Fraunhofer
AISEC

FRAUNHOFER RESEARCH INSTITUTION FOR
APPLIED AND INTEGRATED SECURITY

ON THE EFFECTIVENESS OF MALWARE PROTECTION ON ANDROID

AN EVALUATION OF ANDROID ANTIVIRUS APPS

RAFAEL FEDLER, JULIAN SCHÜTTE, MARCEL KULICKE

04/2013

Fraunhofer
AISEC

On the Effectiveness of Malware Protection on Android

An evaluation of Android antivirus apps

Version 1.0

Rafael Fedler, Julian Schütte, Marcel Kulicke

Fraunhofer AISEC

Kontakt: fedler,schuette,kulicke@aisec.fraunhofer.de

April 2013

Abstract

Android is currently the most popular smartphone operating system. However, users feel their private information at threat, facing a rapidly increasing number of malware for Android which significantly exceeds that of other platforms. Antivirus software promises to effectively protect against malware on mobile devices and many products are available for free or at reasonable prices. Their effectiveness is supported by various reports, attesting very high detection rates.

However, a more detailed investigation is required in order to understand the real risk level arising from malware for the Android platform. Neither do the exceedingly high numbers of different malware variants reflect the real threat in comparison to other platforms, nor do the results of testing antivirus software against a set of already known malware samples (*retrospective tests*) provide a clear picture of the capabilities and limitations of antivirus software on the Android platform.

The primary objective of this report is thus to help corporate and private users to assess the real risk level imposed by Android malware on the one hand, and the protection level offered by antivirus software on the other hand. For this purpose, we discuss how malware spreads and which limitations antivirus apps are subject to. We then evaluate how well Android antivirus software performs under real-world conditions, as opposed to retrospective detection rate tests. Based on our findings, we give recommendations for private and corporate users and sketch possible future solutions to overcome some of the current issues of antivirus software.

For this report, we conducted various tests on several antivirus apps for Android. As we aim to reflect real-world threats better than retrospective tests, in which antivirus software is tested for recognizing known malware samples, our test setup considers the ability to cope with typical malware distribution channels, infection routines, and privilege escalation techniques. We found that it is easy for malware to evade detection by most antivirus apps with only trivial alterations to their package files.

In order to test different malware detection techniques, we also used a newly developed proof of concept malware. This proof of concept malware demonstrates advanced functionality which is not present in most of today's Android malware, and is intended to determine how Android antivirus software will deal with unknown and upcoming malware.

Contents

1 Introduction **4**

2 Background **6**
2.1 Previous Work 6
2.2 Android Risk Assessment 7
2.2.1 Threat Reports 7
2.2.2 Antivirus Tests: Methodology, Limiting Factors and Evaluation 8
2.3 Distribution Channels 10
2.4 Post-Distribution Techniques 12
2.5 Trends and Future Scenarios 13

3 Antivirus Tests **15**
3.1 Scenarios 15
3.2 Test Cases 16
3.3 Candidates 17
3.4 Custom Malware 19
3.5 Results 20
3.6 Discussion 23
3.7 Related Work 25

4 Potential Remedies **26**
4.1 Currently Applicable Countermeasures 26
4.2 Future Approaches for Improved Malware Protection 28

5 Advisory **30**
5.1 Corporate Usage 30
5.2 Private Usage 31

6 Conclusion **32**

References **33**

1 Introduction

While more and more antivirus products for Android appear and users feel the urge to protect against an increasing number of malware, the overall risk situation on the Android platform is very intransparent. Users, both private and corporate, do not know against which threats antivirus products protect reliably and which one is most effective.
Numerous threat reports, issued primarily by antivirus companies, report ever increasing numbers of unique malware samples [1, 2, 3, 4]. Users may be tempted to think that the malware threat on the Android platform is rising at a fast pace. Contrariwise, detection rates of retrospective tests report very high detection rates for most antivirus apps on the Android platform. More than 25% of antivirus solutions are attested a detection rate of 90% and above, and more than 50% of Android antivirus apps reportedly detect more than 65% of the test samples [5]. Combined, these reports and tests culminate in a quickly increasing perceived threat level on the one hand, and seemingly very high protection rates on the other.

Thus, the overall risk situation for Android users, as a result of 1) the threat level by malware, and 2) the protection level offered by Android antivirus software, is intransparent and not easy to assess. Not only are the aforementioned tests' and reports' conclusions not easy to make, but also interpretations are often not accurate. In this report, we will give a current evaluation of the general Android risk situation in regards to malware, and give special attention to the protection level offered by current antivirus software for Android. We will conduct a number of tests on Android antivirus apps, using samples of known Android malware and Android exploits. We will explicitly not test retrospective detection rates. Instead, we will introduce minor changes to malware samples to test detection of re-distributed malware. We aim to determine detection robustness against hardly altered malware, and if antivirus software is only capable of detecting known samples. Also, we will simulate one typical infection channel – a low-profile dropper app – and the capabilities of detecting privilege escalation exploits, which are used by more than one third of malware to gain unlimited control over a target device [6]. Thus, we will simulate malware re-distribution, infection and privilege escalation scenarios to gain an insight into real-world performance of antivirus software on Android.
To test current Android antivirus software against unknown and advanced upcoming malware, we will also conduct tests using a proof of concept malware developed as part of our previous work [7]. As it is not deployed in the wild, it is unknown to Android antivirus software. It also demonstrates various functionality only seen in the most recent Android malware [8], but which is expected

to be deployed more widely in the future, such as cross-platform infection. No special obfuscation or stealth techniques have been used, and the core malicious functionality could be implemented by malware authors in a similar fashion. On the technical level, this test of an unknown malware threat addresses antivirus apps' behavioral analysis capabilities.

In Chapter 2, we will explore the aforementioned tests and threat reports, and explain why neither the threat, nor the protection level is fully described by them. To allow the reader to better assess the risk situation, we will also provide them with background knowledge about typical distribution channels and malware methodology on Android. Building on previous work [9], we will explore in how far past assumptions and predictions proved to be true, and deduce current trends in malware threats from it.
Chapter 3 contains the results of our real-world tests. Along with the analysis of the current threat situation and considerations about threat reports' and test results' relevance, they are the main part of this publication. They reflect how well Android antivirus software holds up against malware in action, dynamic scenario changes like altering malware packages (which affects signature-based detection), and novel and advanced threats. We deployed known malware samples, exploits, and also a proof of concept malware unknown to antivirus vendors. Where possible, we altered code portions of our test samples. Simulations of dynamic behavior were carried out, as opposed to mostly static tests of malware samples.
After the tests, we will introduce a number of possible improvements in Chapter 4. These improvements address antivirus software, the Android platform itself, and potential new solutions which can be offered as additional services. In the subsequent Chapter 5, an advisory is given to corporate and private users. We conclude the report in Chapter 6.

2 Background

In this chapter, we will analyze claims regarding the malware threat on the Android platform and the protection level offered by Android antivirus software, as reported by antivirus vendors and independent antivirus tests. We will also give a retrospective evaluation of our own predictions made in May 2012, and draw conclusions for current trends in Android malware evolution. For this, and for the better understanding of our subsequent tests in Chapter 3, the reader will also be introduced to typical malware methodology on the Android platform. This includes distribution, infection and core functionality. After this chapter, the reader will be able to assess the risk level on the Android platform better, and understand relevant aspects for real-world effectiveness of Android antivirus software.

