apk files that are not in the store. These files are not checked by Google and may contain malware. MDM-E reports this setting as well. Since MDM-E is not published in the Google Play store, the setting Unknown Sources needs to be enabled on the device when MDM-E is installed. When MDM-E is finalized it can be uploaded to the store and Unknown Sources can thus be disabled by then.

5.3.3 Exported intents

In the Technical background I have explained how Android components can interact with each other and pointed out that exported intents can be an entry-point into an application. It is feasible that a MDM has an intent that updates the policy on the device - if this was intent is exported any other application can invoke this update. To make sure the MDM intents are protected, MDM-E enumerates the exported intents. The Android API does not make this enumeration easy, but with a detour it is possible: ask the OS for all services, activities and receivers; filter all relevant to the component retrieved in the Policy overview

step; if it is exported print it. In the future modifications to MDM-E can be made to automatically test the consequences of these exported intents. For now this has to be tested manually.

5.3.4 Not included in PoC

The component of MDM-E that verifies secure communication with the MDM-server is not implemented: communicating with the hardware modules that connect to the server is hard and requires more research. Another function that is not in the proof of concept is the ability to automatically change settings that are stored on the device (mostly in the /data folder). And the usefulness of this function has to be explored: this component should verify whether the MDM package's settings can be changed and if these changes are noticed by the MDM - but *where* these settings are stored and *how* these settings can be altered depends entirely on the implementation, which makes this component hard to automate. The last component that will not be implemented in the PoC of MDM-E is the server. This is mainly a portal that displays findings for a specific penetration-test and highlights findings that do not meet Best Practices.

6 Results

In this section the results found during this research are presented. First the findings from the survey are listed, followed by the adjustments made to Rhee's framework including the final version of ERF. This chapter ends with results found when using MDM-E.

6.1 Survey Mobile Devices in a Corporate Environment

The survey that has been performed grants an overview of mobile device usage in a corporate environment. First I will present the respondents that filled out the survey. After that I will provide the findings split into company provided and personal devices.

A total of 146 respondents have started the survey. This leads to an error margin F of roughly 8 percent:

$$\frac{20000 * 1{,}96^2 * 0{,}50(1-0{,}50)}{1{,}96^2 * 0{,}50(1-0{,}50) + (20000-1) * F^2} = 146$$

Unfortunately only 118 respondents have actually finished question one. This leads to error margin that is a bit higher (9%). The survey was online for roughly two months: from the 12th of March 2014 up to the 13th of May 2014. I now have to verify that these respondents are a balanced reflection of society. The first couple of questions allow me to make statements about this, as show in the next two images.

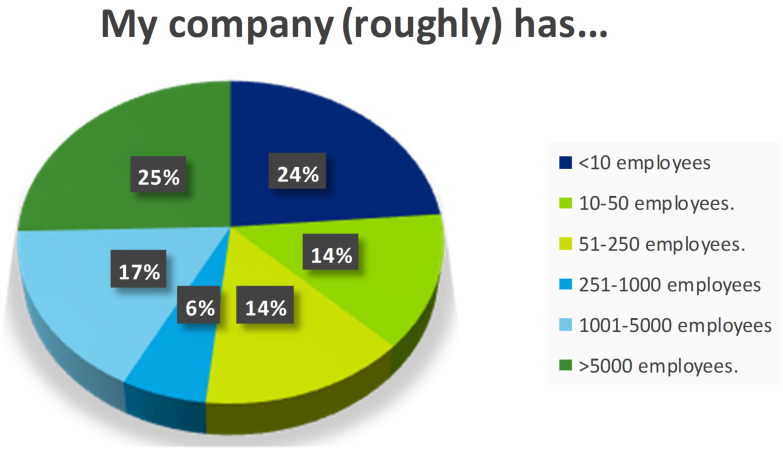

Figure 7: Question 2: My company (roughly) has...

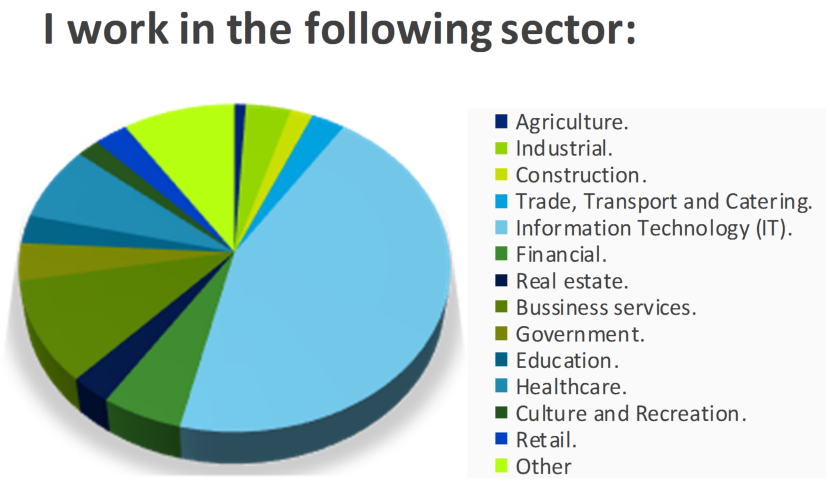

Figure 8: Question 3: I work in the following sector.

From this data we can see that the respondents are nicely split between company size, but from question 3 we conclude that there is a large portion of respondents working in the IT-sector. This can be explained due to the nature of the survey and the methods of finding respondents: people that work in IT are more likely to fill in an IT-related survey and discussion boards on Tweakers and XDA-developers draw in a lot of IT-interested people. This also leads to a higher technical knowledge: 89 percent of the respondents rate their computer knowledge as Good or Very good. To verify that my survey is not clouded by the large amount of IT-ers I have to compare the results of the IT-ers and the non-IT-ers. Doing this for multiple of the control questions shows that the IT-ers are responsible for the higher computer knowledge. Most of the differences between the IT-ers and non-IT-ers stay within the 10 percent margin, with a few exception that will be named as such: in 27 percent of the cases the IT-ers responded that there were no strict rules about mobile device usage, versus 45 percent with non-IT-ers. Although this slightly distorted image, this survey will still be able to give a general image of mobile devices in the corporate environment.

75 percent of the respondents have received a mobile device from their employer. These devices are mostly used for calling, texting and e-mailing. 14 percent uses the device to report, while 31 percent stores data on it. It is interesting to see that, despite the much higher sales number for Android in comparison to iOS, iOS has a much higher share in the corporate world. From the survey 53 percent have a company supplied phone with iOS, in comparison

to 26 percent Android. This supports my research by showing that Android for corporations is still in its infancy. Or maybe the companies are anxious to use Android, since it is targeted by malware on various occasions [6]. What supports this statement is that for the personally owned devices the largest share is held by Android (43 percent Android versus 41 percent iOS). Only Android-versions of 4.0 and up were reported in the survey, which is in contradiction with the current Android Dashboard statistics (where there is still a 17 percent market-share for older versions) [16]. This may be the case because companies force the devices to be updated. The complete spread of OS's on corporate devices can be found in this chart:

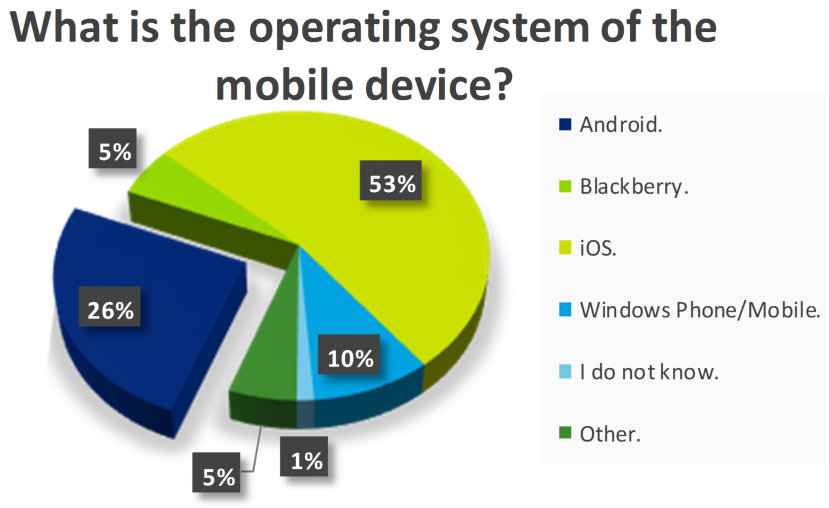

Figure 9: Question 7: What is the operating system of the mobile device? (Company provided)

One of the questions asked in the survey was related to their companies mobile security policy. It is interesting to see that 54 percent of the companies do not have such rules (or they are not known to the end-user). What is striking about this number is that 23 percent of the people that said their company does not have a mobile policy, actually have a MDM installed. Having a MDM installed on your device leads me to believe that the company does have a policy. Even more alerting is that 81 percent of the users store confidential data on their device (ranging from e-mails to reports). Planning an awareness session on the company policies may improve the knowledge of the employee on data protection. 32 percent of all respondents have a MDM installed on their mobile device. Like expected, market-leader MobileIron is at the top of the chart.

