Ana Patrícia Nunes Fernandes

Bachelor in Computer Science and Informatics Engineering

App Threat Analysis: Combining static analysis with users' feedback to accelerate app store response to mobile threats

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: António Ravara, Associate Professor,
NOVA University of Lisbon
Co-adviser: João Casal, Head of R&D, Aptoide

Examination Committee

Chairperson: Dr.João Leite, NOVA University of Lisbon
Raporteur: Dr.Ibéria Medeiros, University of Lisbon
Member: Dr.António Ravara, NOVA University of Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

September, 2018

App Threat Analysis: Combining static analysis with users' feedback to accelerate app store response to mobile threats

Copyright © Ana Patrícia Nunes Fernandes, Faculty of Sciences and Technology, NOVA University Lisbon.
The Faculty of Sciences and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

"It is impossible to live without failing at something, unless you live so cautiously that you might as well not have lived at all - in which case, you fail by default. - J.K. Rowling"

I can not believe that I am actually here, writing the final words for this thesis. I am not going to lie by saying that all this process was a mixture of wonder and fear. I would probably not repeat it so soon, but now I can only think that I am at the finish line - "finally". However, I did not arrive here alone. I had help from lots of people who I now express my gratitude.

First, I have to thank my mother, for all the hard work, for all the cosy meals, for never giving up of me. Thank you mom for giving me your everything, even when there was nothing to give. To my sister for always listening to my complaints and for always know how to distract me when I needed. You are forever my Mega Sis. To my father for the patience with my late night projects, and for always being available to cross cities in the latest hours.

Even though my friends did not help me writing this thesis, I have to acknowledge some of them for making this 5 years more than bearable. To my friend Ana for all your fight spirit, for teaching me to fight in lost battles and get out winning. To my friend David, for the rescue in my first year when I nothing knew about coding (after all I was not a lost cause). To my friend Raquel for showing me that we can escape our little world by doing something so simple as singing together in a crowd of despairing people. To my friend Beatriz for showing me that unicorns are real. To my friends Joana and Mariana for showing me some good times and for being a fantastic duo. All of you made me a little more happier even in the gloomiest days.

Last but not least, I would like to thank and express sincere appreciation to my advisor António Ravara and co-adviser João Casal for the constant assistance and support during the development of this dissertation. This thesis was partially supported by NOVA LINCS grant UID/CEC/04516/2013 and by Aptoide app store. I would also like to thank for the support provided by the company, particularly the R&D and QA team.

Abstract

Today's smart-phones are ubiquitous in people's lives, collecting and storing private and confidential data. At the same time, users are exposed to mobile apps with bad engineering practices and to malicious apps, both endangering the security of their data. This happens because app stores face considerable challenges, like the efficient analysis of the huge volume of apps received, the moving target nature of the threats and the lack of accuracy of users' feedback.

In this dissertation we present a study on the use of automated verification tools of applications at the app market level for improving the security of the end users. This study led to a platform that combines static analysis tools for Android apps with users' feedback to determine the apps threat level. We implemented this platform as a module and evaluated it in Aptoide - an Android app store - to support the quality assurance decisions of app inspection, which might lead to the removal of the app from the store.

The assessment shows that for the 19% of the APKs ranked with the highest threat level, the proposed module only failed in 2%. This means that, in a context of an app store that receives thousands of apps per day, the module is able to inform with considerable certainty which apps need to be inspected by the quality assurance team with urgency, because are likely a threat to consumers. Therefore, the proposed solution contributes to accelerate the app store response to mobile threats and, consequently, to the reduction of its impact on app consumers.

Although the module improves and strengthens the application verification process by uncovering problems that were not previously exposed, after we made more tests we realised that the specification of these problems could be further adjusted.

Keywords: android apps, app store services, mobile quality assurance, software testing, static analysis

Resumo

Hoje em dia temos à nossa disposição telefones sofisticados, *tablets* e *wearables*, que nos permitem fazer quase tudo. Com uma peça de hardware tão pequena, temos o mundo à nossa disposição dentro do alcance de uma app. No entanto a utilização destas aplicações que coletam e armazenam dados privados e confidenciais, representam uma ameaça à segurança do utilizador quando são compostas por más práticas de engenharia ou de más intenções. Isto acontece devido aos grandes desafios que as *app stores* enfrentam num mundo cada vez mais digital, como a análise eficiente do grande volume de aplicações recebidas diariamente, a constante mudança da origem das ameaças móveis e devido à falta de precisão de feedback dada pelos utilizadores.

Nesta dissertação apresentamos um estudo sobre a utilização de ferramentas de verificação automatizada de aplicações ao nível das *app stores* de modo a melhorar a segurança dos seus utilizadores. Este estudo levou nos ao desenvolvimento de uma plataforma que combina ferramentas de análise estática para aplicações Android com o feedback dos utilizadores, de modo a determinar o nível de ameaça de uma aplicação. Implementamos esta plataforma como um módulo que suporta as decisões da equipa que controla a qualidade das aplicações e a sua avaliação foi feita no contexto da Aptoide - uma app store de aplicações Android.

A avaliação mostrou-nos que, para os 19% dos APKs classificados com o maior nível de ameaça, o módulo proposto falha apenas em 2%. Isso significa que, num contexto de uma app store que recebe milhares de aplicações por dia, este módulo é capaz de informar com considerável certeza que aplicações necessitam de urgentemente passar por uma inspeção manual da equipa que controla a qualidade. Pois, provavelmente, são uma ameaça para os consumidores. Portanto, a solução proposta contribui para acelerar a resposta da *app store* às ameaças móveis e, consequentemente, para reduzir o impacto negativo sobre os consumidores.

Embora este módulo melhore e fortaleça o processo de verificação de aplicações, descobrindo problemas que não eram anteriormente expostos, depois de termos efetuado mais testes, percebemos que a especificação destes problemas poderia ser ajustada.

Keywords: aplicações Android, serviços de app store, controlo da qualidade de aplicações móveis, testes de software, análise estática

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface.

APK Android Package.

app Application.

MOBSF Mobile Security Framework.

QA Quality Assurance.

QARK Quick Android Review Kit.

Chapter 1

Introduction

This chapter contains an introduction to the accomplished work in this dissertation. It begins by presenting a description of its context and problem, and what its goals are. The main contributions are also presented below.

1.1 Context

Android has become the most used mobile operating system in the world. In March 2017, the world's largest app store, Google's Play Store, had 2.8 million applications, Apple App Store 2.2 million and the Amazon app store (Android) had 600,000 applications, the same number of Aptoide [Staa]. In 2016 alone there were over 149 billion downloaded applications, worldwide [Stab]. However the noticed exponential growth of mobile applications downloaded has been followed by an increase of security threats of the mobile devices. These threats are made to sensitive data, such as the location, contacts and financial information of the user, stored by these applications. The Nokia threat intelligence report has shown that the overall monthly smart-phone infection rate averaged 0.90% in the second half of 2016 - up 83% from the first half of the year and also highlights the most prolific threats facing mobile devices. Where mobile applications are concerned, this variety of applications can be harmful to the integrity and security of devices and data by leaking sensitive information. These applications are usually free, and have the ability to carry out their proposed functions, but also extract important information from devices. This sensitive information is then sent to a remote server where the cybercriminals can exploit it [Com]. With millions of applications in an app store environment, and more being created every day, the developers goal is to make the next most popular app. They are not focused on security [Inf]. Thus, many of the problems found in an application stem not only from the illicit tries to capture information but also from the

lack of good security practices in application development, which facilitates the success of threats. Software weaknesses have become one of the top problems in recent years, for there are many ways software developers can make mistakes that can lead to insecurity. Every one of these weaknesses is subtle and many are tricky. Software developers are not taught about these weaknesses in school and most receive no training on the job about these problems [OWAa].

This is a global issue, spanning most countries that have a significant smart-phone user base. Different countries have different threat profiles, depending on the nature of mobile phone usage in the country [McA]. The fact that these threats have been changing makes them continue to succeed in their illicit purposes, due to the lack of automatic intelligent mechanisms capable of detecting and mitigating them efficiently [Nar+16]. Given the moving target nature of the threats on the mobile security arena and the considerable challenge that outcomes from the high volume of applications received on the app stores, the users are key actors on the detection and response to attacks not blocked by app stores. The fast analysis of their feedback is key to a swift removal of the threats from the repositories and, thus, reduce the impact on consumers. Even if the users' feedback is not frequently accurate, during this paper we have taken it into consideration. We find it to be a key on the detection and response to attacks not blocked by app stores, given that their feedback could be a way of work around the moving target nature of the threats stemming from the poor engineering choices.

1.2 Problem

The recent survey made by the Stackoverflow website about the developers profile (favourite technologies, coding habits, and work preferences, as well as how they learn) shows that about 65% of mobile developers work with Android [Staa]. This platform has gained over 85% of the smart-phone market, [Stab] and just its official app store contains more than 2.8 million applications. As a result, a security mistake in an in-house application may jeopardise the security and privacy [CVE] of billions of users. The security of smart-phones has been studied from various perspectives such as the device manufacturer [Wu+13], its platform [Xu+16], and end users [JC15]. Manifold security APIs, protocols, guidelines, and tools are proposed. Nevertheless, security concerns, in effect, are outweighed by other concerns [BC14]. Many developers undermine their significant role in providing security [Xie+11]. As a result, applications still suffer from serious proliferating security issues. For instance, the analysis of 100 popular applications downloaded at least 10M times, revealed that over 90% of them, due to development mistakes, are prone to SSL vulnerabilities that allow criminals to access credit card numbers, chat messages, contact list, files, and credentials [OC15].

After an empirical study made by a research team in Switzerland [Gha+17], where they have statically analysed 46 000 applications from the AndroZoo and Google Play data set they have concluded that despite the diversity of applications in popularity, size,

and release date, the majority suffer from at least three different security vulnerabilities (XSS-like code injection, dynamic code loading and the use of custom scheme channels and expose a unique hardware identifier).

Despite the best efforts of app stores, the existing markets provide little meaningful security or privacy guarantees because market providers have neither the tools nor the resources to perform a detailed analysis of submitted applications [ME10]. Then malicious applications can enter their ecosystem e.g., app installation from untrusted sources, inability to detect malicious behaviour in applications, access of malicious websites. Hence, there is a need to develop automatic tools and techniques that can detect vulnerabilities in the submitted applications [RM18].

1.3 Goals

1.3.1 General

An automated validation of applications at the app market level is a promising approach for improving the security of the end users. In this thesis, we foresee a validation process applied by app stores to applications submitted by the developers and we present a system that aims to achieve this goal.

At a high level, the system goal is of devising an automatic tool for targeting the inspection of applications with programming malpractices (due to poor implementation / engineering choices).

1.3.2 Specific

For that purpose we propose the construction of a tool supported by automated analysis tools that would analyse applications and based on that analysis would estimate the threat that an application represents.

To estimate the threat level of an application, we propose the elaboration of metrics based on the quantity of the problems found as well as the amount of users' feedback that an application has. During this dissertation we gave value to the users' feedback, so we could analyse its impact on the detection and response to attacks not blocked by app stores. Giving us a way of work around the moving target nature of the threats stemming from the poor engineering choices.

By elaborating metrics related to quantity parameters, it is possible to promote adjustments and changes in the development process, in order to reduce or eliminate causes of problems that significantly affect the threat level estimation system.

The proposed metrics will undergo an experimental evaluation and a validation applied to a set of real case studies.

1.4 Contributions

The main contribution of this dissertation is the definition, implementation and evaluation of a module capable of targeting the inspection of applications with programming malpractices, in order to provide to app stores an automatic tool able to give a threat level system of the submitted applications after a detailed analysis.

This threat level system would inform an app store of which of the submitted applications pose the biggest threat to their ecosystem. Giving to an app store a tool to improve security or privacy guarantees.