2.1 Previous Work

In May 2012, we published a comprehensive report about the general security of the Android operating system, focusing not on theoretical aspects of security, but on practical attack vectors. It also dealt with the resilience of the Android operating system against local privilege escalation attacks and all related consequences [9]. For practical evaluation purposes, we developed a proof of concept app which served as a wrapper for local privilege exploit and payload execution. Using the experience drawn from this practical evaluation and as a result of then current malware, we made a few conclusions and assumptions about future malware threats. This includes propagation/distribution, persistent infection, and core malware functionality.

For one, we concluded that future Android malware can propagate between Android smartphones and desktop PCs. First attempts of this can already be witnessed with Zeus-in-the-Mobile (ZitMo, part of the Zeus malware family which is primarily a banking malware) samples, though they rely mostly on social engineering [10], and in the very recent SuperClean malware, which manages to attack this vector without social engineering [8]. Furthermore, the feasibility of a fully automatic approach has been demonstrated by practical work of one of the authors [7]. For this matter, a full malware has been developed. This same malware, as it is unknown to antivirus vendors, will be used later in this report to conduct antivirus software tests. While most malware strives to gain high infection numbers and thus does not profit from this possibility very much, more specialized attack scenarios may build upon this attack vector. In this context, recent high profile attacks such as *Flame* and the *Red October* network can be counted as examples which also deployed targeted attacks for infection. Under such a scenario, a targeted attack on an employee's Android device may be used to infiltrate networks.

As can be seen, many of our predictions turned out to be correct, either shown by malware samples detected in the wild, or by practical proof drawn through a proof of concept malware. We assume that the trends we observed already one year ago will continue to be pursued by malware authors.

2.2 Android Risk Assessment

The general perception of Android security has been largely shaped by two classes of reports: On the one hand, antivirus vendors – as they have access to the biggest set of malware samples – regularly release threat reports on the state of malware threats for many platforms, including mobile platforms. On the other, magazines, companies, and institutes publish test reports of antivirus product tests. Both reports contribute to the perceived risk level on the Android platform. Risk, in this case, is a function of both the threat and the protection level. However, we identified several issues not addressed in both publication groups. These issues mainly concern the practical implications and conclusions of these reports for users and their devices' security.

2.2.1 Threat Reports

Multiple antivirus vendors release threat reports on a regular basis. Most of the time, these provide information about unique malware samples, which are ever rising [1, 2, 3, 4]. The conclusion often drawn is that the malware threat situation on Android is dramatically increasing, based on these unique malware sample numbers.

However, any minor change in a malware sample makes it considered "unique", as its hash/checksum is then different from all other samples of the same malware family. Thus, there may be dozens or hundreds of unique samples which have completely identical functionality. Minor changes are often introduced into malware to avoid different detection techniques. For example, single bits and character strings can be changed to avoid checksum-based detection, executable obfuscation may be applied to circumvent signature- or heuristic-based detection, and so on. No statement about the real number of infections, nor about malware developers' and distributors' efforts to spread their malware, nor about attack frequency can be made. General device vulnerability is also not taken into account. Only the number of new, marginal different malware samples in the wild is counted. Such counting is often considered not very valuable, as it does not reflect the real malware threat situation. [11]

Much more important for threat assessment is the number of devices susceptible to certain attacks or distribution channels, or actual infection numbers, or new malware families. These are sometimes examined, but mostly not focused on.

Alongside with those threat reports, press releases often warn of the increasing threats for Android users. However, we argue that the threat situation is not accurately depicted by those threat reports. Actual new threats and vulnerabilities are often not analyzed in detail, while rising unique sample numbers are used as a basis for warnings of increased threats.

2.2.2 Antivirus Tests: Methodology, Limiting Factors and Evaluation

One of the most notable and complete Android antivirus software tests up to date has been performed by AV-Test [5]. It deploys the same techniques as used for antivirus tests on other platforms such as Microsoft Windows. Most importantly, this is the benchmarking of the retrospective detection rate of all products. While this approach already does not accurately depict the protection level offered by each product at the time of a new malware threat's emergence – as it is retrospective – there are more issues on the Android platform, which make these test methodologies less appropriate for the assessment of the protection level offered by the examined products.

The Android platform has fundamentally different approaches for controlling software access to operating system, device, and other program's resources. On desktop platforms, most software is a priori considered trusted and has, once installed, far reaching access to the system's, the users' and other software's data on that system. On Android, however, a file system based sandbox ensures that each installed app may only access its own data and not that of the user or other apps, unless explicitly permitted by the user (through the well-known permission system [12]). Android antivirus software cannot even list other directories' contents on an Android device.

Furthermore, antivirus software on Windows has the capability of monitoring file system operations. This way, if a software which has not been conspicuous before suddenly downloads malicious code to the harddrive[1], this code will be detected nonetheless, as the download to the harddrive will be noticed by the antivirus software's on-access scanner. On Android, however, filesystem monitoring is also not possible, as necessary techniques such as hooking are not allowed to user-installed apps.

Thus, neither full file system scans, nor filesystem monitoring can be implemented by antivirus software on Android. This has grave implications. Otherwise harmless apps may start downloading executables to their working directories and run them – a technique not controlled or prohibited by Android. This cannot be detected by Android antivirus software. Hence, these attacks cannot be

[1] a technique commonly deployed by droppers, which themselves do not demonstrate malicious behavior and thus do not get detected by antivirus software easily

covered by retrospective detection rate tests of Android antivirus at all. Such attacks are feasible and can be easily deployed. High detection rates reported by antivirus tests, however, suggest a near-perfect protection of devices with detection rates of 90% and above.
In the case of a widely spread malware family with many development iterations, it did not get detected by antivirus software at all at the time of its release, leading other researchers to criticize antivirus software on Android for low effectiveness[2].

Also, like all retrospective tests, Android antivirus tests fail to reflect how quickly the examined products detect new malware threats. On Android, this is even of bigger importance than on desktop platforms such as Microsoft Windows, as successful infection cannot be noticed by users easily. Removal of malware may even be completely impossible for almost all users, as it might require the reinstallation of the device's software image. This would be necessary, for example, if malware gains elevated privileges and installs itself to the system partition of the device, which is only readable by all other software. Thus, timely reaction to new threats is of utter importance, as device reinstallation is often not possible, and malware can often remain unnoticed because of antivirus' limitations on the Android platform.

Ultimately, Android antivirus software has to resort to two primary sources of information for malware detection:

1. Package database[3]
2. Package files (APK files) of installed apps

The package database stores package names, and also package file locations. As Android antivirus software cannot list the contents of the directory with installed packages, it has to rely on the package database to provide it with the locations of installed packages. These package files can then be read to be checked using typical antivirus detection techniques. All files added after installation, however, remain invisible to antivirus software on the Android platform.

All of the above lead us to the conclusion that retrospective detection rate tests of Android antivirus software do not reflect the real protection level offered by such antivirus software. The only threat it can protect against is known and not very advanced malware, using package names and package files of installed packages. Any activity after installation cannot be controlled or detected by Android antivirus software. Dynamic downloading of malicious components after installation can – for this exact reason – be expected to be increasingly used by malware in the future.

[2] "In fact, when the first version of DroidKungFu was discovered, it has been reported that no single existing mobile anti-virus software at that time was able to detect it, which demonstrated the "effectiveness" of this approach." [6]

[3] stored in `/data/system/packages.xml`

2.3 Distribution Channels

To understand the methodology we applied in our tests in Chapter 3, to gain a better insight into the problems antivirus software faces on the Android platform, and to understand Android malware methodology, we will provide background information on infection channels.