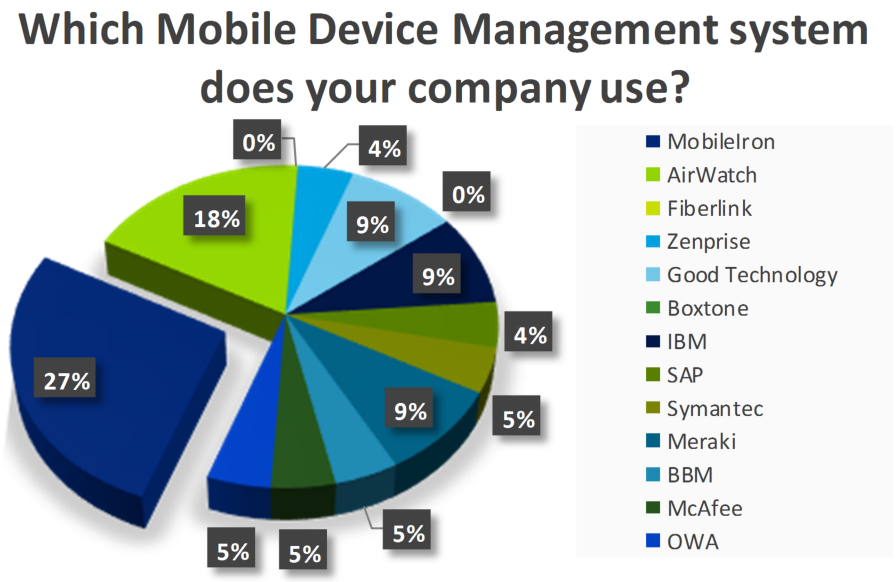

Figure 10: Which Mobile Device Management system does your company use?

The questions of the vulnerable security settings showed that seven percent has his company provided device rooted/jailbroken. Some do this to install third party markets; some to circumvent the policies implemented by the MDM; and others are required to do so by the company so that it can manage the security settings. The last one leads to questions, since most sources advise to exclude your rooted devices from the corporate environment. Due to the anonymous nature of the survey I had no opportunity to ask follow-up questions. 56 percent of the Android users have the Unknown sources option active, making them an easier target for attackers - since the attacker can target these users from outside of the Google Play store. A numeric pincode is the unlock-mechanism for 68 percent of all corporate provided devices; 21 percent have another mechanism and 11 percent (versus 21 percent on personal owned devices) can unlock his device without any protections. The length of the average pincode was 5,42 characters, with an upper bound of 23.

63 percent of the respondents also have a personal device that they sometimes use to perform corporate tasks with. They perform the same tasks on their personal devices as on their company provided devices. With an average pincode of 5,23 characters the difference between company provided and personally owned devices is minimal. More than double the amount of devices are rooted are personally owned, which was to be expected since they actually own

the device. In seven percent of the cases this was done to install custom ROMs. Twelve percent of the respondents with a personally owned device have a MDM installed on their device, showing that the control companies have over BYOD devices is not that large. Eighty percent of the owners of personal devices have Unknown sources activated, which indicates that (in comparison with the 56 percent of the company provided devices) the BYOD devices pose more risk to the companies data.

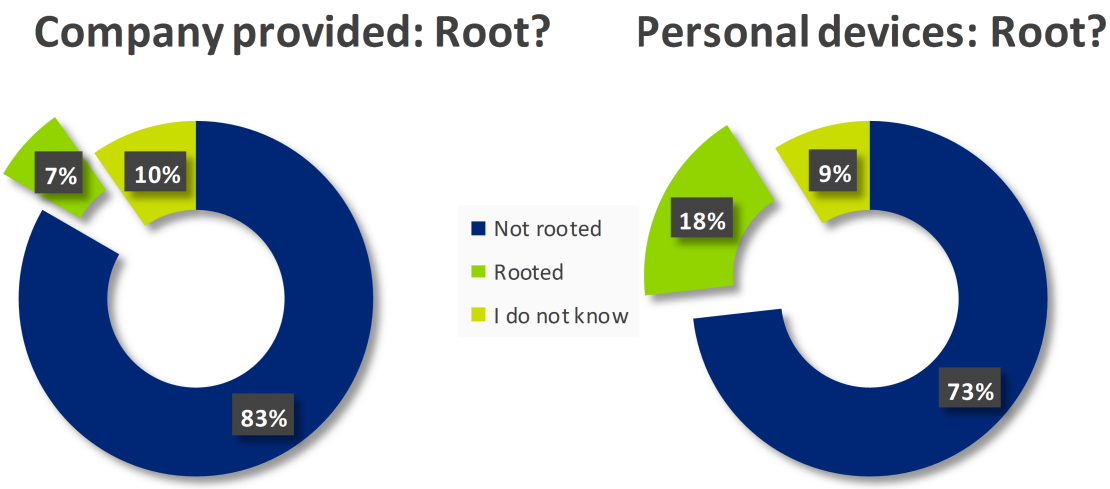

Figure 11: Company provided devices versus personal devices: root.

The last information I have drawn from the survey is the understanding of the permissions asked when an app is installed (on Android) or when it requests a permission during runtime (on iOS). For this I have congregated the corporate and personal devices, so I can get a general image on this issue. Unlike most other questions there is a gap between the IT people and the other respondents: 73 percent of the IT-ers understand the permissions versus 51 percent non-IT-ers. This does show a significant increase in understanding if you compare the overall result of 60 percent "Yes, I read and understand them" to the survey performed in [11]. This may be the result of the large amount of media coverage malware for mobile devices has received lately. The other 40 percent of the respondents are not aware of the permissions and will most-likely accept any permission requested. This may be exploited by a piece of malware.

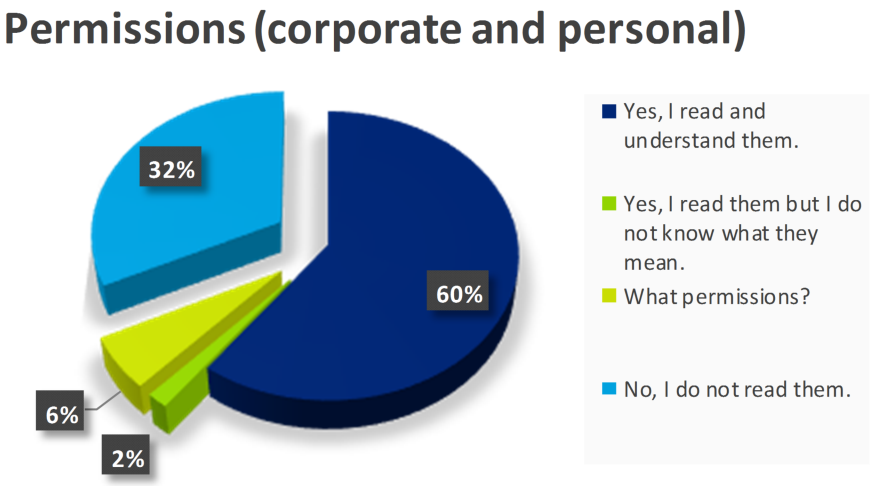

Figure 12: Company provided devices and personal devices: permissions.