Another contribution is the definition and evaluation of metrics for the threat level system, in order to support the quantitative evaluation of the submitted applications. Based on the related work there is still no well-accepted solution to the problem; quoting from the paper Static Analysis of Android Apps: A Systematic Literature Review [Li+17] "There is no single work that has been proposed to tackle all challenges of static analysis that are related to Android programming". Therefore, the metrics are proposed according to the data we obtained from the analysis of the module.

We also performed an experimental evaluation where the metrics are applied to a set of real case studies.

Chapter 2

Related Work

This chapter contains important concepts pertaining to software verification and validation techniques, security vulnerabilities and static analysis tools. Its main objective is to introduce some important concepts that are related to this dissertation, making a framework of it.

2.1 Software Verification and Validation Techniques

The verification and validation of software is typically done through three techniques: manual code review[Gee16], automatic analysis comprehended between dynamic[Bal99] or static[Wic+95] approaches and semi-automatic with theorem provers [Duf91].

A manual code review depends entirely on the reviewer and it is conducted to find bugs and improve overall quality of the software [Sma], while an automated code review checks source code for compliance with a predefined set of properties, rules or best practices [Cas]. In conclusion automated code review solutions have advantages over a manual audit, for they are able to complete the review at a much faster pace, there is not the possibility of human error, they improve code coverage, and are more effective and ultimately more affordable. However, an automatic review does not eliminate the need for a human reviewer, as they produce both false positive and false negative results [Sma].

The techniques used in automated code analysis can be either static or dynamic. Static analysis is done by inspecting a program (on source, binary or byte code level) without actually executing the program. Dynamic analysis runs a program to observe its actions in an effort to uncover more subtle defects or vulnerabilities.

Both static and dynamic analysis have their advantages and limitations. A dynamic

analysis reveals subtle defects or vulnerabilities whose cause is too complex to be discovered by static analysis. It plays a role in security assurance, but it will only find defects in the part of the code that is actually executed. In the other hand, static analysis examines all possible execution paths and variable values, not just those invoked during execution [Int].

After we weigh these considerations with the complexities of our own project, we have chosen static analysis as the selection criteria of the tools to be integrated in our system.

2.2 Static Analysis

Static analysis is a method of computer program debugging that is done by examining the code without executing the program [Tecb]. In most cases the analysis is performed on some version of the source code, and in the other cases, some form of the object code. There are also some static analysis tools that make their analysis at the byte-code level. There are multiple advantages to static analyse code such as revealing errors, or identifying code fragments that can lead to errors in the future [PVS]. Besides many automatic static analysis tools can detect problems at security, reliability and functionality levels. They are useful for security threats such as buffer overflows, SQL Injection Flaws, and so forth [Tecb]. However, there are weaknesses too. There are many types of security vulnerabilities that are very difficult to find, such as authentication problems, access control issues, insecure use of cryptography, etc. The current state-of-the-art only allows such tools to automatically find a relatively small number of application security flaws. Configuration issues are also something hard to find since they are not represented in the code. The same happens to proving that an identified security issue is an actual vulnerability. Many of these tools have difficulty analysing code that cannot be compiled. Analysts frequently cannot compile code because they do not have the right libraries, all the compilation instructions, all the code, etc [OWAb]. There is also the problem of false positives. Correct tools/approaches never give false negatives, i.e., never say a "bad" program (one that has violations of the property) has no problems. Complete tools/approaches are unattainable for non-trivial properties, since by the Rice's Theorem [Wika], they are undecidable for Turing-complete computational mechanisms [Wikb]. Furthermore, Godel's incompleteness theorems [Sci] ensure that some properties are improvable and moreover, there are no "interesting" reasoning mechanisms both consistent and complete. Therefore, static analysis cannot be precise and to be correct must over-approximate and give false positives.

2.3 Security Vulnerabilities

There is a vast amount of published work on the topic of Android security analysis. We began our research by learning and analysing the different types of vulnerabilities to be expected in Android applications. According to a present study on security smells in

Android [Gha+17], in resume, there are 28 code smells (symptoms in the code that signal the prospect of a security vulnerability) that may lead to vulnerabilities in applications and show their prevalence in real-world Android applications. From those 28 identified code smells, we have selected a 14 according to the conclusions observed in the study.

From the category Insufficient Attack Protection, we selected 3 problems: untrustworthy libraries, outdated libraries and unnecessary permissions. From the category Security Validation, we selected one problem: weak cryptographic algorithm. From the category Broken Access Control, we selected 4 problems: unauthorised intent, unconstrained inter-component communication, exposed adb-level capabilities and debuggable release. From the category sensitive data exposure, we selected 3 problems: exposed persistent data, insecure network protocol and exposed credentials. From the category lax input validation, we have chosen 3 problems: XSS-like code injection, dynamic code loading and SQL injections.

The categorisation and explanation of the selected problems are described below and are summarised in Table 2.1.

Category	**Security Vulnerability**
Insufficient Attack Protection	*Untrustworthy Libraries*, *Outdated Library*, *Unnecessary Permissions*
Security Invalidation	*Weak Crypto Algorithm*
Broken Access Control	*Unauthorised Intent Receipt*, *Unconstrained Inter-Component Communication*, *Exposed adb-level Capabilities*, *Debuggable Release*
Sensitive Data Exposure	*Sensitive Data Exposure*, *Insecure Network Protocol*, *Exposed Credentials*
Lax Input Validation	*XSS-like Code Injection*, *Dynamic Code Loading*, *SQL Injection*

Table 2.1: Security vulnerabilities summary.

Insufficient Attack Protection

- **Untrustworthy Libraries**
  To cope with the complexity of modern software systems and as way to speed up the process of development of an application, the developers usually rely on the functionalities provided by off-the-shelf libraries. This is a problem since these libraries are often maintained by different developers and, as the result, their update cycles generally do not coincide when there are bug fixes and improvements in newer releases [Gha+17]. Consequently, a security breach in an old library or a deprecated API could lead to serious issues, i.e., introduce vulnerabilities and compromise user data [Wat+17].

- **Outdated Library**
  Libraries usually offer various bug fixes and improvements in newer releases, but often different developers maintain libraries and applications, and their update cycles generally do not coincide. Consequently, a security breach in an old library or a deprecated API could lead to serious issues [Gha+17]

- **Unnecessary Permissions**
  The use of protected features on Android devices requires explicit permissions, and developers occasionally ask for more permissions than necessary [TM16]. The more permission-protected features an app can access, the more sensitive data it can reach. Consequently, a more permission-hungry app may expose users to additional security risks [TM17].

Security Invalidation

- **Weak Crypto Algorithm**
  The fundamental set of cryptograph algorithms can be categorised into symmetric, asymmetric, and hash functions. The use of weak cryptographic hash functions like SHA1 or MD5, insecure modes e.g., ECB for block ciphers, could subject to security issues [Gha+17].

Broken Access Control

- **Unauthorised Intent Receipt**

  An intent is an abstract specification of an operation that applications can use to utilise the actions provided by other applications. An explicit intent guarantees communication with the specified recipient, but it is the Android system that determines the recipient(s) of an implicit intent among available applications. The existence of an intent with private data, but without a particular component name (the fully-qualified class name) [Gha+17], can be a target for a threat called intent

hijacking could arise in which user information carried by the intent could be manipulated or leaked [Chi+11].

- **Unconstrained Inter-Component Communication**

  Since Android applications can reuse components (e.g., activities, services, content provider, and broadcast receivers) from other applications trough the intent-filter element or android:exported = true attribute in the manifest file without any permission check to ensure that a client app is originally permitted to receive that service, a threat called component hijacking can arise when a malicious app escalates its privilege for originally prohibited operations through other applications that access those operations [Wu+16], [Dav+11].

- **Exposed adb-level Capabilities**
  The Android Debug Bridge (adb) is a versatile tool that provides communication with a connected Android device. Many developers opt for adb-level capabilities to legitimately access a subset of signature-level resources [Lin+14]. If an app communicates locally with an adb-level proxy through the TCP sockets opened on the same device, which exposes the adb server to any app with the INTERNET permission, a malign app with ordinary permissions can command the adb and establish serious attacks [Hwa+15].

- **Debuggable Release**
  During app development there exist two major build configurations, debug and release. The first is meant for active development, while the latter is for signed in-market releases. However, developers may forget to switch to release mode before publishing an app [Xu+13]. Hence the manifest file will contain the attribute android:debuggable = true. Applications shipped with debugging enabled always try to connect to a local Unix socket opened by the Android Debug Bridge (adb). While adb is not running on every consumer device, a malign app could disguise itself as an adb service and connect to random debuggable applications. Consequently, a malicious app is able to gain full access to the Java process and can execute arbitrary code in the context of the debuggable app [MWR].

Sensitive Data Exposure

- **Exposed Persistent Data**
  Android provides various storage options to store persistent data. These options vary depending on the size, type, and accessibility of data [Dav+11]. If a developer chooses a particular option without considering its security implication, , private

data can be exposed. For example if in the application there is a private storage with global access scope (i.e., MODE_WORLD_READABLE or MODE_WORLD _WRITEABLE), the app relies on ContentProvider to access data, but there is no access restriction for other applications.

- **Insecure Network Protocol**
  There are many channels devoted to data transportation, however insecure ones like HTTP are more prevalent and easy to maintain. These insecure channels transfer un-encrypted data, giving a chance to relay the data and possibly alter it by the attackers.

- **Exposed Credentials**
  Passwords, private keys, secret keys, certificates, and other similar credentials are commonly used for authentication, communication, or data encryption. If the app contains hard-coded credentials, or they are stored without any password protection such as when the KeyStore.ProtectionParameter is null, in some circumstances such data is inadvertently disclosed to unauthorised parties. Which could break the intended security [Gha+17].

Lax Input Validation

- **XSS-like Code Injection**
  WebView is an essential component that enables developers to use web technologies such as HTML and JavaScript to deliver web content within an app. Unlike Web browsers like Chrome, FireFox, etc. which are developed by well-recognised companies that we trust, each app using a WebView is like a customised browser which may not have undergone thorough security tests. If the setJavaScriptEnabled call with value true which enables execution of JavaScript exists in the code, and the app fetches web content from untrustworthy sources (e.g., by calling loadUrl or loadData on WebView) without applying proper sanity checks [Gha+17], this means that the app may load web content unsafely i.e., without sanitising the input from any code. Consequently, an adversary could inject malicious code through any channel that the app uses to get web content [Jin+14].

- **Dynamic Code Loading**
  Android allows applications to load and execute external code and resources, trough the use of any class loader. Although dynamic code loading is widely adopted, developers are often unaware of the risks associated to this generally unsafe mechanism or fail to implement it securely [SPV14]. An attacker can replace the code that is to be loaded with a malicious one. Consequently, this can lead to severe vulnerabilities

such as remote code injection [Fal+15].

- **SQL Injection**
  If a developer does not make a proper validation of the input that is passed to the database and if the application was developed to directly use the input to build a query that will be run by the database engine; an attacker who succeeds at inserting malicious code into SQL statements, can access or modify the data from the database.

2.4 Static Analysis Tools

In this section we present our search process on static analysis tools and the decisions made during that process. To select tools for the module, we have restricted the search to the following factors:

- **Free**
  We searched for solutions that were freely available, ignoring the payed ones.
- **Maintained**
  During the selection of the tools we gave preference to the solutions that were actively maintained. Since in most cases the unmaintained/abandoned tools do not support the currently supported versions of Android.
- **Security Vulnerabilities Detection**
  From the multiple tools we have encountered, we have selected the ones capable of flag signs of malicious behaviour in an application according to the security vulnerabilities we have chosen to target (see section 2.3). Despite the many tools available for vulnerability detection (typically used by app developers), we have chosen to not pursue them since our target are APKs[1]. For using such tools it would be necessary to have the source code of the application and for that purpose we would have to get the source code from each APK[1]. Although we tried to follow that path through the use of reverse engineering tools, it showed us that due to the obfuscation of parts of the code that some applications have, combined with the fact that some tools are restrained to a limited set of android versions, this was not a viable solution.
- **Minimal Adaptation**
  We were also interested in solutions that could detect malicious behaviours with none or minimal adaptation, e.g., build an extension to detect a specific vulnerability. The ground behind this decision was to eliminate human bias and errors

[1] APK explained in chapter 3

involved in identifying, creating, and using the proper modifications. This way we could maintain a typical workflow: build the tool, follow the documentation, and apply it to the APKs.