Generally, any data transmission and communication channel can serve as an attack and malware distribution vector. Channels which are not meant for transmitting software may still be taken control of by typical techniques for assuming control over any app's or service's control flow as a result of erroneous programming. These vectors include USB, bluetooth, and NFC connections, barcodes, QR codes, unsecured wireless connections which can be exploited to inject data, erroneous GSM/UMTS/LTE radio package handling, and many more. Typical communication channels which are designed to carry software are the official Google Play Store and third party app markets.

In the following, we will focus on the most important infection channels for typical malware which can be found in the wild today. For more in-depth information on Android propagation channels, persistent infection, and malware methodology in general, refer to our previous work [9].

App Markets The official Android app market is fairly well controlled. While techniques exist to circumvent this, the Google Bouncer dynamic heuristic malware detection service exists to protect the official Android market, called Google Play. Google employees also have the option to manually take off malicious apps from the market and even remotely wipe it from devices. Pirated and non-sophisticated malware gets removed fairly quickly and well-known and easily detectable malware does not get admitted to the Google Play Store at all.

Third party app markets, however, are much less well monitored. Often they are run by a community which does not have access to facilities like Google Bouncer or other malware detection solutions well-suited for Android. Not rarely, these markets are not well monitored on purpose: Free access to otherwise paid software is considered a feature by many users of these markets.

This makes such markets very attractive for malware distributors: They can download apps which would usually have to be paid, add malicious code to these apps, and then upload them again to app markets. Due to less control, third party markets are predestined for this. Also, users are purposefully looking for such pirated apps. Thus, these so-called *repackaged* apps are of very high attractiveness to users, and hence also to malware developers and distributors. The degree of their attractiveness also correlates with the functionality the apps offer. The more functionality, the more permissions are needed for the app without letting the user get suspicious. These permission are then also available to the malware.

Rooting Some users "root" their phone, meaning they make the root account on their phone always accessible. Root access is normally not possible for end-users or end-user software. Rooting one's device is necessary for some modifications, some installed software, and often for flashing alternative operating system images to a device. In order to grant root access to apps, the `su` utility is often installed on rooted devices. This utility can also be used by malware, which then does no longer need to run privilege escalation exploits on its own to acquire root privileges.

Thus, rooting one's own device, just like using third party app markets, can introduce higher risks.

PC-to-Device and Device-to-PC Infections As deemed theoretically feasible by our previous work [9], and proved in practice by our proof-of-concept malware [7] and recent malware [8], infections from PCs to Android devices and vice versa are possible using USB.

The direction from Android devices to PCs can be achieved using vulnerabilities in operating system components, such as USB drivers or the file explorer. The latter (CVE-2010-2568[4]) has been exploited by the Stuxnet malware and also by our proof of concept malware. A recent Android malware spotted in the wild is the first to our knowledge to actively attack Windows PCs [8].

Propagation from PCs to Android devices can be achieved most easily via Android Debug Bridge (adb). First, malware can display forged update notifications, telling the user to activate Android Debug Bridge (adb) for the update to be installed. Then, the malware active on a PC may start an adb daemon and install malware on the Android device. Similar infection is already attempted by the Zeus-in-the-Mobile (ZitMo) malware, though it is mostly conducted using social engineering. ZitMo tries to achieve cross device infections presenting its targets with forged SMS text messages prompting them to install an update, also providing the link allegedly pointing at the update itself. The update is an APK file the user has to download and install manually.

However, more automated infection is definitely possible and can be used by malware in the future. The latest ZitMo samples and the aforementioned "SuperClean" malware [8] confirm this trend.

Cross-platform infection effectively opens up the possibility of hybrid botnets comprised of Android and Windows (or other desktop platform) bots, which is of high interest since smartphones are often used as a second factor in a two-factor-authentication scenario, e.g. for mTAN in online banking applications.

Furthermore, Android devices, in the same manner USB sticks have already been used for targeted attacks against corporate networks, can be used for the same objective. Using any distribution channel suitable, the Android device may be

[4] *http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568*

infected beforehand. If it is then plugged into a USB port inside the target network, it may then commence to infect the host it is plugged into. Thus, Android devices may be used to infiltrate corporate networks, without their owner's knowledge.

2.4 Post-Distribution Techniques

After malware has initially been installed to a target device, it deploys different techniques to reach its final modus operandi. These techniques depend on how it was initially spread to a device, and what the core functionality of the malware is. In general, malware seeks to acquire the privileges necessary to carry out its desired purpose.

Some malware may not need additional steps after its initial installation to a target device. This applies for example when it already has been installed with all necessary permissions, e.g. if installed manually by the user due to social engineering, proper disguise, or carelessness. This is also true if the malware has been installed via adb (through a PC, or more uncommonly, through another Android device), as this does not require user confirmation. Such behavior is common, for example, with simple SMS receiving or sending trojans which do not request a very alarming set of permissions.

Other malware tries to keep a low profile and not to raise any user suspicion by not formally requesting any permissions necessary for its core functionality, or may be unable to acquire necessary permissions due to a permission's high *protection level*. Such malware then needs to deploy privilege escalation techniques. Usually, they use root exploits to this end. These provide them with unlimited access to target devices, which malware may use for irremovable infection[5], installation of other apps (in which case the initial malware serves as a dropper), among other use cases. Malicious components and exploits are often downloaded at runtime and thus invisible to antivirus software. More than one third of all malware collected by the Android Malware Genome Project makes use of such exploits [6].
This kind of malware can be considered more advanced. It is more versatile, more powerful, and sneakier, as it does not need to request any other permission than `INTERNET` or, for some exploits, `READ_LOGS`. On the other hand, as of now most current Android versions (2.3.4 and greater & 4.0.4 and greater) are not prone to local standalone[6] root exploits. However, new vulnerabilities leading to root exploits will probably be discovered in the future, as they have been in the past.

Especially the latter case of more advanced malware directly profits from the deficiencies of antivirus software on Android. The root exploits can and – for

[5]to a device's system partition; for details, see [9]

[6]Other exploits may work with USB access. "Standalone" refers to exploits which can be run without any external help.

improved stealthiness – often will be dynamically downloaded at runtime [6]. As Android antivirus cannot monitor dynamic behavior of other apps and working directories' contents, antivirus software is completely oblivious to such activities.

Dynamically Downloaded Code A technique increasingly deployed by malware due to its advantages in detection evasion is the dynamic downloading of code, for example by RootSmart [13]. In essence, any app on Android may download executables containing native code and run those. As a result, malware apps with completely harmless functionality may serve as disguised malware droppers. They cannot be detected by traditional antivirus techniques on Android due to the shortcomings detailed in earlier sections. Android does not control or limit the executability of native code files, which has already been addressed in our previous work [9]. Any app may download arbitrary files and run them, enabling them to download root exploits for privilege escalation and malicious payloads.

In the proof of concept malware developed by one of the authors, the payload in this case was a script, which is being run with root privileges and then installs the core malware to the target device. This way, the dropper may implement any cover functionality and cannot be detected. Any malicious code is downloaded at runtime and will remain invisible to antivirus software.

We consider this loophole one of the biggest issues remaining in Android security.