6.2 Extension of Rhee's Framework (ERF)

Rhee's research is a good basis for a MDM implementation testing framework. However I do not agree with certain cases that Rhee incorporated in the framework. Since the focus of this research lies on MDM implementations on Android, Android specific cases can be included in the framework.

The case study performed on Cisco Meraki has given me insight in this framework. First I will give some general remarks on the framework, followed by detailed comments on cases that are not complete or sound. Based on these observations I have created a new framework named Extension of Rhee's Framework, since it is largely based on Rhee's work.

Rhee's framework [27] consists of 17 cases that are described in a very general manner. Sometimes it is not clear whether the case has PASSed or FAILed the test. A more concrete explanation of the cases will improve the consistency of the tests. Also, to be able to refer to cases, numbers should be added so they can be referenced to. In the detailed comments on all the cases below I have already done this. Furthermore the two possible results for a case, PASS and FAIL, are not sufficient in my eyes. A new type named WARNING should be added to tag states that PASS the test, but may cause additional risks. The last general remark is that there are no Android specific cases in Rhee's original work: these will be incorporated in the final version of ERF.

Case 1: Audit data generation. "The MDM agent should generate audit data", but what actually falls within this category is not discussed in detail. In ERF SIM/IMEI alterations and authentication failures are explicitly named as well as a (basic) roadmap to check this.

Case 2: User authentication. In this case the enforcement of authentication policies is tested. In Android this enforcement is done by the Device Management API. Checking whether the authentication policies apply fall under the policy overview case in MDM-E.

Case 3: Device locking before user authentication. In Rhee's Device locking before user authentication step he verifies that the device can not be accessed without the proper login details. He FAILs the case-study performed, since ADB can access the file-system. The way he executes this test-case is not about authentication but about USB-debugging, which is treated separately in ERF.

Case 4: Authentication failure. "The MDM agent should have countermeasures when user fails a certain number of authentication attemps." As said in the incompleteness in case 2: this is enforced by the API and should thus not be checked separately.

Case 5: Session locking. This test case is about the locking of the device after a certain interval. Just as case 2 and 4 this falls in the Policy category in ERF.

Case 6: Data encryption. Encryption is enforced by the OS. The policy should be reviewed in regards to the implementation. Also if encryption is enabled, it is good practice to change the encryption key such that it is no more reliant on the user-supplied PIN/password.

Case 7: Data integrity. This case is very relevant. If you can alter the controls enforced by the MDM, this would severely damage the system. When performing this case on Cisco Meraki I was able to find the configuration file, but editing this file did not result in changed policies. This is probably the case because the API-calls are only performed when the app receives commands from the server and the saved settings are only used to be reflected in the application. In ERF this question gives more directions than the framework by Rhee does.

Case 8: Device locking. This case should only be about the ability of the administrator to remotely lock the device. Rhee once again FAILs the test due to USB-debugging, which in my opinion does not match the case description.

Case 9: Device wiping. Just as the previous case, this should only be about the ability of the administrator to wipe the device. In ERF this is taken into account to verify that the developers of the MDM correctly called the right API-function.

Case 10: Encryption key and cryptographic data management. This is a very good case especially on Android. Since every application can be reverse engineered without much effort we can search the source code for relevant information about the protocols the MDM uses and if the developer is not security aware, we may be able to retrieve encryption keys. In ERF this case has been expanded with keywords that may be of interest during this case.

Case 11: Transferred data encryption. This case is to generic. If the data is encrypted with a Ceasar cipher it would PASS the test, but it does not provide any security whatsoever. In ERF it is specified that SSL should be used and if another protocol is used a WARNING will be thrown.

Case 12: Connectivity. A connectivity test should be performed to see whether the MDM has control over the devices it should be controlling. A good practice is to disconnect the device for two hours and after-worths verifying if the MDM system has detected this.

Case 13: Device control. On Android the only hardware that can be disabled by the MDM is the camera. This falls under the policy implementation category in ERF.

Case 14: Application installation, removal, execution, termination. The method of the application (dis-)allowance should be included in this case. A blacklist can for instance provide some protection, but it is seldom complete. In the revised version of the framework, having only a blacklist should result in a WARNING. A whitelist that is not deceivable should be a PASS.

Case 15: Protection of an MDM agent. When the MDM is disabled from the device the connectivity-test will yield the same result as Rhee's test for protection of the MDM agent and is therefore redundant.

Case 16: Anti-malware interworking. On Android versions 4.2 and higher the built-in malware-scanner is sufficient.

Case 17: Modification detection. This method basically describes rooting, but does not name it as such. The MDM should also be tested to see whether it can detect foreign rooting methods: MDM-E tests the device for multiple rooting techniques.

The discrepancies between the companies policy, the MDM implementation and current Best Practices do not play a part in Rhee's framework. Beside the incompletenesses listed above, none of the vulnerabilities specific for Android are taken into account in Rhee's work. The security relevant settings (USB-debugging and Unknown sources); exported Intents; the presence of a brute-forceable encryption key and injection attacks are for instance not included in the current framework. Also the enrollment protocol is not part of the tests

performed in the framework. Therefore I propose ERF, which can be found in the appendix.

The Extension to Rhee's Framework consists of 23 cases, split in seven fields:

1. Enrollment (two cases).
2. Policy (two cases).
3. Device settings and audit data (five cases).
4. Data protection (five cases).
5. Secure communication (one case).
6. Application management (two cases).
7. MDM implementation (five cases).

A couple of cases from Rhee's framework have been redesigned to meet the Android specific threats, others have been combined and several have been added. This framework gives the security expert a road-map to perform his work. In section 4.2.1 the threats for MDM as described by Rhee were introduced. The cases from ERF are linked to the following threats, including the new threat "Policy flaws":

#	**Threat**	**ERF related case(s)**
1	Disclosure	5.1, 7.2, 7.3, 7.4
2	Software	3.1, 3.2, 3.3, 6.1, 7.1, 7.2, 7.3
3	Bypass	3.1, 3.5, 4.2, 4.5, 7.2, 7.3
4	Data authenticity	4.2, 7.3
5	Data transfer	1.1, 5.1
6	Traffic	1.1, 5.1
7	Spoof	1.1, 1.2, 5.1, 7.2
8	Malware	3.1, 3.2, 3.3, 6.2, 7.1, 7.2, 7.3
9	Denial of Service	7.2
10	Leakage	1.2, 3.1, 3.2, 4.1, 4.5, 7.2, 7.3, 7.4
11	Record	3.4, 3.5, 4.2, 7.4
12	Disaster	Not within scope.
13	Zero-day	7.1
14	Policy flaws	2.1, 2.2

Table 5: ERF cases related to threats to MDM as described in [35].

To ensure that execution of the Extension of Rhee's Framework, as proposed in this paper, is repeatable and verifiable a clear list of allowed tools needs to be used. These tools are all publicly available via the Google Play store or

present in the Android SDK, with the exception of MDM-E which has been discussed earlier. These tasks are all freely available to make sure that there are no obstructions executing ERF. The following tools are proposed to be used for the following tasks.

Tools on device	**Description**
MDM-E	Mobile Device Management Evaluator - allows the penetration-tester to quickly summarize the enforced policies, checks for dangerous settings and exported intents.
ProxyDroid	Enables a proxy on the device to connect with Burp.
SQLite Editor	Can open and modify the SQLite databases present on the device.
Shark for Root	Enable the pen-tester to capture network traffic which can be read with Wireshark. Only works on rooted devices.
Tools on PC	**Description**
Burp	Proxyserver which allows the pentester to route all traffic through the program.
Dex2Jar	Part of the .apk-decompiling set. Translates a .apk into a .jar.
JD-GUI	Displays the content of a .jar.
ADB	Android Debug Bridge is used to communicate with the mobile device via USB.
Wireshark	Can analyze the .pcap-files obtained with Shark for Root.

Table 6: Tools allowed to be used when executing ERF.

At the beginning of May 2014, I informed Rhee of my revised version of his framework. I sent him the current version of ERF together with the thinking process behind it, hoping to receive comments on it. Unfortunately he was unable to respond before this paper's deadline.