To select tools for this evaluation, we collected information from repositories and blog posts approaching Android security solutions. We then considered 8 tools: AndroBugs [Gita], AndroWarn [Deb], Argus-SAF [Gitb], DroidRA [Li+16], Evicheck [Evi], MobSF [Gitc], Qark [Gitd] and SUPER [Gite]; and evaluated them by reading their documentation and the available resources.

Since we focused on selecting only maintained tools, we rejected AndroWarn as it was not updated after 2013 [Deb]. We also rejected a tool if we had error messages that has no clear fixes by exploring the available bug report. This resulted in rejecting DroidRA [Li+16].

Of the remaining tools, we tested 6 local tools using 10 APKs (Android Package) from Google Play store. The APKs tested were divided into 5 categories, Banking, In-app purchases, Games, Privacy and Tools. For the banking category we have chosen MB Way [Plag] and Wallet [Plaj]; for the in-app purchases category we have chosen Drink Water [Plac] and Freeletics Bodyweight [Plae]; for the games category we have chose Clash Royale [Plab] and Millionaire 2018 [Plah]; for the privacy category we have chosen LOCKit [Plaf] and My Passwords - Password Manager [Plai]; finally for the tools category we have chosen Always On AMOLED [Plaa] and Flashlight [Plad]. We executed each tool with each of the above applications as input on an Ubuntu Xenial Xerus with the following features: Processor: Intel Core i5-6200U CPU @ 2.30GHz × 4 and Memory: 7,2 GiB. If a tool failed to execute successfully on all of these applications, then we rejected the tool. We rejected Argus-SAF because it ran out of time or memory while processing the test apps [Gitb]. The results were grouped by the remaining 5 tools in appendixes: A, B, C, D and E.

In the following we will describe and present each tool and point out some of their main features.

AndroBugs is an android vulnerability scanner. It is a Python framework that targets application developers only. It is based on AndroGuard, which it uses to decompile APKs. Based on a present list of vulnerability vectors, it scans code hotspots to find any potentially vulnerable code, function calls or fields, and then goes to the source function of those issues and reanalysis the code to confirm its findings [Pet]. Once it discovers security issues, AndroBugs will dump data inside the user's terminal, data that holds information on the vulnerability vector's type, code path inside the APK's source, severity level, detailed explanations, mitigation recommendations, and even some reference research papers, if available [Cim].

Evicheck is a tool for the verification, certification and generation of lightweight fine-grained policies for Android. It uses a lightweight static analysis to show the conformance between policies and application behaviour. These policies are a set of rules defined by the

user and can be about the use of APIs, permissions allowed or anti-malware patterns. In the verification mode, it takes an application together with a policy as input, and answers whether the policy is satisfied by the application and eventually outputs a certificate (digital evidence). EviCheck also returns diagnosis pointing to the first violated rule of policy [Seg].

Mobile Security Framework is an open source mobile application (Android/iOS) automated pen-testing framework capable of performing static and dynamic analysis. The static analyser can perform automated code review, detect insecure permissions and configurations, and detect insecure code like SSL overriding or bypass, weak cryptography, obfuscated codes, improper permissions, hard-coded secrets, improper usage of dangerous APIs, leakage of sensitive/PII information, and insecure file storage [Cyb].

Qark is a static code analysis tool, designed to recognise potential security vulnerabilities and points of concern for Java-based Android applications. It stands for Quick Android Review Kit and is a tool capable of inform developers about potential risks related to Android application security, providing clear descriptions of issues and links to authoritative reference sources [Tru].

Super Android Analyser is a command-line application that analyses APKs in search for vulnerabilities. It does this by decompressing them and applying a series of rules to detect those vulnerabilities. It is written in Rust and detects SQL injections, XSS attacks, superuser checking applications, URL disclosure, weak algorithms, dangerous permissions, sending sms-mms, get device ID, cell location, get SIM serial and bad practices [Egu].

Although, there are tools that find the same issues we have chosen to use them all together. This decision was made because we observed that even if the same issues are reported, these tools do not operate the same way. For several apps we got, for some of the tools, different outputs for the analysis of the same issue (weak cryptography use, for example, is reported differently between Androbugs and SUPER). We concluded that the tools complement each other and decided to use them together. Meaning that we can have two tools that analyse, for example weak cryptography use, but have different outputs. So these different results complement each other.

Table 2.2 shows the relation between the selected tools and the security vulnerabilities we have chosen to target.

Security Vulnerability	AndroBugs	EviCheck	MobSF	QARK	SUPER
Untrustworthy Libraries	Yes	Yes	No	No	No
Outdated Library	Yes	No	Yes	Yes	No
Unnecessary Permissions	Yes	Yes	Yes	Yes	Yes
Weak Crypto Algorithm	Yes	No	Yes	Yes	Yes
Unauthorised Intent Receipt	Yes	No	No	Yes	Yes
Unconstrained Inter-Component Communication	Yes	No	No	Yes	Yes
Exposed adb-level Capabilities	Yes	No	No	Yes	Yes
Debuggable Release	Yes	No	No	Yes	No
Sensitive Data Exposure	Yes	No	Yes	Yes	Yes
Insecure Network Protocol	Yes	No	No	Yes	Yes
Exposed Credentials	Yes	No	Yes	Yes	Yes
XSS-like Code Injection	Yes	No	No	No	Yes
Dynamic Code Loading	Yes	No	No	Yes	No
SQL Injection	Yes	No	Yes	No	Yes

Table 2.2: Features of the static analysis tools

Chapter 3

Framing of the Work

This chapter introduces some important concepts related to Android applications and obfuscation. We also make an introduction on the Aptoide app store and its app validation system. Since this app store is the context of our dissertation, we will also provide a description of its problem and goals.

3.1 Android Basics

Android is an open source platform based on Linux, consisting essentially of operating system and middleware, a software structure that manages the functionality of the device and integrates a set of APIs (Application Programming Interface) that allow the development of applications and the interface with the user, developed primarily for mobile platforms such as smart phones and tablets [Col+11; Mei08].

All Android applications run through the Dalvik VM (Virtual Machine). Although it is a virtual machine that allows to execute applications developed in Java, this one is not truly a virtual machine of Java and has some differences. The Dalvik VM runs only Java files after they have been compiled in its own format (.dex - Dalvik executable) and has been specially developed so that executions of multiple instances of this virtual machine are possible [KM12].

Android applications have multiple entry points. They can rather use parts of other Android applications on-demand and can require their services by calling their event handlers, directly or through the operating system. In particular, Android applications contain activities (code interacting with the user through a visual interface), services (background operations with no interaction with the user), content providers (data containers such as databases) and broadcast receivers (objects reacting to broadcast messages). These Android application are installed in the form of application package files and

commonly known as APK (Android Package) files, which is, basically a zip compressed archive. APK files are container files that contain both Application code, resources as well as the application manifest file [Sch+16].

3.2 Obfuscation

During the development of an Android application the developer may choose to protect his intellectual property by using obfuscation techniques in his code. For instance, some companies may decide to obfuscate the application code as a post-processing step to the class file generation, so as to make reverse-engineering an application difficult or impossible. This decisions poses as a double-edged sword by the security community. To a legitimate software company, obfuscation keeps its competitors away from copying the code and quickly building their own products in an unfair way. To a malware author, obfuscation raises the bar for automated code analysis and manual investigation [Don+18].

In general, obfuscation attempts to garble a program and makes the source or machine code more difficult for humans to understand. Programmers can deliberately obfuscate code to conceal its purpose or logic, in order to prevent tampering or deter reverse engineering.There are several common obfuscation techniques used by Android applications, including identifier renaming, string encryption, excessive overloading, reflection, and so forth [Don+18].

3.3 Aptoide App Store

Founded in 2011, Aptoide is an Android app store with over 200 million users and 1 million applications that has partnerships with more than 15 thousand developers (numbers of 2017). In 2018, Aptoide is receiving over 15 thousand applications per day (new applications or new versions), which need to be tested in order to assure the quality of service and the security of the users. It is an alternative marketplace for mobile applications which runs on the Android operating system and according to a study on security management of global third-party Android marketplaces [Ish+17], is one of the safest. In Aptoide there is not a unique and centralised store but each user manages their own store. This means that any user can create and manage its own app store, not being necessarily a developer.

When an application is submitted to Aptoide, it needs to pass on two testing processes to be available for consumers: the Aptoide Sentinel and the Fake Detector (see Figure 3.1). Sentinel emerged as a system to automatically detect malicious software, PUAs (Potentially Unwanted Applications) and aggressive adware libraries [Pir]. Each uploaded application goes through a verification process composed by 8 antiviruses. Given that this system is in continuous upgrade and improvement (antivirus updates or more antivirus

added to the process), the apps already on the repository are periodically re-tested to assure the users protection to recently discovered threats.

The Fake Detector is a tool that automatically detect fake apps, like apps that have just ads. It is based on rules or heuristics detected by Aptoide's QA team, whose pattern is modelled to automatically block apps that not being malicious, are not offering genuine functionalities to users. The fake detector is also a process in development, so if there are any new faulty patterns those will be added to the process. This ensures that the process will always be updated and prepared for new known threats.

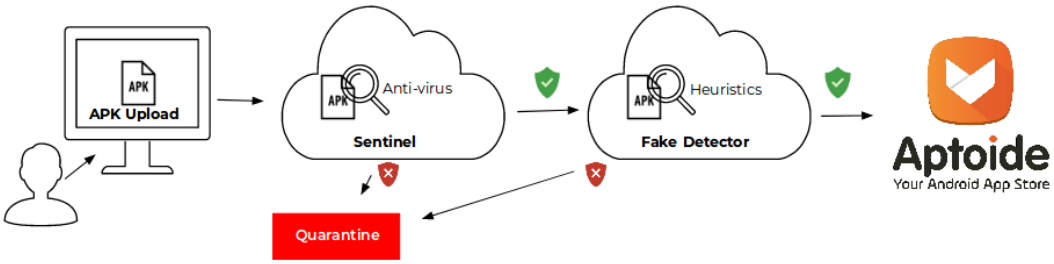


Figure 3.1: Aptoides' validation process [Ico; Teca]

3.4 Problem

Even if Aptoides' validation process protects the app store against known malware, there is so little it can do about uploaded applications containing bad engineering choices. Like applications that are a vulnerability due to development mistakes, e.g. SSL vulnerabilities that allow criminals to access credit card numbers. These applications are then analysed by the quality assurance (QA) team if an application receives bad user feedback. In Aptoide an user is allowed to give feedback to an application by raising a flag. The flag system is constituted by 5 types of flag: good, needs a license, fake, freeze and virus. However, there was in average around 1753 negative user feedbacks per day between 2016 and 2018, and since the QA team workload is ordered by a FIFO policy, where apps with recent feedbacks appear in first place to be reviewed, this huge amount of feedback increases the QA team backlog and does not provide them an effective work methodology.

It is also important to point out that most of the negative user feedback is not accurate (e.g. a user can make a complaint about the result of a game and give an unsatisfactory feedback with no regard for the feedback system implemented), which results in a waste of time when the applications analysed by a QA member are good instead of bad.

Despite the best efforts of Aptoide app store, the existing security and privacy guarantees are still not enough. The lack of an automated approach to check for these security vulnerabilities and the growing QA workload are two of the reasons that makes it necessary to develop an automatic approach that can detect vulnerabilities in the submitted applications even after they passed all the Aptoides' validation system.

3.5 Goal

In order to respond to the problem and the context of Aptoide, previously presented, the proposed solution will be modelled as to serve as a decision support to the QA team. It will automatically analyse an application for security vulnerabilities that would most likely put at risk the security of both the user and marketplace. Combining that analysis with the users feedback the solution will provide a prioritised APK list. Thus the time spent manually reviewing apps with some bad feedback would be better used since the ones with a higher threat level would be reviewed first instead of others that represent a lesser threat. This combination can be rather challenging since the user feedback tends to not be accurate, however the users' feedback can also help us surpass the challenges posed by code obfuscation and to better target the moving threats.