2.5 Trends and Future Scenarios

Google has introduced the Google Bouncer dynamic heuristic malware detection system, and most recently also a reputation-based app warning system. This has different implications. First, the malware problem, though percentally not very widespread on the Android platform, is taken serious by Google itself and thus is probably considered an issue of major importance. Second, the difficulty of distributing malware via the official Android app market "Google Play" has dramatically increased, with a 40% decrease of downloads of potentially malicious apps from the Play Store right after Bouncer's introduction [14].

For malware developers to keep up with these developments and to be able to spread their malware widely, they have to deploy more advanced techniques. The most prominent and already known one is to download and execute native code at runtime, as this is neither controlled or monitored by the Android operating system, nor visible to antivirus software. Thus, we expect this technique to be increasingly used in the future.

Also, we have developed different scenarios for targeted attacks which we deem feasible, and which we have already carried out in a controlled environment using our own proof of concept malware. Amidst recent developments and discoveries of advanced and targeted attacks aiming to infiltrate corporate networks such as

Flame, *Stuxnet* and *Red October*, we consider such attacks very likely to happen in the future.

Current antivirus software is neither able to detect, nor to prevent such attacks, as shown by our tests in the following Chapter 3. However, we have identified and developed different additions to the Android platform. These improve its resilience against attacks and especially against root exploits, and also enable antivirus software to monitor apps' working directories, while still keeping the Android security architecture intact. Another result of our work is an effective app scanner[7] which can be used by corporations to scan apps for malicious code and company policy compliance. All these measures will be presented in the subsequent Chapter 4.

[7] AppRay, *http://www.app-ray.de/*

3 Antivirus Tests

In this chapter, we detail both our test cases and their results and point out the relevance of the test scenarios for real-world conditions. Our goal was to test more realistic ways of malware infection and operation, as opposed to so-called "retrospective" tests, in which antivirus software is tested against already known and unmodified malware sample. For the design of our test cases we thus paid special attention reflect some common techniques used by malware authors to lower the detection rate by antivirus software. As the main focus of evaluation is however to make statement about the "real-world" effectiveness of antivirus apps, we limit our test cases to simple techniques which could be applied even by unexperienced malware authors.

3.1 Scenarios

There are a number of scenarios against which antivirus software should protect users, which were taken as a basis for the design of our test cases. Most notably, minor changes are introduced into samples of known malware to test if antivirus software will still detect such malware. Also, we simulate downloading of malicious code at runtime, and run tests with a custom proof of concept malware which is currenlty unknown to antivirus vendors.

Detection of altered malware In this scenario, malware is installed on a device and then antivirus software tries to detect it. Prior to this, the malware test samples have been slightly modified. Most importantly, in this scenario antivirus software can make use of signature-based detection or even static heuristics on already present package files. Also, cloud-based detection is mostly useful during or shortly after installation as a preemptive measure. This is often named "real-time protection" and is tested by this scenario as well.

This test case reflects easy approaches which can be used by malware authors to repeatedly distribute their malware while evading detection. It demonstrates in how far antivirus software is capable of detecting known malware with only slight alterations.

For comparison, we will also test the detection rate of unaltered malware.

Detection of Altered Root Exploits Similarly to the malware test cases, we explore how antivirus software performs given root exploits which have undergone minor modifications. Alterations are again used to avoid detection. For comparison, we will also determine the detection rates of unaltered root exploits. The issue addressed in this scenario is whether otherwise seemingly harmless apps may easily ship with altered root exploits. This scenario addresses static detection of malicious payloads, while the former one covers dynamic downloading of such payloads and as a result also behavioral analysis.

Detection of Malicious Behavior at Runtime Some malware may not have any malicious components at installation time – in order to prevent detection – but only download malicious components at runtime, as often done by droppers. This scenario is to test whether antivirus software can detect seemingly harmless apps "turning bad". We use a small app to download root exploits to its working directory.

Detection of Unknown Threats Our proof of concept malware is deployed as an example for an unknown threat. We install its dropper, its core component, and an independent USB infection tool which is capable of installing apps on other Android devices given USB access. This is to determine whether antivirus software can detect completely new threats at the time of their release.

3.2 Test Cases

We designed six test cases to reflect different malware behavior and methodology, which address different capabilities of antivirus software.

Some of the tests had to be carried out using repackaging. Android application package (APK) files are signed with their developer's private key to prevent tampering. In order to alter their contents, they had to be unpacked, modified, repacked, and then signed with our own key. After that, they were installed on the test device, either manually or by using the Android debug Bridge (ADB).

The first of our test cases we conducted involved disassembling and reassembling the dalvik bytecode files of the malware samples. Rearrangement of resources and our own renaming schemes automatically lead to changes in file characteristics, which would prevent a full-file signature or checksum-based detection.

1. **Detection of altered malware**: The aim of this test case is to test to which extend the detection capabilities of antivirus apps depends on simple static properties of an app such as its package name or file checksums. Malware application package files are decompiled and their package and class names renamed, but no code is altered. Antivirus software is installed prior to installing the malware on the test device.

2. **Detection of unaltered malware**: Reference values for Test Case 1, to determine how effective only slight alterations are for evading detection.
3. **Detection of altered root exploits**: The purpose of this test case is to evaluate whether antivirus apps can detect slightly modified exploit code, whereas modifications do not affect the functionality of the exploit but comprise insertion of bogus print operations, changes of method names, removal of unnecessary statements, or simply re-compilation with different settings. Such minor changes can be introduced to work around signature- or checksum-based detection. The exploits are placed onto the device as part of an app.
4. **Detection of unaltered root exploits**: Reference values for Test Case 3.
5. **Dynamic downloading**: We let an app download a root exploit to its working directory. This test case is to evaluate how well droppers and other dynamic infection routines are discovered by current antivirus apps.
6. **Advanced and unknown threats**: Our own custom proof of concept malware is used to test antivirus products' heuristic capabilities for detecting unknown, yet typical, threats.

3.3 Candidates

The test candidates have been chosen to represent a fair number of well-known companies:

- avast! Free Mobile Security (2.0.3849)
- AVG Mobilation Anti-Virus Free (3.1.1)
- Bitdefender Mobile Security (1.2.249)
- ESET Mobile Security (1.1.995.1221)
- F-Secure Mobile Security (8.1.11894)
- Kaspersky Mobile Security Lite (9.36.28)
- Lookout Security & Antivirus (8.10.1-9e3ede2)
- McAfee Mobile Security (2.3.1.559)
- Norton (Symantec) Mobile Security Lite (3.3.0.892)
- Sophos Mobile Security (2.0.870 (5))
- Trend Micro Mobile Security (3.0)

We chose only free or free-to-test versions of Android antivirus apps, as paid apps usually do not offer additional detection capabilities according to [5].

Malware test samples are taken from the following families:

- AnserverBot
- DroidKungFuUpdate
- FakeInst
- GingerMaster

- JiFake
- OpFake
- Plankton
- SpyEye-in-the-Mobile (SpitMo)
- SuperClean
- Zeus-in-the-Mobile (ZitMo)
- Our own proof of concept malware

The malware families have been chosen according to their prevalence in the wild on the one hand, and their impact on the mobile threat landscape. For the first category, we selected for example OpFake and FakeInst (cf. e.g. [1] for numbers on malware prevalence). In the latter category, we chose most notably ZitMo and SuperClean. Other families such as GingerMaster have been selected because they target Android versions which are not very recent, but still widely in use[1].
As can be seen, malware has been chosen to demonstrate a wide range of characteristic properties, malicious behavior and functionality to present antivirus software with a high degree of variety.