6.3 Mobile Device Management Evaluator (MDM-E)

The MDM-E application that has been built during this thesis can be installed to any device that supports the Android Device Management API. The source code for the application can be found on GitHub [2]. Other researchers are encouraged to expand and correct MDM-E. Once it is installed on an Android device the penetration-tester has the option what test he wants to perform, as shown in the following picture.

[2]GitHub: `https://github.com/JKremers/MDM-E`

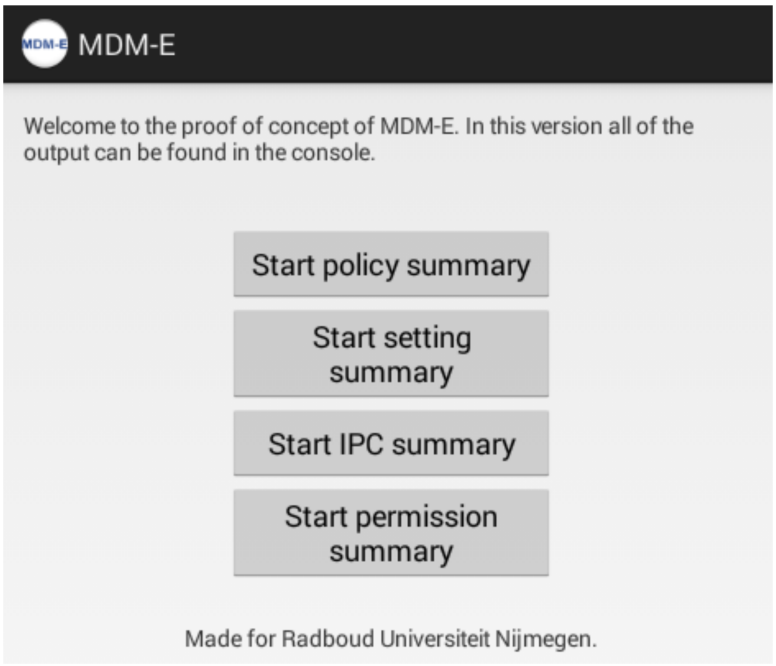

Figure 13: The main screen in MDM-E

Once the penetration-tester selects the case he needs to execute, the result is outputted to the console. In the case of the policy test, MDM-E prints twenty lines that all have relevant information about the MDM in it. This includes what MDM is installed, its version and the policies it is enforcing. In the example result in Figure 14, we have identified Meraki System Manager version 0.9.47. It is furthermore disabling the camera and forcing the user to have a password of nine characters, with at least one symbol and one number. Multiple steps from the framework are easier to perform using the information provided by MDM-E.

```
com.mdme    System.out    ActiveAdmin name: com.meraki.sm
com.mdme    System.out    ActiveAdmin versionName: 0.9.47
com.mdme    System.out    Policy summary:
com.mdme    System.out    Camera disabled: true
com.mdme    System.out    Keyguard flags: 0
com.mdme    System.out    # before wipe: 0
com.mdme    System.out    Time before wipe: 0
com.mdme    System.out    Password expiration in: 0
com.mdme    System.out    Password history length: 1
com.mdme    System.out    Minimum password length: 9
com.mdme    System.out    Minimum lowercase: 0
com.mdme    System.out    Minimum nonletter: 0
com.mdme    System.out    Minimum numeric: 1
com.mdme    System.out    Minimum symbol: 1
com.mdme    System.out    Minimum uppercase: 0
com.mdme    System.out    Password quality: 262144
com.mdme    System.out    Encryption: false
com.mdme    System.out    Encryption: 0
com.mdme    System.out    Is active: true
com.mdme    System.out    ---------------------
```

Figure 14: The output in the console created by MDM-E

When performing a case study with MDM-E multiple exported intents were found in the Cisco Meraki MDM. After manual investigations I found one exported intent that can be exploited. The relevant part of the Android Manifest that was vulnerable was the following:

```
<activity android:label="@string/locate_device_name" android:name=".LocateDeviceActivity" ar
            <intent-filter>
                <action android:name="com.meraki.sm.locate.SHOW" />
            </intent-filter>
</activity>
```

The activity .LocateDeviceActivity can be called without any permissions. In this case this activity is the function that triggers an alarm. There is no reason for this activity to be called by any other than the application itself. By adding the line *exported=false* you can make sure that the activity is only accessible to internal components. Since this intent can be sent out by everyone, an attacker can perform the intent spoofing attack as described in the Technical Background by launching the activity. A very basic application can exploit this by simply starting the activity. I have devised a more elaborate attack, which exploits the bug in the following manner:

1. The attacker creates an application that the target may want to install. This application requires two permissions: Internet to display the fake

content of the application and the permission to start on boot. The URL of the .apk is distributed to the victim.

2. The victim installs the malicious .apk.

3. When the user starts the application an activity is started that looks like a legitimate component. This is to make sure the victim does not expect any malicious actions performed by his new application.

4. When the application is paused or exited, the logo of the malicious application is removed from the application menu for obfuscation purposes.

5. The malicious content of the application starts when the device is rebooted. The BOOT_COMPLETE command is sent throughout the device.

6. The broadcast receiver component listens to this BOOT_COMPLETE command and starts our listening service.

7. When the user presses the Screen ON/OFF hardware button, the ACTION_SCREEN_OFF intent is broadcasted.

8. Our service listens to this intent and triggers the vulnerable component of Meraki, which rings the alarm.

The graphical overview of the malware can be found in Figure 15. One of the assumptions of this attack is that the user has the Meraki System Manager installed, which has been downloaded between 50.000 and 100.000 times. Also Unknown sources need to be allowed, which between 56 percent and 80 percent of the users have activated according to my survey. The target also needs to agree the requested permissions, which between 40 percent (from my own survey) and 83 percent [11] of the targets do without question. Half of the people having Unknown sources allowed, do not pay attention to the permission. The users that actually read and understand the permissions may also allow the app to install, since it does not directly cost the target any money (like SMS-messages do). This leads to multiple thousands of targets. The service that is triggering the .LocateDeviceActivity will keep on running until the malware has been removed from the device.

Every time the Screen ON/OFF hardware button is pressed, the alarm activity is started. This exploit makes the device very difficult to use and the

battery is drained since the screen can only be turned off by waiting for the automatic screen lock that occurs after a set time. Since the alarm activity is a legitimate activity of Meraki, it is unlikely that they will link this annoying behavior to the malware described above. The consequence of this may be that the Meraki helpdesk will be flooded or (even worse) that companies and individuals lose confidence in Meraki.

The bug and the proof of concept as described above have been reported to Meraki on the 24th of February in line with their Responsible Disclosure agreement. On the 18th of March Meraki reacted to confirm the issue. After a month a member of the security team responded that my finding "is both obnoxious and a bug". A few days later they thanked me for my help, but have decided this does not fall within their reward program. Given the worked out scenario above and the far stretching implications this has for the application on devices all around the world and the company itself, this surprises me. I have decided to publish this vulnerability, since it has been multiple months since I have reported it - and provided a solution to fix it. At the time of writing this bug has not been fixed. Meraki is aware of this publishing and does accept this. The vulnerability found shows that MDM-E can be applied to real life MDM implementations and that it has already contributed to a safer MDM landscape on Android.

Several steps were taken to hide the malicious intentions of the application:

- A legitimate looking activity is used, this can easily be tailored to the target: e.g. if he is into Bitcoins make the app display the current Bitcoin share price or when he is a sports fan give him the soccer rankings.
- The application is removed from the app-drawer to impede deletion. The application can be deleted from the device from the Apps tab in the Settings menu.
- The malicious components are dormant until the device is rebooted. This way the target will not suspect the actual malware to be the cause of the problems his device is facing.