It is important to notice that the applications that will go through this solution were already checked for virus and for common malicious patterns. Hence the solution will not have the power to exclude an uploaded APK based only by the issues it might find, nor will it replace the existing validation processes. Moreover, this solution depends on the validation made by the two main processes, the Sentinel and the Fake Detector (if any of these validation processes finds any threat on an uploaded application, then that application will be rejected and will not even go through the solution we propose).

Chapter 4

Implementation of the Solution

This chapter presents the implementation main ideas that were considered, followed by the description of the modules' architecture and information about the technologies used in the development of the module that allows an automatic validation of applications.

4.1 Main Ideas

To achieve our goal, we propose a module to integrate in the Aptoides' system and check automatically APKs for infractions that could put at risk the security of both the user and marketplace.

This module is supported by our selected static analysis tools and work as a decision support for the QA team. The support is given trough a prioritised collection of APKs, made according to the analysis of the static analysis tools and according to the users feedback. The module will allow the members of the QA team to better manage their time by manually reviewing the applications with more security vulnerabilities.

4.2 Architecture of the Solution

In order to better visualise our goal we have designed and followed the architecture shown in Figure 4.1.

The first thing we would like to point out from the architecture is that our system will not interfere with the already used and operable Aptoides validation system. In fact, our proposed module will depend on the validation made by the two main processes, the Sentinel and the Fake Detector: if they find a virus on an APK or a malicious pattern, the module will not even process that APK. We also have chosen to follow a modular design so that this solution can be independently created and then used in different systems.

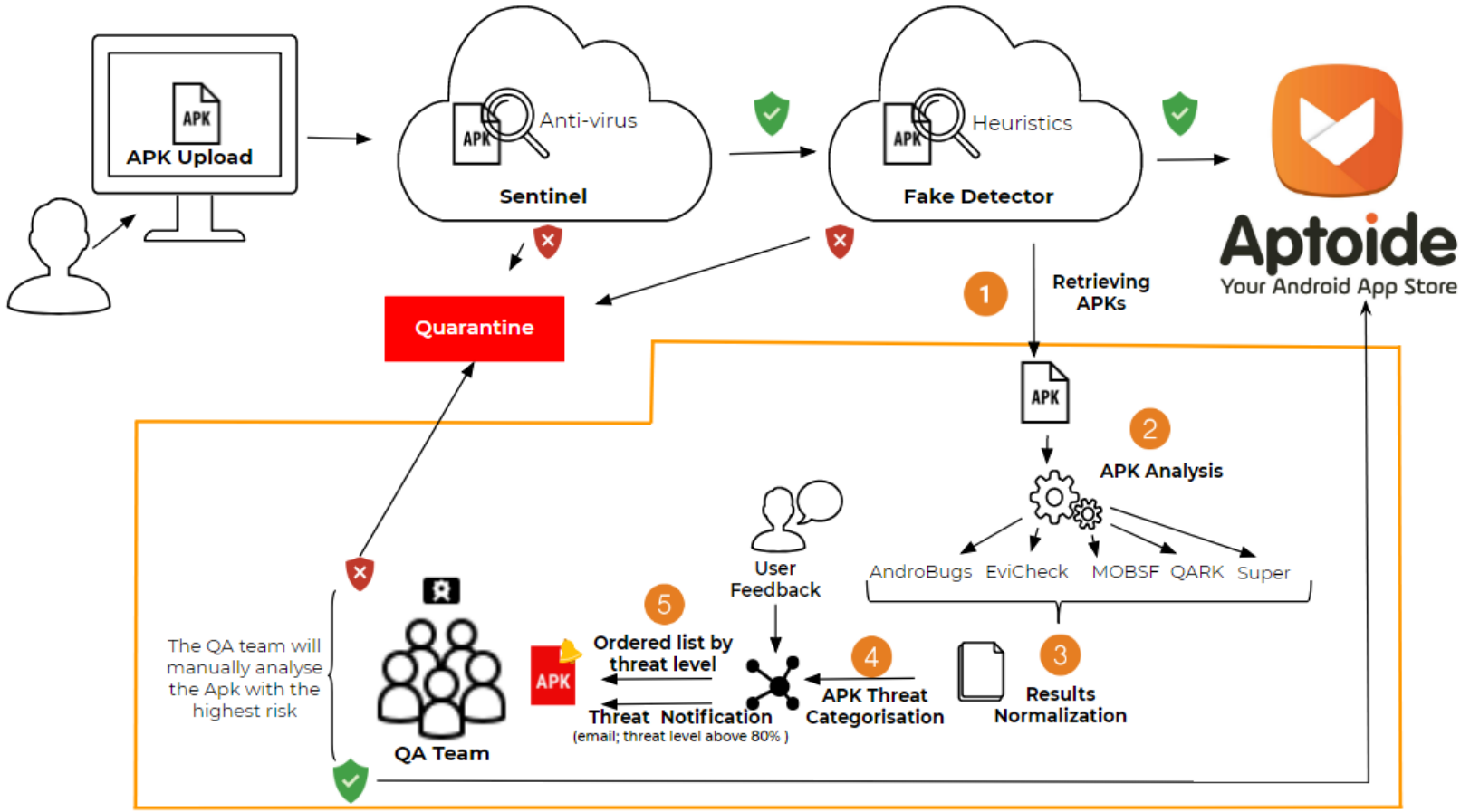


Figure 4.1: Proposed architecture for the module [Ico; Teca]

4.3 App Threat Analysis Module

In order to have a clear understanding of the module and how it works, we will describe the process that an APK goes trough as well as the response of the module when a user uploads an APK that is validated. As Figure 4.1 shows there are 5 stages of operations in the implementation of the module:

- **Retrieving APKs**
  First, the module will extract the uploaded APKs that already went through the validation process, as the architecture shows. These APKs will be fetched from the point where they are available in the app store. For this part, it was necessary to resort to the Aptoide's API.

- **APK Analysis**
  After retrieving the APKs we will submit them to the analysis of each one of the five selected static analyse tools. Despite this process being represented as a parallel execution, we have tested the modules performance against the two methodologies: sequence and parallel. More details about this option are displayed on the results.

- **Results Normalisation**
  Once the analysis is complete, we will have 5 different results that will need to be normalised so we can have some quantifiable results. Each tool has already a system to quantify the severity of each problem found:

  - AndroBugs
    The AndroBugs tool has a severity system of 4 levels (Critical, Warning, Notice and Info).
  - Evicheck
    This tool is the only one with a binary severity system, which means that or there is a problem or there is not.
  - MOBSF
    The MobSF tool has a severity system of 4 levels (High, Warning, Info and Good).
  - Qark
    Qark has a severity system of 4 levels (Vulnerability, Warning, Error and Info).
  - Super
    Super has a severity system of 5 levels (Critical, High, Warning, Medium and Low).

  Since all tools have different severity rating systems, to better comprehend the results we took those systems, and we made an unique correspondent one capable of including all of them. We have transformed those rating systems into one that

could only have values between 0 and 1 (where 0 is not a threat and 1 is a threat). We took the severity of each problem found by these tools and we have associated it to a normalised result. Hence, instead of getting words as the quantifiable variable of the problem we gave it a value:

  - AndroBugs
    The corresponding severity system of 4 levels (Critical=4/4; Warning=3/4; Notice=2/4 and Info=1/4).
  - Evicheck
    The corresponding severity system of 2 levels (True=1; False=0).
  - MOBSF
    The corresponding severity system of 4 levels (High=4/4; Warning=3/4; Info=2/4 and Good=1/4).
  - Qark
    The corresponding severity system of 4 levels (Vulnerability=4/4; Warning=3/4; Error=2/4 and Info=1/4).
  - Super
    The corresponding severity system of 5 levels (Critical=5/5; High=4/5; Warning=3/5; Medium=2/5; and Low=1/5).

- **APK Threat Categorisation**
  The APK threat categorisation combines (a) the normalised results of the static analysis tools with (b) the users' feedback. During this dissertation we have given value to the users' feedback, since it could be a means to the detection and response to attacks not blocked by app stores. Hence, helping our solution block the moving target nature of the threats made on the mobile security arena and to overcome the considerable challenges that outcomes from the high volume of apps received on the app stores.

  Thus the users' feedback is important for this categorisation given the motives previously mentioned. However, since frequently the users' feedback is not accurate, we propose to combine the results of static analysis tools with the users' feedback.

  The hypothesis we suggest is that if an APK has several severe problems reported by the static analysis tools and there is a big quantity of negative user's feedback, then it is likely that the application is harmful to the consumers. Therefore, the quality assurance team should be notified. In this hypothesis we take into consideration not only the quantity of problems observed during the second stage of the process (APK Analysis) but also the severity of the problems found. So we also added more two variables into consideration: the quantity of user feedback that the analysed APK has and the type of user feedback.

As already explained, the Aptoide user feedback system has 5 types of flags: good, needs a license, fake, freeze and virus. To each one of these types of flags we have given a value comprehended in the set -1, 0, 1. So that we could quantify the type of user feedback in our hypothesis. Hence, the flags that represent bad user feedback (freeze, virus, fake) we have given a positive value (1); to the positive user feedback (good) we have given a negative value (-1); and because we could not verify with the selected tools the copyrights of the uploaded application, we had to give a null value (0) to the flag entitled "needs license".

Based on the related work there is still no well-accepted solution to the problem; quoting from the paper Static Analysis of Android Apps: A Systematic Literature Review [Li+17] "There is no single work that has been proposed to tackle all challenges of static analysis that are related to Android programming". Therefore, we proposed the following solution that we will validate using a sample of reviewed APKs (a sample reviewed by the Aptoide's quality assurance team and labelled as "threat" or "not a threat", given that "not a threat" applies to the applications that passed their test and "threat" to the ones that did not).

To obtain the threat level of an APK, we decided to formulate three metrics that would take into consideration all the variables mentioned in our hypotheses. Firstly, we said we would take into consideration: (a) the quantity of problems observed during the APK analysis and (b) the severity of the problems found. We have placed both of these variables together in an equation 4.1:

$$WeightedResult = \frac{AmountOfProblems}{Max(AmountOfProblems)} * W1 + \frac{\sum ProblemsSeverity}{AmountOfProblems} * W2 \tag{4.1}$$

$$W1 + W2 = 1 \tag{4.2}$$

In Equation 4.1 we have taken our first variable, the quantity of problems (a), to calculate the corresponding value of amount of problems directly proportional to the maximum amount of problems found in all APKs analysed. With this we want to see the proportion of problems of one APK when compared with the sample. Then we took our second variable, the severity of the problems found (b), to calculate the average of the severity of the problems found for the analysed APK.

The AmountOfProblems in the equation number 4.1 refers to the total amount of problems found in the tools analysis for the analysed APK, as for the Max(AmountOfProblems) it refers to the maximum total number of problems found during the analysis of a set of APKs, at the moment of the analysis of the APK currently reviewed. To each part of the equation we have assigned a weight (W1 and W2) and then added the two parts. We have assigned these weights in order

to analyse the impact of each variable in the formulated equation and how the variation of those variables can give us a better threat rating system.

After formulating the weighted result, we then devised the second formula taking into account the other variables of our hypothesis. Thus we payed attention to: (c) the quantity of user feedback that the analysed APK has and (d) the quantifiable type of user feedback. We have placed both of these variables together in equation 4.3:

$$UserFeedback = \frac{AmountOfFlags}{Max(AmountOfFlags)} * W3 + \frac{\sum WeightedFlags}{AmountOfFlags} * W4 \tag{4.3}$$

$$W3 + W4 = 1 \tag{4.4}$$

In Equation 4.3 we took our first variable, the quantity of user feedback that the analysed APK has (c), to calculate the corresponding value of user feedback directly proportional to the maximum found in all APKs analysed. Since we wanted to use these variables in a similar way, the Equation 4.3 is an analogous equation to the one in 4.1. Hence the first part of the Equation 4.3 is meant to be used as a way to see the proportion of user feedback of one APK when compared with the sample. Then we took our second variable, the quantifiable type of user feedback (d), to calculate the quantifiable average of the users' feedback for the analysed APK (the quantifiable type of user feedback was the result of the numeric system implemented, explained above).