Most of the malware samples have been provided by the Android Malware Genome Project [15, 6]. Some samples were taken from "ContagioMiniDump", a blog for exchanging mobile malware samples [16]. Sample validity has been confirmed using VirusTotal[2].

The exploits used for some of our test cases are:

- Exploid
- GingerBreak
- RageAgainstTheCage

They have been extracted from the package files of malware samples. The source codes for modifications to the exploits have been obtained from various sources. We chose these three exploits as they have been most commonly used by malware samples in the wild [6]. Many other exploits can only be used via USB access – a distribution channel not yet widely used by malware for cross-platform infections [9].

[1] C.f. *http://developer.android.com/about/dashboards/index.html* for versions market share
[2] *http://www.virustotal.com*

3.4 Custom Malware

As pointed out before, a custom proof of concept malware has been developed as part of our previous work [7]. As it is not to be found in the wild, it cannot be detected using signature tests, with the exception of some components it utilizes. These components, which are privilege escalation exploits, though, are downloaded dynamically at runtime to the malware's working directory and thus cannot be detected by antivirus apps.
This malware directly addresses antivirus apps' capabilities of detecting unknown, new threats based on static heuristics/behavior analysis. No obfuscation techniques have been implemented in the proof of concept malware to hide its behavior.

Its functionality is described by several requirements which have been formulated prior to its development. Some of the most notable, among others, are:

- Propagation
  - From PC to Android smartphone
  - From Android smartphone to PC
  - From Android smartphone to Android smartphone
- Infection
  - Circumvention of Google Bouncer
  - Privilege escalation
  - Irremovable permanent installation
  - Start upon device boot
- Core functionality
  - Communication protocol for command transmission
  - Credential extraction from other apps' local databases
  - Sending of SMS
  - Network-based denial of service attacks
  - Sending of spam emails
  - Extraction of a target device's contacts from the address book
  - Interception of incoming mTANs

The proof of concept malware's interaction with its environment and the command-and-control-server are described by Figure 3.1.

As can be seen, typical malware methodology has been applied for the development of this proof of concept malware. Some of its functionality can currently be considered advanced, such as cross-platform infection. In general, however, it openly demonstrates typical malware functionality. Thus, our proof of concept malware is well suited for testing antivirus apps' performance in detecting unknown threats.

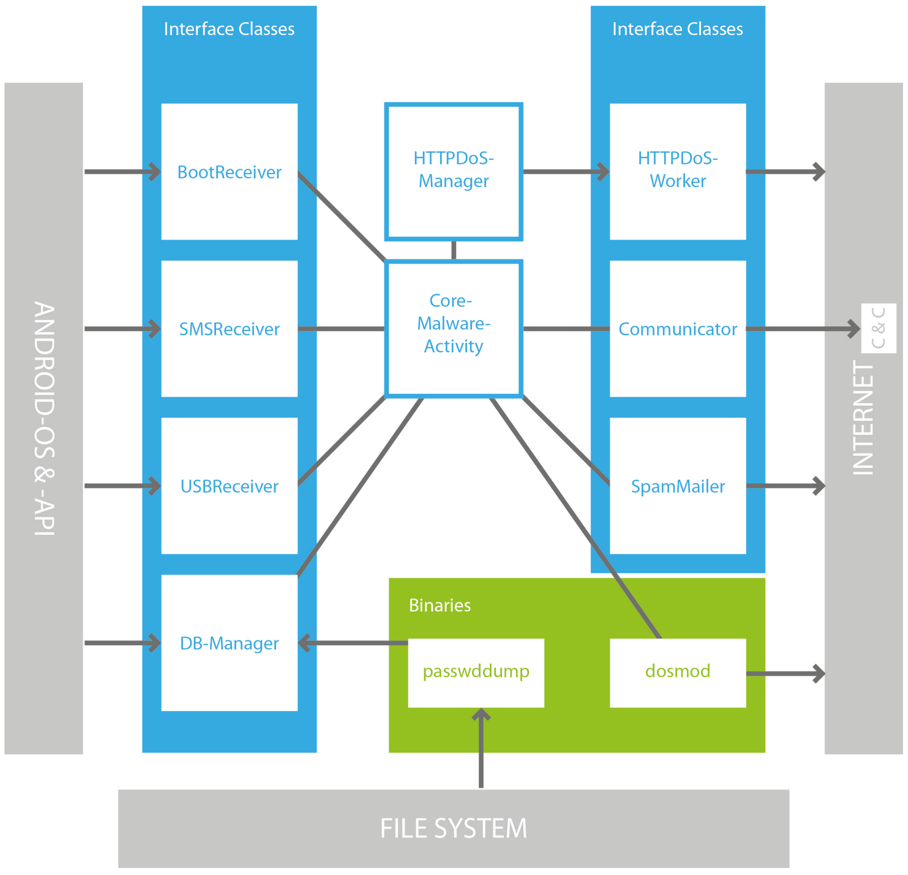

Figure 3.1: Interaction of our custom proof of concept malware with its environment [7]

3.5 Results

In the following, we present the results of our tests.

For the sake of better presentation, malware names have been replaced by numbers. The legend is as follows:

1. AnserverBot
2. DroidKungFuUpdate
3. FakeInst
4. GingerMaster
5. JiFake
6. OpFake
7. Plankton
8. SpitMo (SpyEye-in-the-Mobile)
9. SuperClean
10. ZitMo (Zeus-in-the-Mobile)

Test Case 1: Altered Malware

	1	2	3	4	5	6	7	8	9	10	Total
avast	✓	–	✓	✓	–	✓	✓	–	✓	–	6/10
AVG	–	–*	–	✓	–	✓	–*	–	–	–	2/10
BitDefender	✓	–	–	✓	–	✓	✓	–	–	–	4/10
ESET	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	9/10
F-Secure	✓	–	✓	✓	–	–	✓	✓	✓	–	6/10
Kaspersky	✓	–	–	✓	–	–	✓	–	–	–	3/10
Lookout	✓	–	–	✓	–	✓	✓	✓	✓	✓	7/10
McAfee	✓	–	–	✓	–	–	–	–	–	–	2/10
Norton	✓	–	–	✓	–	✓	–	–	–	–	3/10
Sophos	–	–	–	–	–	–	–	–	–	–	0/10
Trend Micro	✓	–	–	✓	–	–	–	–	✓	–	3/10

Table 3.1: Detection rates for Test Case 1 (–* denotes that the sample has been detected as aggressive adware, not as malware)

Test Case 2: Unaltered Malware (Reference Values for Test Case 1)

	1	2	3	4	5	6	7	8	9	10	Total
avast	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10/10
AVG	✓	–*	✓	✓	✓	✓	–*	✓	✓	✓	8/10
BitDefender	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10/10
ESET	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10/10
F-Secure	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10/10
Kaspersky	✓	–	✓	✓	✓	✓	✓	✓	✓	✓	9/10
Lookout	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10/10
McAfee	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10/10
Norton	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10/10
Sophos	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10/10
Trend Micro	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	9/10

Table 3.2: Detection rates for Test Case 2, which serve as reference values for Test Case 1 (–* denotes that the sample has been detected as aggressive adware, not as malware)