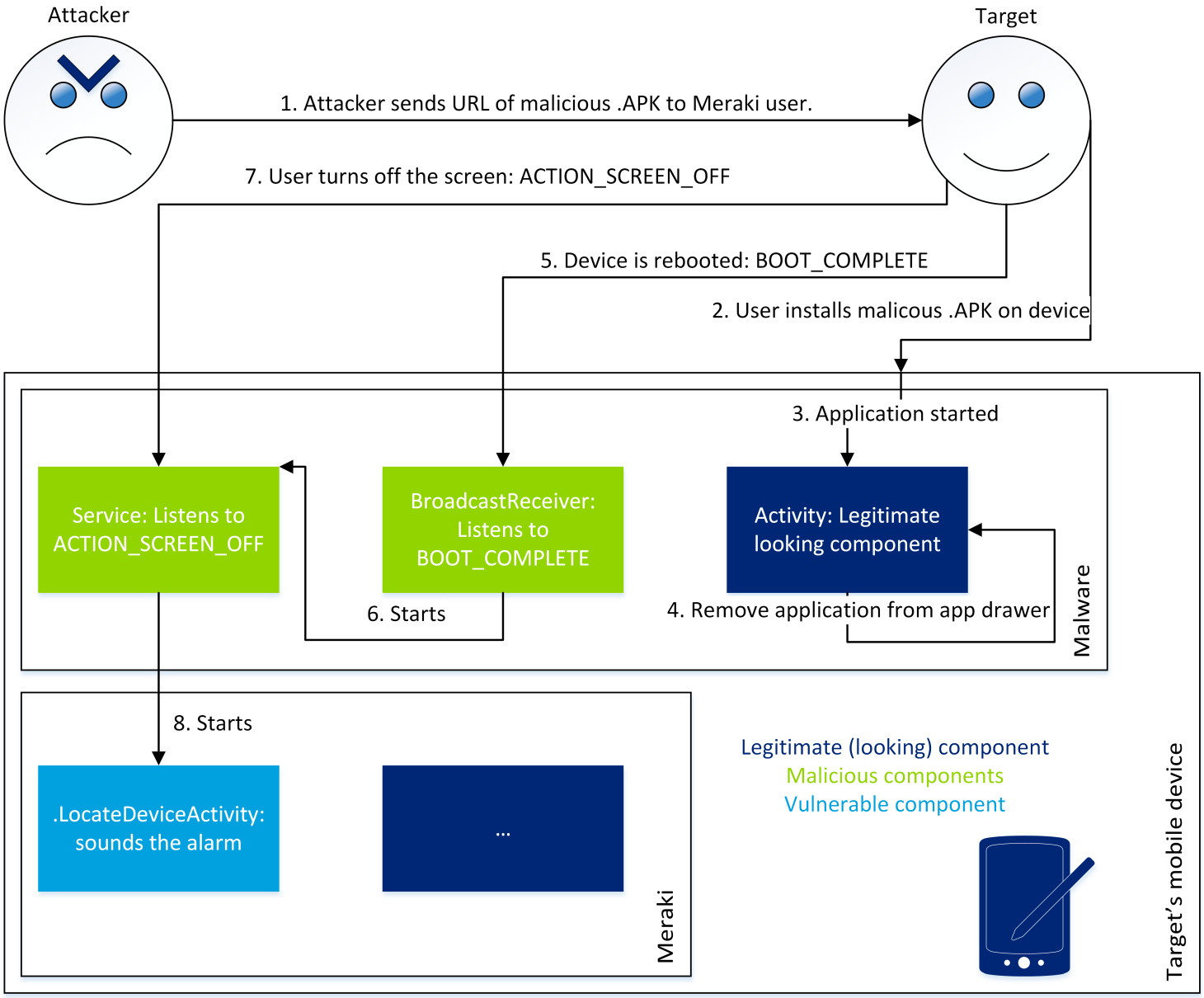

Figure 15: An overview of the PoC exploiting an unprotected intent in the Meraki MDM system.

7 Discussion

In this section I will draw the conclusion from my results and discuss where possible imperfections to my research approach are. Following this I will give some directions to fellow researchers in need of a thesis topic - or possible subjects to explore myself.

7.1 Conclusions

RQ: How do Mobile Device Management solutions operate on Android?
In this paper I have shown what is under the Android hood: a Linux kernel and a Linux based permission-model. I have discussed Android and the vulnerable areas that an attacker can exploit: security settings, IPC spoofing, rooting and encryption-flaws. Most of the functionality that a MDM can use on Android is enforced by the OS. The Device Administrator API allows people to easily design their own MDM. Many companies now market a MDM that can be used on Android, but their maturity-level is rather low. Especially when considering the threats to MDM and to the Android OS.

SQ1: How are mobile devices used in a corporate environment? And what are the risks?
The survey that was performed for this research gives a general idea on how people use their mobile devices in a corporate environment. Companies seem to be restrained when it comes to providing Android mobile devices, especially when you compare it to personally owned devices (26 percent versus 43 percent). Most of the devices are used for calling and email, while reporting is done on a seventh of the devices. Although 81 classify some data on their device as confidential, only 54 percent of the respondents are aware of the mobile policies in place at the company. The personally owned devices are at more risk than their company owned counterparts: a higher percentage has Unknown sources enabled and more than double the amount is rooted. A total of 32% have a Mobile Device Management system in use, with market-leader MobileIron on the top of the chart.

SQ2: What method is there to structurally test a MDM implementation on Android? Is this method complete and correct?
Basic frameworks for mobile application testing lack MDM specific cases and the MDM specific evaluation framework designed by Keunwoo Rhee shows some incompletenesses as shown in the Results-section. I have adjusted the incompletenesses as I encountered them and added Android specific cases. The Extension of Rhee's Framework (ERF) includes a list of tools and a roadmap that can be taken when performing a security evaluation test for a MDM system. The 23 cases allow the security expert to work in a structured and verifiable manner.

SQ3: Is it possible to automate (part of) this security-testing process?
Yes, the ERF framework is largely implementable for Android. Part of this

framework is already incorporated in an application called MDM-E that the security expert can use to automate seven cases. In multiple other cases MDM-E can assist with obtaining a result. It is able to give information on the installed MDM; verify the enforced policy on the device; detect vulnerable security settings; investigate if a device is rooted and find exported IPC components. I have also suggested several extensions to MDM-E that would automate more tasks the security expert otherwise has to perform manually. MDM-E has already demonstrated its usefulness by exposing a security vulnerability that can be exploited as shown in this paper.

7.2 Discussion

The survey performed for this research had 146 respondents out of which 118 answered the first question. The largest share of respondents was from the IT-sector, which I explained by the method of finding respondents. This large share of respondents may give a biased result for my survey. When repeating a similar survey it would be wise to have a completely random sample group, that has much more respondents to decrease the error margin. This can be achieved by either hiring a marketing company to distribute the survey or to make the survey more appealing by adding a (potential) reward.

Asking every respondent for their contact details would also be a wise idea. In my survey I only asked them for their details if they wanted to receive a copy of my final thesis. I would have liked the ability to contact several respondents that filled in unexpected answers. Does someone for instance really have a 23 character pincode? And what does the company need rooting for to achieve more security on the device?

The combination between Rhee's MDM evaluation framework and the Android specific cases that were investigated in this paper seem like a good starting point when testing a MDM implementation. It can be used as such, but its maiden voyage will show its applicability in practice. It has been discussed with two security professionals, but due to time constraints it was not case-tested yet. Adjustments can and must be made to ERF to make it accurate and up-to-date.

7.3 Future work

Most MDM implementations rely on the Simple Certificate Enrollment Protocol to create the link between the MDM server and client. A vulnerability to the Simple Certificate Enrollment Protocol has been shown in [23]. There was just one attempt to formalize this algorithm [24], but this did not include all elements specifically relevant for MDM device enrollment. Formalizing SCEP in for instance SPI-calculus, in regards to MDM implementations, and verifying it with Proverif can give a great insight in potential unknown weaknesses to the

algorithm.

The extended mobile device management evaluation framework proposed in this paper is specific for Android. Some parts may currently be relevant for other OS's, while others are Android specific cases and can be altered to be applicable to iOS or Windows Phone. It is worth creating a similar framework for the other popular operating systems. It may also be an idea to create one general framework, with OS specific cases in it.

MDM-E has implemented part of the extended framework proposed in this paper. The PoC for MDM-E covers a variety of tests, including root-detection and policy enumeration. There are however some areas that need further investigation before they can be added to MDM-E. Part of the connectivity and data integrity tests can potentially be included in application. Also the proposed server-element has not been developed, but can provide the penetration-tester with a good overview of the results of the security test. The last step to be taken to make MDM-E available to the public would be to export it to the Google Play store.