The AmountOfFlags in the Equation 4.3 refers to the total amount of flags for the APK analysed, as for the Max(AmountOfFlags) refers to the maximum total number of flags found during the analysis of a set of APKs, at the moment of the analysis of the APK currently reviewed. To each part of the equation we assigned a weight (W3 and W4) and then added the two parts.

Due to the appliance of our numeric system to the Aptoide user feedback system, if we joined the two equations (4.1 and 4.3), we could reach negative values that would be important to the determination of the threat level of an application. Hence our last equation was nonetheless the junction result of the two equations:

$$ThreatLevel = WeightedResult * W5 + UserFeedback * W6 \tag{4.5}$$

$$W5 + W6 = 1 \tag{4.6}$$

During our tests we assigned multiple combinations to the weights. These values were comprehended in the set 4.7. We decided to take a simplistic linear approach

to the values given to the different weights between the 3 formulas as a way to better understand our results.

$$W1, W2, W3, W4, W5, W6 \in [0.3, 0.4, 0.5, 0.6, 0.7] \tag{4.7}$$

$$WeightedResult, UserFeedback, ThreatLevel \in [0, 1] \tag{4.8}$$

Our last equation (4.5) will be the one to define the priority of the APKs for the manual inspection. And since the user feedback can have a negative value, the weighted result calculated in Equation 4.1 may decrease, adjusting the threat level according to the needs of the app store.

Despite the fact that the currently available flags do not focus in the report of programming malpractices, we believe that the negative ones we take into account (fake, freeze and virus) could detect problems that stemmed from the bad engineering decisions made by the developer.

- **Threat Notification**
  After the module concludes the attribution of a threat level to an APK by the former stage of this process, the module will give to the quality assurance team two forms of decision support. First if an APK receives a threat level higher than a critical point, e.g. over 80%, then the module sends an automatic e-mail to a member of the quality assurance team. This e-mail gives the quality assurance team information about the md5sum of the APK and the corresponding threat level, so that the manual review can take place. Secondly, the module provides an automatic service to get an ordered list of APKs by threat level.

  In conclusion, we based the proposed decision support on an ordered list of APKs and on notifications triggered by some conditions, for instance, when an APK gets a high threat level.

4.4 Database Structure

As the architecture in section 4.2 shows, this module will fetch the APKs that already went through the validation process and have bad user feedback. For that purpose we created an isolated data base for testing and to guarantee the micro services ideology. We made our database proposal considering the database used by the Aptoide micro services, to have an equal representation. Figure 4.2 shows the entity relationship data model for this database.

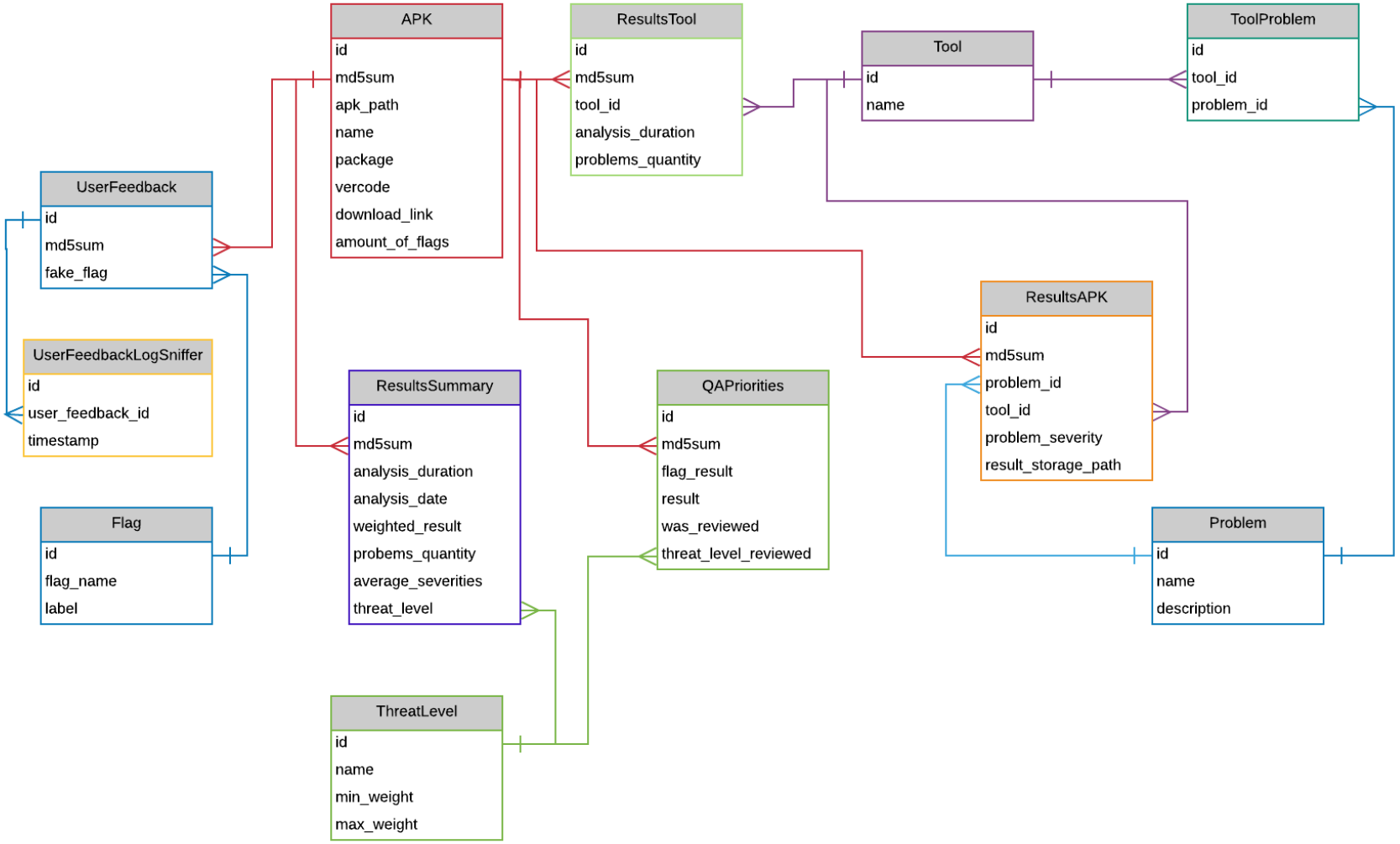


Figure 4.2: Proposed entity–relationship model for the module

The tables Flag, UserFeedback and APK were designed at the image of the original ones. The content of the tables Tool, ToolProblem, Problem, ThreatLevel and Flag is populated before the module can run:

- The Tool table is populated with the name of the tools that are used in the module;
- The Problem table is populated with the name and description of the security vulnerabilities we decided to analyse;
- The ToolProblem table is populated with the combinations between each tool and the security vulnerabilities they can catch;
- The ThreatLevel table is populated with the minimum and maximum values of each of the 10 threat levels;
- The Flag table is populated with the name and label of the five different types of flag.

The table APK stores information about the already validated APKs that have user feedback. In this table we store the name, package, version code and md5sum of the application and where it is stored and its download link. The table UserFeedback stores each entry of user feedback made to an application (recognised by its md5sum).

The tables ResultsTool, ResultsSummary, ResultsAPK and QAPriorities store the results of the analysis made to the applications and the outcome of that analysis:

- The ResultsTool table stores the amount of problems discovered by a certain tool during the analysis of a specific APK. It also stores the time it took to the tool to make that analysis.
- The ResultsSummary table stores a summary of the information retrieved from the analysis of a specific APK as well as when it happened and how much time it took.
- The ResultsAPK table stores the type and the severity of each security vulnerability found by each one of the 5 tools. It also stores the location of the output of the analysis (the non-processed output given by each tool).
- The QAPriorities table stores the threat level of each application as well as the information of if an APK was already reviewed.

Lastly, we have the UserFeedbackLogSniffer table, which stores the last entry of the UserFeedback table that was analysed by the module.

4.5 Used Technologies

To develop the solution, we used python as the programming language and used the PyCharm Professional IDE [Jet] as the tool to implement the module. We choose the Pycharm IDE not only because it offers multiple features (access the command line, connection to a database, it creates a virtual environment, and it manages the version control system all in one place) but also because this project was made from scratch.

We used MySQL together with peewee to implement the database structure described in section 4.4. Peewee is an ORM (Object Relational Mapping) written in python that provides a lightweight querying interface over SQL [Lei]. An ORM is a mechanism that makes it possible to address access and manipulate objects without having to consider how those objects relate to their data sources [Rou]; it makes data access more abstract and portable.

To retrieve the order APK list by threat level, we used Flask as the webservice. Flask is a web framework that provides tools, libraries and technologies that allow us to build a web application. Since it is a micro-framework it has little to no dependencies to external libraries, which makes it a light framework to use [Das].

The e-mail notifications sent from the module to the quality assurance team, about the APKs that have a high threat level were sent trough the use of an email service API, the mailgun [Mai].

Chapter 5

Evaluation

This chapter presents a set of test cases with the purpose of evaluate the functionality of the module as well as the evaluation of the previously defined metrics. To test the functionality of the module we started by defining use cases to ensure that the delivered software works properly. We performed three tests entitled "Sequential and Parallel Processing" (section 5.1), "Module Tools Composition" (see section 5.2), "Modules Accuracy" (see section 5.3) and "Metrics Results" (see section 5.4).

5.1 Sequential and Parallel Processing

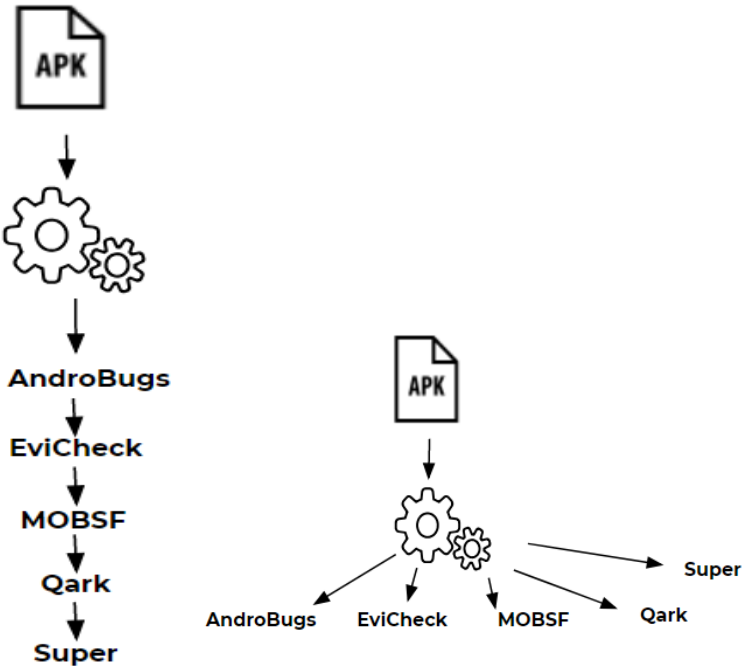
The performance of the module was tested by examining the modules execution time. To test the execution time, we gather 100 APKs from the Aptoides' repository and used two use cases (see subsection 5.1.2 and subsection 5.1.3) . The sample used for these two test cases is called Sample_1. The test case procedure we followed is represented in subsection 5.1.1.

We compared the modules results for the execution time between a sequential and a parallel composition of the tools, in order to understand how great was the time difference between the two of them (subsection 5.1.4). Since our results showed us that the time difference was only of 19%, we then proceeded to an analysis of the composition of the tools.