Test Case 3: Altered Root Exploits

	Exploid	GingerBreak	RageAgainstTheCage	Total
avast	–	✓	–	1/3
AVG	–	–	–	0/3
BitDefender	–	–	–	0/3
ESET	–	–	–	0/3
F-Secure	–	–	–	0/3
Kaspersky	–	–	–	0/3
Lookout	✓	✓	–	2/3
McAfee	–	–	–	0/3
Norton	–	–	–	0/3
Sophos	–	–	–	0/3
Trend Micro	–	–	–	0/3

Table 3.3: Detection rates for Test Case 3

Test Case 4: Unaltered Root Exploits (Reference Values for Test Case 3)

	Exploid	GingerBreak	RageAgainstTheCage	Total
avast	✓	✓	✓	3/3
AVG	–	–	–	0/3
BitDefender	–	–	–	0/3
ESET	✓	✓	✓	3/3
F-Secure	✓	–	✓	2/3
Kaspersky	✓	✓	✓	3/3
Lookout	✓	✓	✓	3/3
McAfee	✓	✓	✓	3/3
Norton	–	–	–	0/3
Sophos	–	–	–	0/3
Trend Micro	✓	✓	✓	3/3

Table 3.4: Detection rates for Test Case 4, which serve as reference values for Test Case 3

Test Case 5: Dropper Downloads Root Exploits at Runtime

	Exploit Download Detected	Total
avast	–	0/1
AVG	–	0/1
BitDefender	–	0/1
ESET	–	0/1
F-Secure	–	0/1
Kaspersky	–	0/1
Lookout	–	0/1
McAfee	–	0/1
Norton	–	0/1
Sophos	–	0/1
Trend Micro	–	0/1

Table 3.5: Detection rates for Test Case 5: A dropper downloads root exploits at runtime

Test Case 6: Unknown Malware

	Dropper	Core Malware	USB Infection Tool	Total
avast	–	–	–	0/3
AVG	–	–	–	0/3
BitDefender	–	–	–	0/3
ESET	–	–	–	0/3
F-Secure	–	–	–	0/3
Kaspersky	–	–	–	0/3
Lookout	–	–	–	0/3
McAfee	–	–	–	0/3
Norton	–	–	–	0/3
Sophos	–	–	–	0/3
Trend Micro	–	–	–	0/3

Table 3.6: Detection rates for Test Case 6: Detection of a dropper, the related malware, and a standalone USB infection tool; all of them unknown as there are no samples in the wild

3.6 Discussion

Our test cases, most notably Test Case 1 (*Altered malware*), show very well how easy most antivirus products can be evaded by minor alterations to malware. This does not even require advanced code obfuscation techniques or alterations to code at all. Changing characteristics only present in an app's manifest file, its strings file, and package names often sufficed. The weakness of the approaches of current antivirus scanners is also stressed by the fact that sometimes no alterations had to be applied to root exploits – recompiling them with newer compiler versions sufficed to evade detection by many products. To evade almost all, however, we decided to change method names, removing some irrelevant output statements, and inserting some print statements into the exploit code. All of these modification do not change the malicious character of the exploit code and should thus not influence detection by antivirus apps.

It is important to point out that our tests do not claim to be complete or to reflect detection rates of a large number of malware samples. Our test cases are only intended to explore how easily even well-known malware can evade antivirus software through only minor changes, and how antivirus software copes with malware samples which are not 100% identical to the ones that have been observed in the wild before.

Furthermore, as expected, droppers are a major problem. Apps which only serve to download malicious components but do not openly display any malicious behavior themselves are not detected at all. The dropping process itself cannot be detected due to Android's sandboxing model and as confirmed by Test Case 5.

The tested antivirus apps were also not able to detect malware which is completely unknown to date (as we used a non-public custom malware), but does not make any efforts to hide its malignity. This shows that the tested antivirus apps do not provide protection against customized malware or targeted attacks. Consequently, once a new threat emerges, users are only secure after the threat becomes known to antivirus vendors. The weakness lies in the lack of sufficiently effective heuristics and behavioral analysis. Naturally, the signature-based approaches applied by most tested virus scanners will not detect new threats.

The implications are that most antivirus apps on Android are not able to detect slightly altered malware, unknown threats, or droppers. If new threats (or modifications of older malware) emerge, the capabilities to detect them depend on the antivirus app's vendor to supply new signatures quickly. Before that, users are without any protection. Given that modifying existing malware and deploying it on a large amount of devices (e.g., through third-party app stores) is not very difficult for the Android platform, the situation is not yet satifying.

Readers shall note that this evaluation should not be taken as a recommendation in favor or against any vendors or antivirus apps. Test cases and apps have been chosen to reflect realistic scenarios and a representative selection of well-known antivirus apps and are not complete. Rather, the results of this evaluation shall raise awareness of the limitations of current antivirus apps on the Android platform.

3.7 Related Work

Rastogi et al. [17] have performed a number of comprehensive tests which address application package and code obfuscation techniques. In these tests, they applied more advanced methodology. Such methodology is already widely used by desktop malware and not hard to deploy for Android malware as well.

Their findings back our results: Minor alterations to malware package files or their bytecode are sufficient to avoid detection by most products. This is also due to shortcomings in detection methodology, as at least 43% of signatures have been found not to be based on code-level artifacts, i.e. only on file names, checksums or binary sequences. Also, only one in ten antivirus products was found to use static code analysis. One product even only used manifest file contents for detection.

4 Potential Remedies

As evident from our tests, antivirus apps on Android still face various difficulties as they mainly rely on signature or name databases instead of code heuristics and behavioral pattern detection. As a result, the tested antivirus apps do not provide a real-time protection against beforehand unknown malware samples or even barely changed known malware. The consequences of such deficiencies can be seen in recent cases, where publicly unknown malware has been successfully deployed in corporate networks for extended periods of time [18]. In these cases, installed antivirus solutions – even on desktop platforms – proved to be oblivious to these threats. In the case of Flame/Flamer/sKyWIper, the malware was probably operating within corporate networks all around the world for five to eight years without being detected by antivirus software [19].

Different countermeasures could be taken to strengthen the general resilience of Android against malware attacks, to increase the effectiveness of Android antivirus software, and to provide private and corporate users with a means to assess app malignity or compliance with corporate policy:

- Offloading malware scanning to the cloud, in order to detect malicious apps prior to installation, or to filter them before admitting them to a trusted app market
- Stricter controls for native code execution, to prevent downloading and execution of malicious code at runtime
- Improving antivirus capabilities thtough a system interface, to increase effectiveness
- Native code hash and signature validation, as a whitelist approach to only allow the execution of trusted native code

These include alterations to the Android platform and also additional, supportive solutions for malware detection which can serve for pre-screening and filtering of potentially malicious apps. The focus of this chapter is not on current antivirus products, but is intended to provide a glimpse at potential future developments and solutions for malware detection and resilience.

4.1 Currently Applicable Countermeasures

On-device security software is currently very limited in its capabilities. Off-device services with improved detection logic, however, can help both private users for scanning of single files, or integrated into advanced services for security scanning or benign usage policy compliance.

Offloading Malware Scanning to the Cloud One way to work around the limitations of on-device malware scanners is to offload the scanning to an Internet-hosted service. This has a number of advantages: at first, no resources are spent on the device itself and thus its battery lifetime and performance is not negatively affected. Second, as the scanning service is not subject to the security model of the device, it is free to apply much more detailed inspection techniques, resulting in higher detection rates especially for unknown malware variants. Finally, when aggregating results from multiple users, the service can gather additional information about infection rates and malware known to be in the wild, which again can improve the detection rate.