References

[1] bhilgeman. [solved] update bypass mobileiron and root detection ics, jellybean. http://forum.xda-developers.com/showthread.php?t=1611861. Accessed on March 28th 2014, 2012.

[2] B. Bosscher. Steal whatsapp database (poc). http://bas.bosschert.nl/steal-whatsapp-database/. Accessed on May 10th 2014, 2014.

[3] D. Brodie. Practical attacks against mobile device management (mdm), 2013.

[4] T. Cannon. Into the droid - gaining access to android user data, 2012.

[5] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2011.

[6] Cisco. Annual security report 2014, 2014.

[7] A. Cox. An analysis of mobile geofencing app security. Technical report, 2014.

[8] A. Cozzette. Intent spoofing on android. http://blog.palominolabs.com/2013/05/13/android-security/, 2013.

[9] R. Van der Meulen & J. Rivera. Gartner says smartphone sales grew 46.5 percent in second quarter of 2013 and exceeded feature phone sales for first time. http://www.gartner.com/newsroom/id/2573415, 2013.

[10] N. Elenkov. Changing android's disk encryption password. http://nelenkov.blogspot.nl/2012/08/changing-androids-disk-encryption.html. Accessed on February 20th 2014, 2012.

[11] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 3. ACM, 2012.

[12] Google. Android security overview. http://source.android.com/devices/tech/security/index.html. Accessed on March 10th 2014.

[13] Google. Device administration. http://developer.android.com/guide/topics/admin/device-admin.html. Accessed on April 11th 2014.

[14] Google. Manifest.permission. http://developer.android.com/reference/android/Manifest.permission.html. Accessed on March 10th 2014.

[15] Google. Notes on the implementation of encryption in android 3.0. http://source.android.com/devices/tech/encryption/android_crypto_implementation.html. Accessed on March 10th 2014.

[16] Google. Dashboard on march 3, 2014. http://developer.android.com/about/dashboards/index.html. Accessed on March 3th 2014, 2014.

[17] E. Gruber. Android root detection techniques. https://www.netspi.com/blog/entryid/209/android-root-detection-techniques. Accessed on March 10th 2014, 2013.

[18] E. Gruber. Bypassing airwatch root restriction. https://www.netspi.com/blog/entryid/192/bypassing-airwatch-root-restriction. Accessed on March 1st 2014, 2013.

[19] T. Henderson. How mobile device management works. http://www.itworld.com/mobile-wireless/163465/how-mobile-device-management-works, 2011.

[20] J. Janssen. Enterprise mobile security. Master's thesis, Utrecht University, November 2013.

[21] A John. What is rooting on android? the advantages and disadvantages, february 15, 2011.

[22] Litra. Mobile device usage and document security survey results. Technical report, Litra, 2013.

[23] M. Orlando & A. Manion. Vulnerability note vu#971035. http://www.kb.cert.org/vuls/id/971035, 2012.

[24] Fabio Martinelli, Marinella Petrocchi, and Anna Vaccarelli. Automated analysis of some security mechanisms of scep*. In *Information Security*, pages 414–427. Springer, 2002.

[25] OWASP. Dangers of jailbreaking and rooting mobile devices. https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Dangers_of_Jailbreaking_and_Rooting_Mobile_Devices.

[26] OWASP. Owasp mobile security project. https://www.owasp.org/index.php/OWASP_Mobile_Security_Project.

[27] K. Rhee. *A Study on the Security Evaluation of a Mobile Device Management System*. PhD thesis, Sungkyunkwan University, April 2012.

[28] Keunwoo Rhee, Sun-Ki Eun, Mi-Ri Joo, Jihoon Jeong, and Dongho Won. High-level design for a secure mobile device management system. In *Human Aspects of Information Security, Privacy, and Trust*, pages 348–356. Springer, 2013.

[29] J. Rice. Samsung exynos 4 exploit discovered: Root and full access to ram possible in a single app. http://www.androidpolice.com/. Accessed on February 21st 2014, December 2012. Search for: Samsung Exynos 4 Exploit.

[30] RSA. *Realizing the mobile enterprise*, 2012. Report based on discussions with the Security for Business Innovation Council.

[31] R. Sasi. A weekend with cisco meraki bug bounty, a tale of few web bugs. http://www.garage4hackers.com/content.php?r=157-A-weekend-with-Cisco-Meraki-Bug-Bounty-a-tale-of-few-web-bugs. Accessed on May 20th 2014, 2013.

[32] F. Sturrus. Berekening steekproefgrootte. http://marktonderzoek.punt.nl/content/2006/12/berekening-steekproefgrootte. Accessed on March 15th 2014, 2006.

[33] Timothy Vidas, Chengye Zhang, and Nicolas Christin. Toward a general collection methodology for android devices. *digital investigation*, 8:S14–S24, 2011.

[34] J. Wiewiora. Building an effective mobile device management strategy for a user-centric mobile enterprise, 2012.

[35] K. Rhee & W. Jeon & D. Won. Security requirements of a mobile device management system. *International Journal of Security and Its Applications*, 6(2):353–358, 2012.

[36] Yajin Zhou and Xuxian Jiang. An analysis of the anserverbot trojan. Technical report, Tech. Rep., 9 2011.[Online]. Available: http://www. csc. ncsu. edu/faculty/jiang/pubs/AnserverBot Analysis. pdf, 2011.

A Appendix: Survey

Mobile Devices in a Corporate Environment

Q1: This survey is part of the Computer Science Masters Thesis that Joost Kremers is writing for the Radboud University Nijmegen and Deloitte Netherlands. In this survey you will be asked to answer questions regarding the usage of mobile devices in a professional environment. In this survey the definition of mobile devices is limited to smartphones, smartwatches and tablets. Laptops are excluded from the definition.
Please answer the questions as honest as possible - the outcome of the survey is used to obtain an overall view of mobile device usage. Your individual responses are not used - nor will they be linked to you or your company. The first part is about mobile devices that your employer has provided - the second part on personal devices used for work. The survey takes around 5-10 minutes. Thank you for cooperating.

Q2: My company (roughly) has...
☐ <10 employees.
☐ 10-50 employees.
☐ 51-250 employees.
☐ 251-1000 employees.
☐ 1001-5000 employees.
☐ >5000 employees.

Q3: I work in the following sector:
☐ Agriculture.
☐ Industrial.
☐ Construction.
☐ Trade, Transport and Catering.
☐ Information Technology (IT).
☐ Financial.
☐ Real estate.
☐ Business services.
☐ Government.
☐ Education.
☐ Healthcare.
☐ Culture and Recreation.
☐ Retail.
☐ Other

Q4: How would you rate your technical (computer) knowledge?
☐ Very poor
☐ Poor
☐ Moderate
☐ Good
☐ Very good

Q5: Did your employer provide you with a mobile device? If your employer provides multiple devices, answer questions from the perspective of the most frequently used device. ☐ Yes. ☐ No.
Q6: Who is the owner of this device? ☐ I am. ☐ My employer is. ☐ I do not know/unclear.
Q7: What do you use this mobile device for? ☐ Calling/texting. ☐ Social networking. ☐ Connecting to the corporate network. ☐ Reporting. ☐ Storing data. ☐ Gaming. ☐ Information gathering. ☐ E-mail. ☐ Other
Q8: What is the operating system of the mobile device? ☐ Android. ☐ BlackBerry. ☐ iOS. ☐ Windows Phone/Windows Mobile. ☐ I do not know. ☐ Other
Q9: Does your company have strict rules about mobile devices? ☐ Yes ☐ No
Q10: When unlocking my device, I have to enter... ☐ A pincode (numeric). ☐ A password (alphanumeric). ☐ A biometric lock (camera/fingerprint). ☐ A lock-pattern. ☐ An other lock. ☐ Nothing.
Answer If When unlocking my device, I have to enter... A pincode (numeric). Is Selected Or When unlocking my device, I have to enter... A password (alphanumeric). Is Selected Or When unlocking my device, I have to enter... A lock-pattern. Is Selected **Q11: The length of this lock is ... numbers/characters.**
Q12: Do you store any confidential data (competitively sensitive/customer/private) on the device? Note: e-mail can be confidential too. ☐ Yes

☐ No

Q13: Is your mobile device rooted/jailbroken? If yes, what is the main reason?
☐ Yes, to install a custom ROM.
☐ Yes, to install third party markets.
☐ Yes, to circumvent policies implied by the Mobile Device Manager.
☐ Yes, other.
☐ No.
☐ I do not know.