5.1.1 Test Case Procedure

In this section we present the test case model we followed to test the analysis time of the module. We had two scenarios: executing the module with the tools running in sequence Figure 5.1a and executing the module with the tools running in parallel Figure 5.1b.

However, the procedure and expected output of the test case were the same for both scenarios.



(a) Scenario 1 - Sequential

(b) Scenario 2 - Parallel

Figure 5.1: Scenarios to test the analysis time of the module [Ico; Teca]

Test case procedure

1. Use Sample_1 as as input for the module.

Expected Output

After the execution of the module there should be:

- 100 new rows in table UserFeedback with feedback information about 100 different APKs;
- 100 new rows in table APK with resumed information about the APK;
- 100 new rows in table ResultsSummary with the resumed information of the analysis;
- One row in table ResultsTool for each tool that analysed an APK with the time it took to analyse it;
- One row for each problem found per APK in ResultsAPK;

- 100 new rows in table QAPriorities, where each row contains a threat level associated with one APK;
- One row in UserFeedbackSniffer with information about the last row id seen in the UserFeedback table.

5.1.2 Measurement of the analysis time with a sequential composition of the tools

For this test the goal is to understand how much time does the module takes to analyse 100 APKS, as well as how much time it takes for each tool to analyse an APK, given that the tools will be operating sequentially.

After processing this test use case, we confirmed the expected outcome and we analysed the total time of the analysis of each analysed APK. The graphic in Figure 5.2 gives the execution time in minutes for each analysed APK. From the graphic we concluded that the total time of the analysis of the 100 APKs from the Sample_1 was of 20 hours and 45 minutes. Wherein, the average time of the analysis of one APK, by all tools of the module running in sequence, was of 12 minutes and 26 seconds.

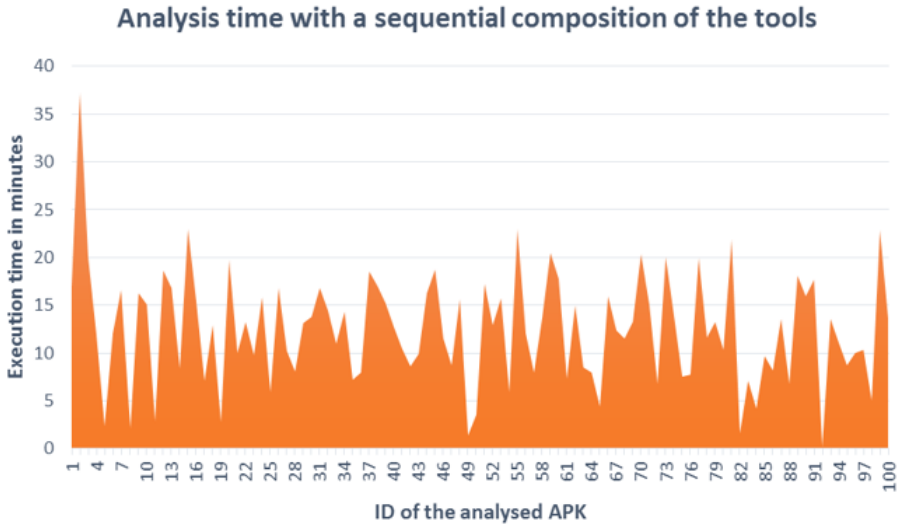


Figure 5.2: Graphic of the time analysis per APK with a sequential composition of the tools

5.1.3 Measurement of the analysis time with a parallel composition of the tools

For this test the goal is to understand how much time does the module takes to analyse 100 APKS, as well as how much time it takes for each tool to analyse an APK, given that the tools will be operating in parallel.

After processing this test use case, we also have confirmed the expected outcome and we analysed the total time of the analysis of each analysed APK. The graphic in Figure 5.3 gives the execution time in minutes for each analysed APK when the tools are executed in parallel. From the graphic we concluded that total time of the analysis of the 100 APKs from the Sample_1 was of 16 hours and 47 minutes. Wherein, the average time of the analysis of one APK, by all tools of the module running in parallel, was of 10 minutes and 4 seconds.

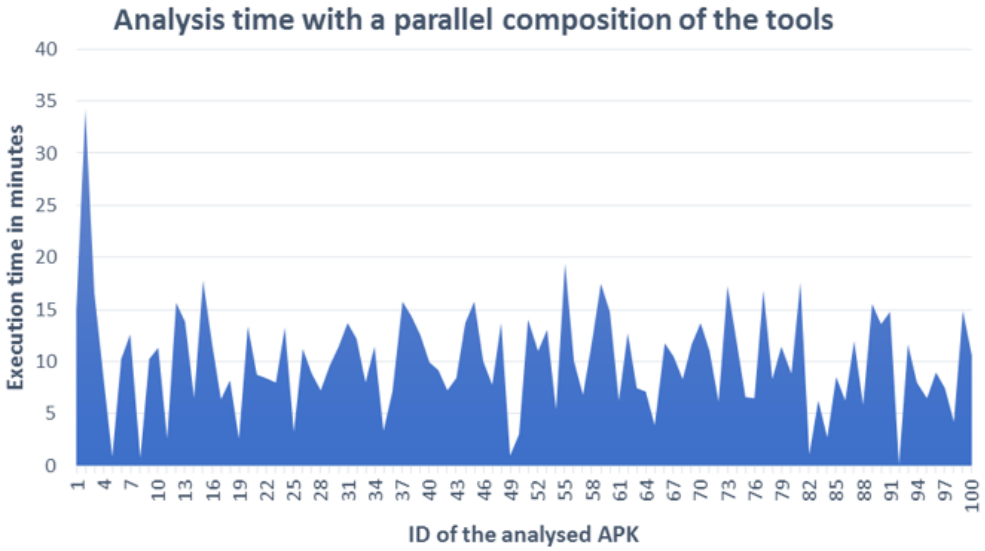


Figure 5.3: Graphic of the time analysis per APK with a parallel composition of the tools

By comparing the result of the two test use cases (subsection 5.1.2 and subsection 5.1.3) as the graphic in Figure 5.3 shows, we were able to conclude that the difference between a parallel and sequential execution of the tools in the module was only of 3 hours and 57 minutes, for the total analysis of the 100 APKs of the Sample_1. Which represents an average of less 2 minutes and 22 seconds per analysis.

Table 5.1 give us a summary of the total analysis time for each scenario and the average time per APK. We can see that the difference between the two scenarios for the total analysis time was of 3 hours and 58 minutes, wherein the average time per APK was of 2 minutes and 22 seconds. This represents a time difference of only 19%.

Scenario	Total time	Average time per APK
Scenario 1 (5.1.2)	20h 45m	12m 26s
Scenario 2 (5.1.3)	16h 47m	10m 4s
The Difference	3h 58m	2m 22s

Table 5.1: Summary of the analysis time between the two scenarios

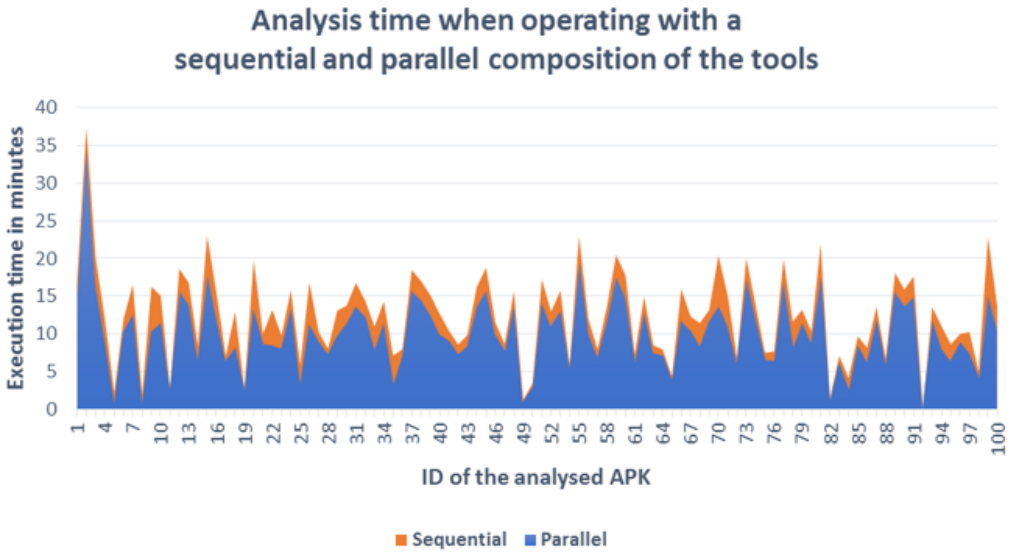


Figure 5.4: Graphic of the time analysis per APK comparing a parallel and sequential composition of the tools

5.1.4 Discussion of the Results

In addition to presenting the results obtained, it becomes necessary to discuss them. This discussion is made for the analysis time, taking into account the results obtained in each one of the use cases. Since this module would be part of the Aptoide's system it became crucial that the analysis time per APK was as less as possible.

Regarding the analysis time, and taking into account the results obtained for the two test use cases (subsection 5.1.2 and subsection 5.1.3), it was observed that:

- The analysis time is less when the tools run in parallel;
- The difference between an architecture in parallel and in sequence is not as great as it should be. The difference in the total time analysis between the two is of 3 hours and 57 minutes (which represents a difference of 19%).

Despite the fact that the analysis time difference between the two types of architecture is not much, we have decided to choose the architecture in parallel. We made this choice to comply with the aim of reducing the time of the analysis. However, and with that goal in mind, we still needed to make more tests to understand what was behind the increase of the analysis time. So we decided to make another test but this time with a parallel architecture of the tools and comparing the time of an analysis with all the tools and an analysis without the slower tool.

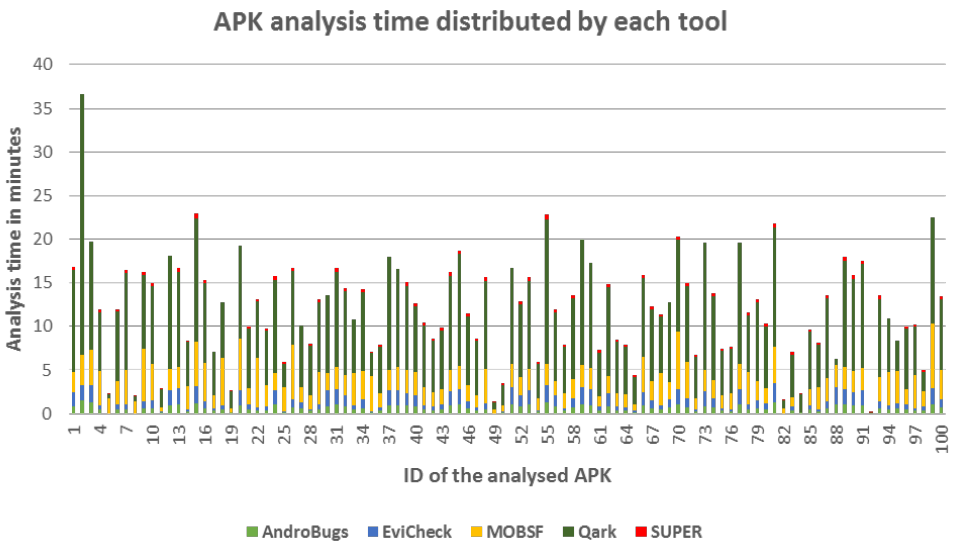


Figure 5.5: Analysis time of an APK distributed by the tools

To determine which tool was the slowest tool, we took a look at the APK analysis time distributed by each tool (see Figure 5.5) and we concluded that Qark was the one taking the most of the analysis time. In fact, according to Figure 5.6, Qark took 64% of the total analysis time.