Google is working in this direction, using its *Bouncer* service [14] and its *Verify App* feature to externalize malware scanning to Internet-hosted services. The latter system, though, has already been shown to have significant weaknesses [20], while Bouncer can be easily circumvented by more advanced malware or even be attacked itself [21].

Another prominent solution is virustotal[1], an online malware checking service, which has been acquired by Google and might thus be integrated into Android's *App Verification* feature at a later time.

In order to better reflect the specific requirements of corporate usage of mobile devices, we have developed App-Ray[2], a tool which does not only classify apps as malicious/benign, but rather checks if they comply with a user-definable catalog of security requirements. In contrast to virustotal, App-Ray also runs dynamic tests to inspect the behavior of an app at runtime and to learn whether it loads additional code from remote locations. One of its advantages is thus that it may serve as a tool to check apps before admitting them to a trusted, policy-compliant app market which may be used by larger corporations for their devices. Such a trusted app market may also be offered as a service to users.

However, it is important to understand that solely relying on an online scanning service makes only sense if the service is integrated into the market (or some other app deployment process) so apps are scanned before they are installed on the device. As soon as a malicious app is being installed, it would be able to disguise as a benign piece of software before it is being uploaded to the scanning service, thereby rendering the approach void.

[1] *http://www.virustotal.com*
[2] *http://www.app-ray.de*

4.2 Future Approaches for Improved Malware Protection

The measures to be presented in this section are potential alterations to the Android platform. They harden Android at the system level, thus being of more effect than remedial measures such as malware scanning. Such measures may be included by custom Android flavors for higher security requirements.

Stricter Controls for Native Code Execution The ability to run native code is the basis of all known root exploits and is currently not limited by Android. In a previous report, we argued that the execution of native code can and should be limited and controlled [9]. Since then, we developed a concept for securing the Android system against the execution of unwanted native code, which can be implemented without any restrictions on legitimate apps. The concept also prohibits the attack vector of dynamically downloaded code [6], which is used to exploit system vulnerabilities with privilege escalation exploits.

The major advantage with this approach is that the Android system has to be modified only marginally, in comparison to Mandatory Access Control (MAC) extensions like SEAndroid [22], Tomoyo [23], or Miyabi. These extensions allow to monitor and control system calls and file system accesses and are thereby able to prevent root exploits. However, properly configuring a MAC rule set which effectively defeats root exploits and at the same time does not negatively affect usability is difficult. Creating a rule set that comprehensively covers all interprocess communication and resource sharing of all apps is extremely complex and requires massive effort.
An Android smartphone shipping with SEAndroid built in is the Samsung Knox[3]. Using SEAndroid, it does not only increase attack resilience, but also manages to separate private and corporate data in BYOD scenarios, according to its manufacturer.

Improving Antivirus Capabilities Through a System Interface One part of the Android security model is the isolation of processes. What is a security feature turns into a hindrance when antivirus apps are prevented from accessing the filesystem and the memory of other processes.

One way to grant extended access for the sake of virus scanning would be to integrate an *Antivirus interface* into the Android middleware, as we have outlined in our research. Through this interface, legitimate antivirus apps would get access to the working directories of other apps and be able to inspect their memory. Using this privileged interface would require an app to be signed by a trusted party, such as a legitimate antivirus company, whose identity has been confirmed before, e.g. by Google.

[3] *http://www.samsung.com/global/business/mobile/samsungknox/index.html*

Native Code Hash and Signature Validation The number of native libraries preinstalled on an Android device is limited. To ensure that only trusted libraries are used, code signing or checksum comparison can be applied.
With this approach, native code libraries and binaries could be signed with a private key, e.g. from the platform provider, so their integrity be verified at runtime nd possibly matched against a whitelist of known benign libraries. The downside of this approach is that the majority or third party-supplied libraries would be excluded from these checks, as long as they have not been signed with the trusted key as well. Also, it must be considered that attackers with privileged access to the device would be able to tamper with the verification process.
While for corporate usage, a whitelist approach might be acceptable, private users would certainly not be willing to accept limitations of native code execution. Although only 5% of all apps use native code at all [24], the majority of those is games, which would be blocked by this approach.

5 Advisory

In this chapter, we give recommendations for private and corporate usage scenarios. For both groups, it can be said that users and administrators should not solely rely on antivirus software for malware protection. While antivirus software may reliably detect long-known threats [5], its abilities to detect new threats, variants of old threats, droppers, or targeted attacks is limited. Recent reports of very successful malware attacks against companies [18] stress this risk.

5.1 Corporate Usage

Corporations, represented by administrators and IT staff, should not be careless about device security after the installation of antivirus software, which also holds true for desktop computers as demonstrated recently.

Strict device usage policies should be established to facilitate the separation of smartphones and stationary systems. If a bring-your-own-device (BYOD) smartphone policy is in place, this directly equals the separation of corporate and private IT assets, with the latter also being taken outside the company and being used for private means to a significant extent.

As shown by recent attacks which have been launched through devices brought into a company from outside, smartphones need physical connection for simplified, direct compromization of corporate computers. Most commonly, employees will plug in their private devices via USB to charge them, or to exchange data. Corporate policy can be formulated to prohibit plugging in private devices completely, in order to prevent the spreading of malware from smartphones to employee PCs.

Eavesdropping on corporate communication can be hindered by setting up a separate wireless network for smartphones.

The aforementioned measures address the integrity of the stationary corporate IT assets. Ensuring the integrity and security of smartphones themselves, though, is a more difficult task.

A very effective approach at ensuring that employees do not get infected with openly or covertly malicious apps is a tight control over installed apps on their devices. In return for network or mobile email account access, employees can be required to use only a specific app market. In non-BYOD scenarios where employees are supplied with corporate devices, enforcing a specific app market is even easier and can just be preconfigured.

To our knowledge, however, an app market with a very high emphasis on security has not yet been opened. Extensive app security checks, e.g. with tools such as App-Ray [1] can be carried out before transferring them from the Google Play Store. Such an app market could be offered commercially as a service to other companies. Alternatively, companies can also open up their own secure app markets backed by extensive security checks and allowing for the installation only of whitelisted apps. At the same time, alternative app installation sources should be disallowed to prevent circumvention of the app market.

5.2 Private Usage

We consider private users to be at a relatively low risk of malware infections, as long as a healthy level of caution is applied. Targeted attacks are unlikely as they require high efforts by the attacker and do not scale to a large number of targets. The majority of current malware is rather spread via third-party app markets. In comparison to these, the security level of the official Google Play Store is relatively high. Consequently, as long as private users do not root their devices and do not use third-party app markets or other alternative software sources, the overall risk of malware is low. In third party app markets, however, the risk of infection is substantial. Users should avoid to download pirated copies of software that otherwise would have to be paid, for the sake of their own security.

A remaining issue which has also not been solved for corporate users is the typical dropper or update attack scenario. Software which does not openly demonstrate malicious behavior may in fact be a dropper and download malicious components, or, through an update, may be replaced with malware. Bad vendor patch policy, leaving many customers without updates to recent Android versions and thus vulnerable, is at fault for this. Customers should pay special attention to the update policy of manufacturers before buying a new device.