Q14: Does your company make use of a Mobile Device Management system? If yes, which? Note: If you are not sure, check the applications for the names listed in the answers.
☐ Yes, MobileIron.
☐ Yes, AirWatch.
☐ Yes, Fiberlink.
☐ Yes, Zenprise.
☐ Yes, Good Technology.
☐ Yes, Boxtone.
☐ Yes, IBM.
☐ Yes, SAP.
☐ Yes, Symantec.
☐ Yes, Meraki.
☐ Other
☐ No.
☐ I do not know.

Answer If What is the operating system of the mobile device? Android. Is Selected
Q15: What version of Android is installed?
☐ 2.3
☐ 3.0
☐ 4.0
☐ 4.1
☐ 4.2
☐ 4.3
☐ 4.4
☐ Other.
☐ I do not know.

Answer If What is the operating system of the mobile device? Android. Is Selected
Q16: Can you install applications from unknown sources (from outside of the Google Play store)?
☐ Yes.
☐ No.

Answer If What is the operating system of the mobile device? BlackBerry. Is Selected
Q17: What version of BlackBerry is installed?
☐ 5.0
☐ 6.0

☐ 7.0
☐ 7.1
☐ 10
☐ Other.
☐ I do not know.

Answer If What is the operating system of the mobile device? Windows Phone/Windows Mobile. Is Selected

Q18: What version of Windows Mobile/Windows Phone is installed?

☐ 7
☐ 7.5
☐ 7.8
☐ 8
☐ Other.
☐ I do not know.

Answer If What is the operating system of the mobile device? iOS. Is Selected

Q19: What version of iOS is installed?

☐ 3
☐ 4
☐ 5
☐ 6
☐ 7
☐ Other.
☐ I do not know.

Q20: Do you read the permissions an application requests when you are installing it/starting it for the first time?

☐ Yes, I read and understand them.
☐ Yes, I read them but I do not now what they mean.
☐ What permissions?
☐ No, I do not read them.

Q21: Do you have any personally owned mobile devices that you sometimes use for corporate tasks (i.e. e-mail, logging into the corporate network)?

☐ Yes
☐ No

Q22: What do you use this mobile device for?

☐ Calling/texting.
☐ E-mail.
☐ Social networking.
☐ Connecting to the corporate network.
☐ Reporting.
☐ Storing data.
☐ Gaming.
☐ Information gathering.
☐ Other

Q23: What is the operating system of the mobile device?

☐ Android.

☐ BlackBerry.
☐ iOS.
☐ Windows Phone/Windows Mobile.
☐ I do not know.
☐ Other

Q24: When unlocking my device, I have to enter...
☐ A pincode (numeric).
☐ A password (alphanumeric).
☐ A biometric lock (camera/fingerprint).
☐ A lock-pattern.
☐ An other lock.
☐ Nothing.

Answer If When unlocking my device, I have to enter... A pincode (numeric). Is Selected Or When unlocking my device, I have to enter... A password (alphanumeric). Is Selected Or When unlocking my device, I have to enter... A lock-pattern. Is Selected
Q25: The length of this lock is ... numbers/characters.

Q26: Do you store any confidential data (competitively sensitive/customer/private) on the device? Note: e-mail can be confidential too.
☐ Yes
☐ No

Q27: Is your mobile device rooted/jailbroken? If yes, what is the main reason?
☐ Yes, to install a custom ROM.
☐ Yes, to install third party markets.
☐ Yes, other.
☐ No.
☐ I do not know.

Q28: Do you have any Mobile Device Management system installed on your device? If yes, which? Note: If you are not sure, check the applications for the names listed in the answers.
☐ Yes, MobileIron.
☐ Yes, AirWatch.
☐ Yes, Fiberlink.
☐ Yes, Zenprise.
☐ Yes, Good Technology.
☐ Yes, Boxtone.
☐ Yes, IBM.
☐ Yes, SAP.
☐ Yes, Symantec.
☐ Yes, Meraki.
☐ Other
☐ No.
☐ I do not know.

Answer If What is the operating system of the mobile device? Android. Is Selected

Q29: What version of Android is installed?
☐ 2.3
☐ 3.0
☐ 4.0
☐ 4.1
☐ 4.2
☐ 4.3
☐ 4.4
☐ Other.
☐ I do not know.

Answer If What is the operating system of the mobile device? Android. Is Selected
Q30: Can you install applications from unknown sources (from outside of the Google Play store)?
☐ Yes.
☐ No.

Answer If What is the operating system of the mobile device? BlackBerry. Is Selected
Q31: What version of BlackBerry is installed?
☐ 5.0
☐ 6.0
☐ 7.0
☐ 7.1
☐ 10
☐ Other.
☐ I do not know.

Answer If What is the operating system of the mobile device? Windows Phone/Windows Mobile. Is Selected
Q32: What version of Windows Mobile/Windows Phone is installed?
☐ 7
☐ 7.5
☐ 7.8
☐ 8
☐ Other.
☐ I do not know.

Answer If What is the operating system of the mobile device? iOS. Is Selected
Q33: What version of iOS is installed?
☐ 3
☐ 4
☐ 5
☐ 6
☐ 7
☐ Other.
☐ I do not know.

Q34: Do you read the permissions an application requests when you are installing it?
☐ Yes, I read and understand them.

□ Yes, I read them but I do not now what they mean. □ What permissions? □ No, I do not read them.
Q35: Do you have any additional information you would like to provide?
Q36: This was the final question of this survey. If you want to receive a copy of the final version of this thesis, fill out you e-mailadress in the text box below. Thank you for your cooperation.

B Appendix: Extension of Rhee's Framework (ERF)

On the next page the Extension of Rhee's Framework is presented in detail. This includes the roadmap and the tools that are needed to perform the test. It is crucial to take notes when executing this framework.

To give an idea of the changes made to Rhee's framework I will in detail explain the changes made to one case. Case 6 on Data Encryption in Rhee's framework has only one step: Check that the MDM agent provides data encryption functions. For Android this is not necessary, since the OS provides the encryption functionality. This means that if the API-call by the MDM agent is handled correctly - the storage on the device is encrypted. The roadmap that is set out in ERF provides steps that install MDM-E; the implemented policies are requested by MDM-E; and the security vulnerability from section 4.1.2 in Android Encryption is taken into account. This leads to a more complete and comprehensive test-case. Since on Android many of the cases rely on an API call, they are grouped together in ERF under the implementation matches policy case (2.2). Besides these alterations several Android specific cases have been added. Examples of this are case 3.2 on USB Debugging and case 7.2 on Intent protection.

Extension of Rhee's Framework	
1. Enrollment	
Case 1.1	Is the enrollment protocol sound?
Criteria for acceptance	Investigate if the enrollment protocol is implemented in a reasonable secure way. If SCEP: is the privilege escalation attack not applicable?
Test tool	Burp, ProxyDroid, Dex2Jar, JD-GUI.
Method	1. Find out what enrollment protocol is used, via either documentation; setting up a proxy with Burp; or reverse engineering. 2. Burp: use Burp and ProxyDroid to set up a proxy with the PortSwigger certificate. Enroll to the MDM server. 3. Dex2Jar: To obtain Android Manifest: apktool d <file.apk> <directory to output to> a. To obtain java-files: d2j-dex2jar -f -o <output.jar> <input.apk> b. To display java-files from .jar use JD-GUI
Result	
Case 1.2	Is the user authenticated before he can enroll?
Criteria for acceptance	If an admin needs to approve the new device: PASS. If user has password that authenticates the request: PASS. If the MDM accepts all new devices: FAIL.
Test tool	-
Method	1. Enroll a new device to the MDM server. 2. Analyse if the enrollment succeeds and if data is pushed to the device.
Result	
2. Policy	
Case 2.1	Policy matches Best Practice
Criteria for acceptance	Manually verify that the policy matches the Best Practice. If it matches PASS. If the policy is stricter PASS.
Test tool	Manual testing.
Method	1. Request the company's information security policy. 2. Compare with Best Practices.
Result	
Case 2.2	Implementation matches policy
Criteria for acceptance	If there are no discrepancies between the policy and the implementation PASS.
Test tool	MDM-E.
Method	1. Request the company's information security policy. 2. Enable Unknown sources on the device (Settings -> Security -> Unknown sources). 3. Push MDM-E.apk to the device (e-mail, link, ADB). 4. Press "Policy summary" button. 5. Compare output in console with policy.
Result	
3. Device settings and audit data	
Case 3.1	Root detection
Criteria for acceptance	If the device is rooted and the MDM reports it: WARNING. If the device is rooted and the MDM does not report it: FAIL.
Test tool	MDM-E.
Method	1. Enable Unknown sources on the device (Settings -> Security -> Unknown sources). 2. Push MDM-E.apk to the device (e-mail, link, ADB).