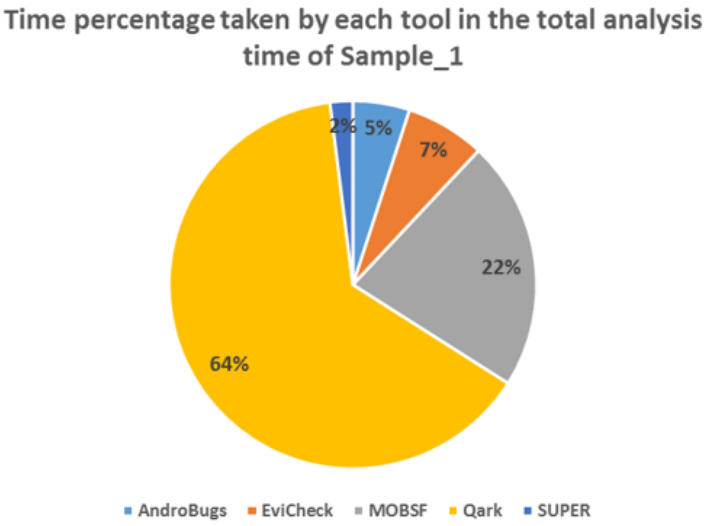


Figure 5.6: The percentage of the total time distributed by each tool of the module, after the analysis of Sample_1

Hence, after these results we decided to make more tests: one running an APK sample in a parallel architecture with all the tools and another where we remove Qark. However, these tests would only serve as a response to the question "Can we make a faster analysis?". We also have to take into account if removing Qark would affect our main purpose, that is, the rating of the threat level system.

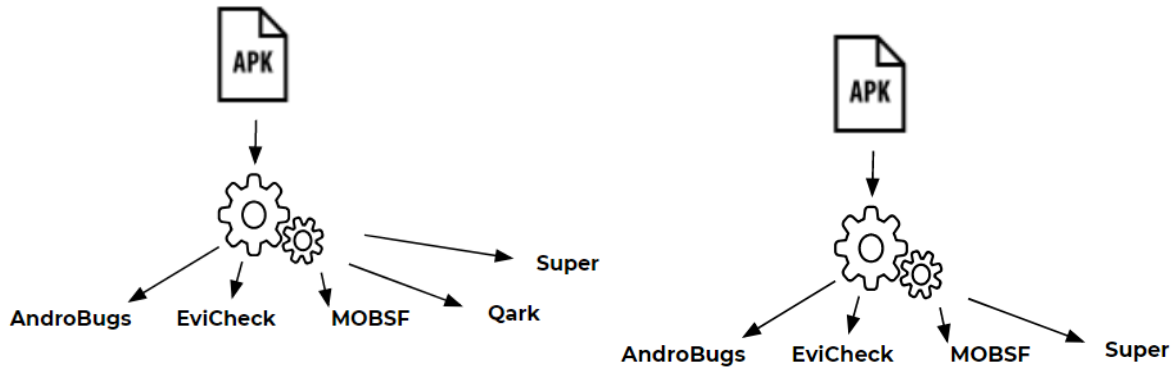
5.2 Module Tools Composition

To test the module tools composition, we have compared the execution time for a parallel analysis with all tools (see subsection 5.2.2) and removing the slower tool (see subsection 5.2.3). To test these two use cases, we gathered another sample, Sample_2, of 100 APKs from the Aptoides' back-office of the members of the quality assurance team. This back-office contains all the APKs in need of a manual review and the ones that were already reviewed by the QA team. In the collected sample we have 50 APKs that passed the quality assurance team, thus labelled as "not a threat", and 50 APKs that failed the quality assurance manual analysis (labelled as "a threat"). The purpose of this selection will be explained in section 5.3.

The results for the module tools composition test, made us believe that removing Qark would be a viable choice, since the time difference between an execution with all tools and removing Qark was about 55% less. However, we could not make this decision based only in the execution time, so we proceeded to test the accuracy of the module.

5.2.1 Test Case Procedure

In this section we present the test case model we followed to test the parallel composition of the tools in the module. We had two scenarios: executing the module with all tools running in parallel Figure 5.7a and executing the module after removing the slower tool and running in parallel Figure 5.7b. However, the procedure and expected output of the test case were the same for both scenarios.



(a) Scenario 1 - All tools

(b) Scenario 2 - Slowest tool removed

Figure 5.7: Scenarios to test the composition of the tools in the module [Ico; Teca]

Test case procedure

1. Use Sample_2 as as input for the module.

Expected Output

After the execution of the module there should be:

- 100 new rows in table UserFeedback with feedback information about 100 different APKs;
- 100 new rows in table APK with resumed information about the APK;
- 100 new rows in table ResultsSummary with the resumed information of the analysis;
- One row in table ResultsTool for each tool that analysed an APK with the time it took to analyse it;
- One row for each problem found per APK in ResultsAPK;
- 100 new rows in table QAPriorities, where each row contains a threat level associated with one APK;
- One row in UserFeedbackSniffer with information about the last row id seen in the UserFeedback table.

5.2.2 Measurement of the analysis time with a parallel composition of all tools

For this test the goal is to understand how much time does the module takes to analyse 100 APKS, as well as how much time it takes for each tool to analyse an APK, given that the tools will be operating parallel.

After processing this test use case, we confirmed the expected outcome, and we analysed the total time of the analysis of each analysed APK. The graphic in Figure 5.8 gives the execution time in minutes for each analysed APK. From the graphic we concluded that the total time of the analysis of the 100 APKs from the Sample_2 was of 14 hours and 36 minutes. Wherein, the average time of the analysis of one APK, by all tools of the module running in parallel, was of 8 minutes and 51 seconds.

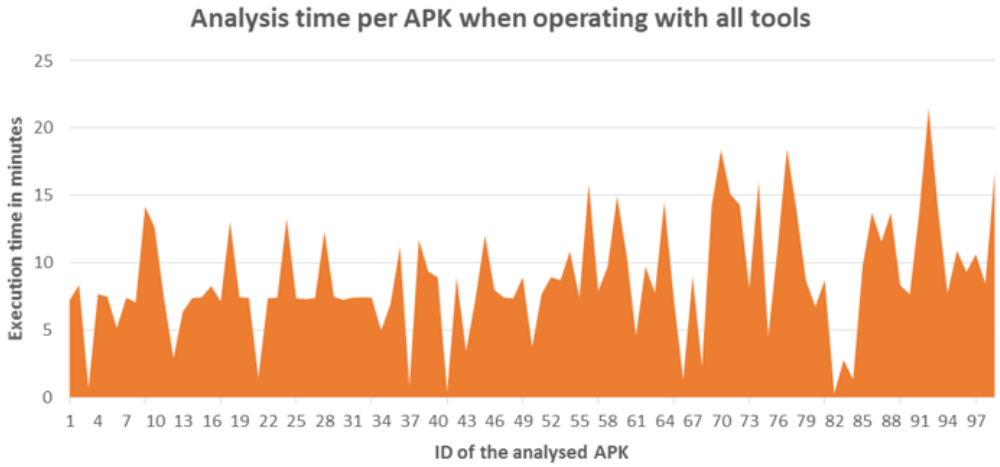


Figure 5.8: Graphic of the analysis time of the Sample_2, with a parallel composition of all tools

5.2.3 Measurement of the analysis time with a parallel composition, when removing Qark

For this test the goal is to understand how much time does the module takes to analyse 100 APKS, as well as how much time it takes for each tool to analyse an APK, given that the tools will be operating parallel and Qark is removed from the group of tools.

After processing this test use case, we confirmed the expected outcome and we analysed the total time of the analysis of each analysed APK. The graphic in Figure 5.9 gives the execution time in minutes for each analysed APK. From the graphic we concluded that the total time of the analysis of the 100 APKs from the Sample_2 was of 6 hours and 3 minutes. Wherein, the average time of the analysis of one APK, by all tools except Qark running in parallel, was of 3 minutes and 40 seconds.

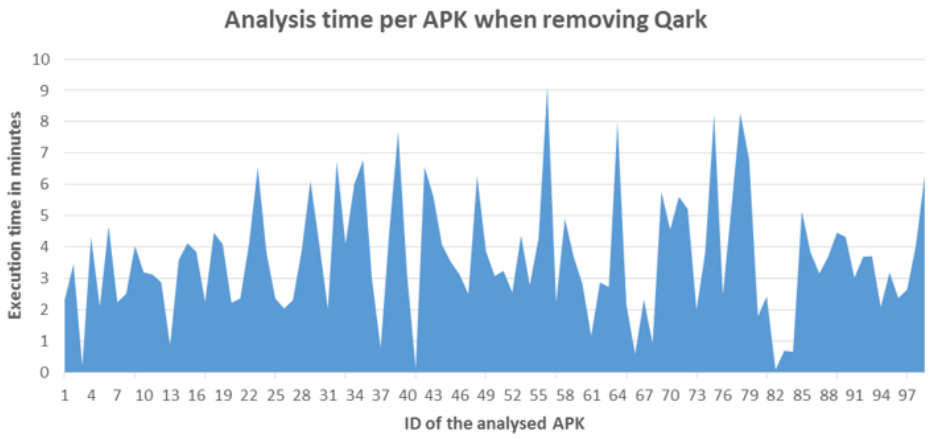


Figure 5.9: Graphic of the analysis time of the Sample_2, with a parallel composition the tools when removing Qark

By comparing the result of the two test use cases (subsection 5.2.2 and subsection 5.2.3) as the graphic in Figure 5.10 shows, we were able to conclude that the difference between a parallel execution with all tools and removing the slower tool was of 8 hours and 33 minutes, for the total analysis of the 100 APKs of the Sample_2. Which represents an average of less 5 minutes and 11 seconds per analysis.

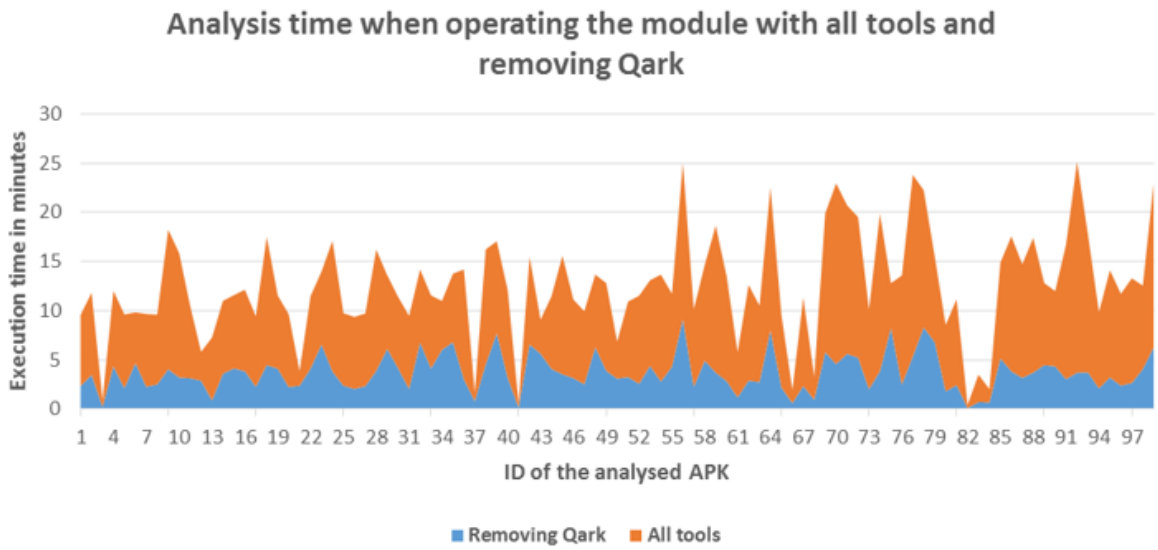


Figure 5.10: Graphic of the analysis time of the Sample_2, comparing the scenario with all tools and removing Qark

Table 5.2 give us a summary of the total analysis time for each scenario and the average time per APK. We can see that in the scenario where all tools are used the analysis time of the module is bigger by 59% then the scenario where the slowest tool is removed. This means that our module became 59% faster by removing the slowest tool.