[1] *http://www.app-ray.de/*

6 Conclusion

The most important contribution of this report is the assessment of the effectiveness of current antivirus apps in scenarios which reflect typical malware (re-) distribution and behavior. We evaluated how well they hold up under "non-retrospective" circumstances, i.e. when they cannot rely purely on signatures of known samples, but have to cope with slightly modified or even completely unknown malware. Similar tests were run for root exploits and a dropper has been simulated. We consider our test cases to provide a much more realistic picture, compared to most existing threat reports and antivirus tests, merely referring to unique sample numbers and detection rates of known, unmodified malware.

Our results clearly show that malware sample numbers on the one, and retrospective test results on the other hand are only "half the truth" and do not sufficiently describe the real threat situation and the performance of antivirus software. Threat and protection level need to be assessed more in detail for various scenarios in order to make comprehensive statements about device protection offered by antivirus products, as done for some malware families and antivirus apps in this report. There are several scenarios antivirus apps currently do not handle very well due to the limitations imposed by Android's security model. Without alterations to the Android platform this will not change significantly in the future. As Test Case 1 (*Altered malware*) showed, however, there is also substantial room for improvement for signature and heuristic detection of current antivirus products. Currently it is trivial for malware authors to slightly alter existing malware, with the effect that it will not be detected by antivirus software until new signatures have been released.

Our findings are in good accordance with tests performed by other researchers who proved that using standard code obfuscation techniques current Android antivirus software can be evaded in almost all cases [17].

Like on desktop and server systems, private and corporate users should not rely on antivirus products for perfect protection of their Android devices. While this is a well-known fact, it is even more important on Android due to the limited capabilities of antivirus apps on this platform. In practice, antivirus software on Android can only offer more limited protection than on desktop and server platforms.
In corporate environments, and especially in BYOD scenarios, decision makers and administrators ought to put tight device usage policies and technical measures to reduce the risks of malware on mobile devices into place. On-device antivirus software is not sufficient to ensure the integrity of phones, employee workstations, and the corporate network.

Bibliography

[1] F-Secure Labs, "Mobile Threat Report Q3 2012," tech. rep., November 2012. *http://www.f-secure.com/static/doc/labs_global/Research/Mobile%20Threat%20Report%20Q3%202012.pdf*.

[2] Juniper Networks, Inc., "2011 Mobile Threats Report," tech. rep., February 2012. *https://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2011-mobile-threats-report.pdf*.

[3] McAfee Labs, "McAfee Threats Report: Third Quarter 2012," tech. rep., November 2012. *http://www.mcafee.com/uk/resources/reports/rp-quarterly-threat-q3-2012.pdf*.

[4] TrendMicro TrendLabs, "2012 Mobile Threat and Security Roundup," tech. rep., January 2013. *http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-repeating-history.pdf*.

[5] AV-Test, "Test Report: Anti-Malware solutions for Android," tech. rep., March 2012. *http://www.av-test.org/fileadmin/pdf/avtest_2012-02_android_anti-malware_report_english.pdf*.

[6] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *IEEE Symposium on Security and Privacy 2012*, pp. 95 – 109, May 2012.

[7] R. Fedler, "Analysis of Novel Malware Threats to the Android Platform and Their Consequences (Original: Analyse von neuartigen Bedrohungen der Android-Plattform durch Schadsoftware und ihrer Auswirkungen)," bachelor's thesis, Technische Universität München, August 2012.

[8] V. Chebyshev, Kaspersky Labs, "Mobile attacks!," February 2013. *https://www.securelist.com/en/blog/805/Mobile_attacks*.

[9] R. Fedler, C. Banse, C. Krauß, and V. Fusenig, "Android OS Security: Risks and Limitations," tech. rep., Fraunhofer AISEC, May 2012. *http://www.aisec.fraunhofer.de/content/dam/aisec/de/pdf/tech%20reports/AISEC-TR-2012-001-Android-OS-Security.pdf*.

[10] The H online, "ZeuS trojan increasingly targets German mTANs," August 2012. *http://www.h-online.com/security/news/item/ZeuS-trojan-increasingly-targets-German-mTANs-1663481.html*.

[11] The H, "The alleged flood of Android trojans," August 2012. *http://www.h-online.com/open/news/item/The-alleged-flood-of-Android-trojans-1668760.html*.

[12] Android Developer's Guide, "<permission>." *https://developer.android.com/guide/topics/manifest/permission-element.html*.

[13] X. Jiang, "New RootSmart Android Malware Utilizes the GingerBreak Root Exploit," February 3, 2012. *http://www.csc.ncsu.edu/faculty/jiang/RootSmart/*.

[14] H. Lockheimer, Google Mobile Blog, "Android and Security," February 2012. *http://googlemobile.blogspot.com/2012/02/android-and-security.html*.

[15] Y. Zhou and X. Jiang, "Android Malware Genome Project." *http://www.malgenomeproject.org/*.

[16] "Contagio mobile mini-dump." *http://contagiominidump.blogspot.com*.

[17] V. Rastogi, Y. Chen, and X. Jiang, "DroidChameleon: Evaluating Android Anti-malware against Transformation Attacks," in *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2013)*, May 2013.

[18] Kaspersky Labs, ""Red October" Diplomatic Cyber Attacks Investigation," January 2013. *https://www.securelist.com/en/analysis/204792262/Red_October_Diplomatic_Cyber_Attacks_Investigation*.

[19] Laboratory of Cryptography and System Security (CrySyS Lab), Department of Telecommunications, Budapest University of Technology and Economics, "sKyWIper (a.k.a. Flame a.k.a. Flamer): A complex malware for targeted attacks (v1.05)," May 2012. *http://www.crysys.hu/skywiper/skywiper.pdf*.

[20] X. Jiang, "An Evaluation of the Application ("App") Verification Service in Android 4.2," December 2012. *http://www.cs.ncsu.edu/faculty/jiang/appverify/*.

[21] J. Oberheide and C. Miller, "Dissecting the Android Bouncer," in *SummerCon '12*, June 2012. *http://jon.oberheide.org/files/summercon12-bouncer.pdf*.

[22] SELinux Project, "SEAndroid." *http://selinuxproject.org/page/SEAndroid*.

[23] TOMOYO Linux. *http://tomoyo.sourceforge.jp/*.

[24] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proceedings of the 19th Network and Distributed System Security Symposium*, February 2012.

About SAS

Fraunhofer AISEC is one leading expert for applied IT security and develops solutions for immediate use, tailored to the customer's needs. Over 80 highly qualified employees work on innovative security solutions and analysis methods at all layers, including hardware and embedded systems, networks, and service & application security.
The department Service & Application Security (SAS) specializes in the design, development, and analysis of secure web- and cloud-based applications, and the security of mobile systems. Based on its knowledge gained in mobile security research, SAS offers its clients profound security analysis of their applications and architectures, as well as customized development of secure mobile solutions.
Clients of Fraunhofer AISEC operate in a variety of industrial sectors, such as the chip card industry, the automotive industry, mechanical engineering, software and healthcare industries, as well as the public sector.

About the Authors

Rafael Fedler is an expert in Android security and offensive security techniques. His current research is on customized malware and extensions of the Android platform for improved detection capabilities.

Julian Schütte is a senior researcher focusing on the security of distributed systems and mobile platforms. He is currently working on the application of advanced software analysis techniques for automated inspection of mobile applications.

Marcel Kulicke is an expert in iOS and web application security. He is conducting in-depth security analysis of such systems and develops techniques to support and partly automate black- and graybox tests.