	3. Press “Settings summary” button. See output in console
Result	
Case 3.2	USB debugging
Criteria for acceptance	If the device has USB-debugging enabled and the MDM reports it: WARNING. If the device has USB-debugging enabled and the MDM does not report it: FAIL.
Test tool	MDM-E.
Method	1. Enable Unknown sources on the device (Settings -> Security -> Unknown sources). 2. Push MDM-E.apk to the device (e-mail, link, ADB). 3. Press “Settings summary” button. See output in console
Result	
Case 3.3	Non-Playstore applications
Criteria for acceptance	If the device allows non-Playstore applications to be installed and the MDM reports it: WARNING. If the device allows non-Playstore applications to be installed and the MDM does not report it: FAIL.
Test tool	MDM-E.
Method	1. Enable Unknown sources on the device (Settings -> Security -> Unknown sources). 2. Push MDM-E.apk to the device (e-mail, link, ADB). 3. Press “Settings summary” button. See output in console
Result	
Case 3.4	Audit data generation
Criteria for acceptance	If the status of the device is correctly recorded and transmitted to the MDM server: PASS. Points of interest: SIM/IMEI alteration and authentication failures.
Test tool	-
Method	1. Alter device settings and try to log in with the wrong credentials. 2. Check on the MDM dashboard if these are detected.
Result	
Case 3.5	Connectivity
Criteria for acceptance	If the MDM reports a device being offline for a certain amount of time: PASS. Else: FAIL.
Test tool	-
Method	1. Put the device on airplane-mode (Settings -> More -> Airplane-mode). 2. Wait 2 hours and see if it is displayed on the MDM dashboard.
Result	
4. Data protection	
Case 4.1	Data encryption
Criteria for acceptance	The MDM should enforce encryption of the storage. If encryption is enabled, but the standard encryption key is used: WARNING. If the encryption key is changed to something not derived from the user’s password: PASS.
Test tool	MDM-E.
Method	1. Enable Unknown sources on the device (Settings -> Security -> Unknown sources). 2. Push MDM-E.apk to the device (e-mail, link, ADB). 3. Press “Policy summary” button. See “Encrypted” in console. 4. If True, restart device and check if the encryption password is the same as the PIN.

	Note: normally the encryption key is derived from the users PIN/password. In case this PIN is short, it can be cracked by an attacker within a reasonable time and even offline. It is therefore good practice to change the encryption key, granting a key that is more secure against brute force attacks.
Result	
Case 4.2	Data integrity
Criteria for acceptance	Modifications or removal of configuration and audit data on the device should be detected or ignored. If the MDM detects change - or changes are not reflected by the MDM: PASS.
Test tool	ADB, Sqlite Editor.
Method	1. Find the application name (com.*). This may be done with MDM-E. 2. Enable USB-debugging (Settings -> Developers options -> USB-debugging). 3. Type “adb shell” in a shell. 4. Browse to the /data/<applicationname> folder and search for a settings file. 5. Edit data. 6. Verify if this has any effect on the MDM.
Result	
Case 4.3	Device locking
Criteria for acceptance	The administrator should be able to remotely lock the mobile device.
Test tool	-
Method	1. Unlock the device. 2. On the MDM dashboard, find the test-device. 3. Locate the lock-button. 4. Verify the screen is locked.
Result	
Case 4.4	Device wiping
Criteria for acceptance	The administrator should be able to remotely wipe the mobile device.
Test tool	-
Method	1. Try to remotely wipe the device. 2. Verify that the device has actually reset to factory defaults.
Result	
Case 4.5	Encryption key and cryptographic data management
Criteria for acceptance	The MDM does not have any encryption keys or cryptographic data hardcoded in it.
Test tool	Dex2jar, JD-GUI.
Method	1. Dex2Jar: To obtain Android Manifest: apktool d <file.apk> <directory to output to> a. To obtain java-files: d2j-dex2jar -f -o <output.jar> <input.apk> b. To display java-files from .jar use JD-GUI 2. Search for keywords like “password” and “key”.
Result	
5. Secure Communication	
Case 5.1	Transferred data encryption
Criteria for acceptance	If the app communicates with the server via SSL: PASS. If the app communicates in a different encrypted method: WARNING. Else: FAIL.
Test tool	Shark for Root, Wireshark.

Method	1. Install Shark for Root. 2. Run Shark for Root. 3. Force the MDM to synchronize. 4. Verify the .pcap-file with Wireshark and look for non-encrypted packets.
Result	
6. Application management	
Case 6.1	Applications
Criteria for acceptance	Applications should only be installed if the MDM allows it. Blacklist and not deceivable: WARNING. Whitelist and not deceivable: PASS.
Test tool	Manual testing.
Method	1. Attempt to install a prohibited application for blacklist-test. 2. Attempt to install MDM-E for whitelist-test.
Result	
Case 6.2	Anti-malware
Criteria for acceptance	The app performs continues checks whether the anti-malware app is present. If the app is disabled it tries to activate it and reports to the server.
Test tool	-
Method	1. Check the installed App list for anti-malware applications. Note: Android 4.2+ has a build-in malware scanner.
Result	
7. MDM implementation	
Case 7.1	Known vulnerabilities
Criteria for acceptance	There are no known vulnerabilities for the evaluated version of the MDM.
Test tool	MDM-E.
Method	1. Enable Unknown sources on the device (Settings -> Security -> Unknown sources). 2. Push MDM-E.apk to the device (e-mail, link, ADB). 3. Press “Policy summery” button. Look at the version name of the MDM application. 4. Search the internet (CVEDetails.com & OSVDB.org etc.) for known vulnerabilities on this MDM in combination with the version name.
Result	
Case 7.2	Intent protection
Criteria for acceptance	The MDM app is not vulnerable to intent spoofing.
Test tool	MDM-E.
Method	1. Enable Unknown sources on the device (Settings -> Security -> Unknown sources). 2. Push MDM-E.apk to the device (e-mail, link, ADB). 3. Press “IPC summary” button. See output in console
Result	
Case 7.3	Injection
Criteria for acceptance	The MDM app is not vulnerable to injection attacks.
Test tool	Manual testing.
Method	1. Check whether there are text-fields that are vulnerable to XSS attacks. (i.e. <script>alert(“XSS”)</script> 2. Check whether you can perform path traversal attacks.

	3. Check whether there are text-fields that are vulnerable to SQL(-Lite)-injection. (i.e. ‘ OR ‘1’=’1)
Result	
Case 7.4	Unnecessary logging
Criteria for acceptance	Verify that no sensitive data is logged.
Test tool	ADB.
Method	1. Enable USB-debugging (Settings -> Developers options -> USB-debugging). 2. Type “adb logcat” in a shell. 3. Force the MDM to synchronize. Verify that no sensitive data is logged before or at the time of syncing. 4. Verify that there is no logfile created on the SDCard or in the /data-partition.
Result	
Case 7.5	Secure revocation
Criteria for acceptance	The MDM has the ability to revoke a client to connect to the MDM server.
Test tool	Manual testing.
Method	1. Try to revoke access to a test-device.
Result	