Scenario	Total time	Average time per APK
Scenario 1 (5.2.2)	14h 36m	8m 51s
Scenario 2 (5.2.3)	6h 3m	3m 40s
The Difference	8h 33m	5m 11s

Table 5.2: Summary of the analysis time between the two scenarios

5.2.4 Discussion of the Results

After the tests we observed that Qark, in most cases, was the tool that took about 80% of the analysis time. So we contacted the developers, and we were informed that we were most likely to see a lot worse performance on obfuscated code. Even if there should be no infinite loops, it would take a long time to run due to the increased amount of files and code in general. And as the Figure 5.10 shows, this time the difference between the

two test cases was considerable. As we can see, there was a significant time reduction. The removal of the slowest tool, made it possible to analyse one APK in approximately 4 minutes in average (less 59%). Even if this removal made our solution more viable, we still needed to asses the accuracy of the module so this change would not compromise our goals.

5.3 Modules Accuracy

In order to understand if the decision of removing Qark was a viable choice we made a comparison based on how close is our threat level rating system to the accepted value. This is a measurement that requires accuracy. According to ISO 5725-1 [Iso], the general term accuracy is used to describe the closeness of a measurement to the true value.

In our context, the true value is if the analysed APK is a threat. For our threat level system to be accurate it would have to mean that it can identify properly the APKs that are a threat. For example, if we had 2 APKs, A and B, and if A is a threat but not B, then our module would have to give a higher threat level to A and a lower one to B.

To test the modules accuracy we used our Sample_2 since it was already manually reviewed by the quality assurance team. In that sample we have 50 APKs that passed the quality assurance team test, thus they were labelled as "not a threat". Since the other half of the sample failed the quality assurance manual analysis, this half was labelled as "a threat".

However, our threat rating system assigns a threat level between 0 and 1, which we then extend to a 1 to 10 scale. This means that to assess the accuracy we will have to look at the distribution of the APKs along that scale and compare them to the real labels. Since we have 50 APKs labelled as "a threat" and 50 APKs labelled as "not a threat", we have considered that in the perfect scenario the prioritised threat level list should have at the highest threat levels the 50 APKs labelled as "a threat", and only then the other 50 APKs labelled as "not a threat". In our proposal, the perfect scenario represents an accuracy of 100%. Wherein to calculate the accuracy of other scenarios we would just have to calculate the directly proportional quantity from the perfect scenario. Table 5.3 shows the scenario where the accuracy is of 100%, for 100 APKs.

Label	Top of the threat level list	Bottom of the threat level list
Not a threat	0	50
Threat	50	0

Table 5.3: Scenario for a 100% accuracy

In Table 5.4 we show an example of a scenario where the first 50 APKs of the prioritised list of APKs (ordered by threat level) consists of 10 APKs labelled as "not a threat" and 40 labelled as "a threat". In this scenario, the accuracy is of 80%.

Label	Top of the threat level list	Bottom of the threat level list
Not a threat	10	40
Threat	40	10

Table 5.4: Scenario for a 80% accuracy

For these tests (subsection 5.3.1 and subsection 5.3.2) the goal is to compare the accuracy between a parallel analysis with all tools and removing the slower tool. Despite all metrics have weights, in these tests we will focus in the weight variation of the equations: weighted result and threat level. We have chosen to do it so because the decision of removing a tool needs a study on the effect of the decreasing of the problems found.

Thus, in these tests we will use the five weight variations for the weighted result equation. We used the follow combination pairs: (0,3; 0,7), (0,4; 0,6), (0,5; 0,5), (0,6; 0,4) and (0,7; 0,3) to the weights W1 and W2 of the weighted result equation. To understand the impact of removing a tool from the module, we have chosen to maximise the impact of the problems found (the result of the weighted result). Hence, in the threat level equation the weights W5 and W6 will have the following pair: (0,7; 0,3). To the user feedback equation we gave the same weight value to both weights, (0,5;0,5). Table 5.5 shows a summary of the combinations to be assessed by the tests.

Weighted Result Equation	User Feedback Equation	Threat Level Equation
W1=0,3 and W2=0,7	*W3=0,5 and W4=0,5*	*W5=0,7 and W6=0,3*
W1=0,4 and W2=0,6	*W3=0,5 and W4=0,5*	*W5=0,7 and W6=0,3*
W1=0,5 and W2=0,5	*W3=0,5 and W4=0,5*	*W5=0,7 and W6=0,3*
W1=0,6 and W2=0,4	*W3=0,5 and W4=0,5*	*W5=0,7 and W6=0,3*
W1=0,7 and W2=0,3	*W3=0,5 and W4=0,5*	*W5=0,7 and W6=0,3*

Table 5.5: Weight combinations for the test cases

The subsection 5.3.1 is composed by the results of the 5 test cases presented in the table above, for the scenario where all tools are used (see subsections: 5.3.1.1, 5.3.1.2, 5.3.1.3, 5.3.1.4 and 5.3.1.5). For each test, there is a graphic and a detailed analysis of the results.

In subsection 5.3.2 there are also the results of the 5 test cases presented in the table above for the scenario where Qark is removed from the module (see subsections: 5.3.2.1, 5.3.2.2, 5.3.2.3, 5.3.2.4 and 5.3.2.5). For each test, there is a graphic and a detailed analysis of the results.

After we observed the results from subsection 5.3.1 and 5.3.2, we made a comparative analysis between each combination of the table 5.5. In subsection 5.3.3 is presented the comparative analysis, distributed in 5 subsections: 5.3.3.1, 5.3.3.2, 5.3.3.3, 5.3.3.4 and 5.3.3.5.

In conclusion, after observing the results, we removed Qark from the set of tools because the solution does not lose accuracy and because the tool does not support obfuscated code.

5.3.1 Evaluation of the accuracy of the module, when using all tools

Test case description

The goal is to evaluate the accuracy of the module, following the conditions explained above and using all the tools in the module.

Test case procedure

1. Use Sample_2 as as input for the module.
2. Assess the module accuracy for all the weight variables (see Table 5.5)

5.3.1.1 Modules accuracy for *W1=0,3 and W2=0,7; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

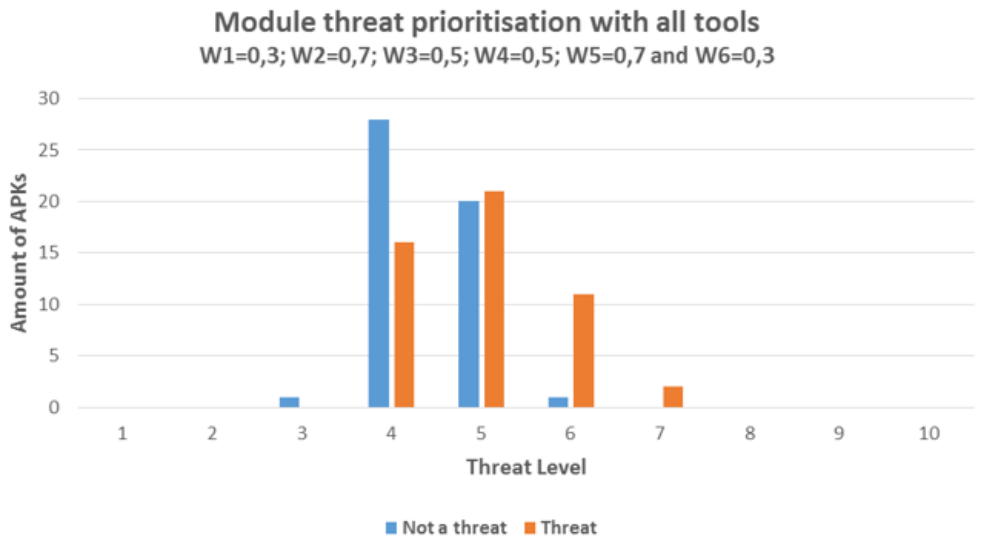


Figure 5.11: Threat prioritisation with all tools, for *W1=0,3 and W2=0,7; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

Looking at the graphic, we can conclude that the APKs labelled as a "threat" are distributed across the 3 highest threat levels (levels 5, 6 and 7). The total number of APKs in these 3 levels is of 55. Of those 55 APKs, 34 are labelled as a "threat" and the others 21 as "not a threat". This means that the module was able to identify 68% (34/50) of the total amount of APKs labelled as a threat. We can also conclude that for the two highest threat levels, 7 and 6, the solution erroneously tags 8% of the labelled "not a threat" APKs as "a threat" (1 out of 13).

The other threat levels (levels 3 and 4), count on 45 APKs: 29 labelled as not a "threat" and 16 as "a threat". Which means that the module was able to identify 58% (29/50) of the total amount of APKs labelled as not a threat. In the second lowest threat level (4) there are 44 APKs, of which 16 are labelled as a threat and 28 are labelled as not a threat. This means for the APKs scored in threat level 4 the module erroneously tags 36% of the labelled "threat" APKs as "not a threat" (16 out of 44).

5.3.1.2 Modules accuracy for *W1=0,4 and W2=0,6; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

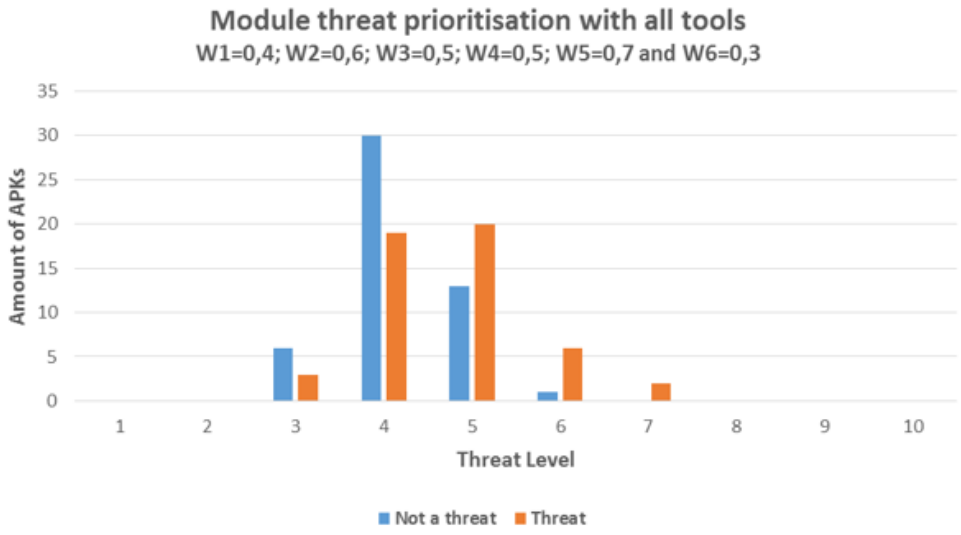


Figure 5.12: Threat prioritisation with all tools, for *W1=0,4 and W2=0,6; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

After the analysis of the Figure 5.12, it is concluded that the APKs labelled as a "threat" are distributed across the 3 highest threat levels (levels 5, 6 and 7). The total number of APKs in these 3 levels is of 42. Of those 42 APKs, 28 are labelled as a "threat" and the others 14 as "not a threat". This means that the module was able to identify 56% (28/50) of the total amount of APKs labelled as a threat. We can also conclude that for the two highest threat levels, 7 and 6, the solution erroneously tags 11% of the labelled "not a threat" APKs as "a threat" (1 out of 9).

The other threat levels (levels 3 and 4), count on 58 APKs: 22 labelled as a "threat" and 36 as "not a threat". Which means that the module was able to identify 72% (36/50) of the total amount of APKs labelled as not a threat. In the second lowest threat level (4) there are 49 APKs, of which 19 are labelled as a threat and 30 are labelled as not a threat. This means that for the APKs scored in threat level 4 the module erroneously tags 39% of the labelled "threat" APKs as "not a threat" (19 out of 49).

In the lowest threat level, the level 3, there are 9 APKs, of which 6 are labelled as not a threat and 3 are labelled as a threat. For the APKs scored in threat level 3, our solution

erroneously tags 50% of the labelled "threat" APKs as "not a threat" (3 out of 6).

5.3.1.3 Modules accuracy for *W1=0,5 and W2=0,5; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

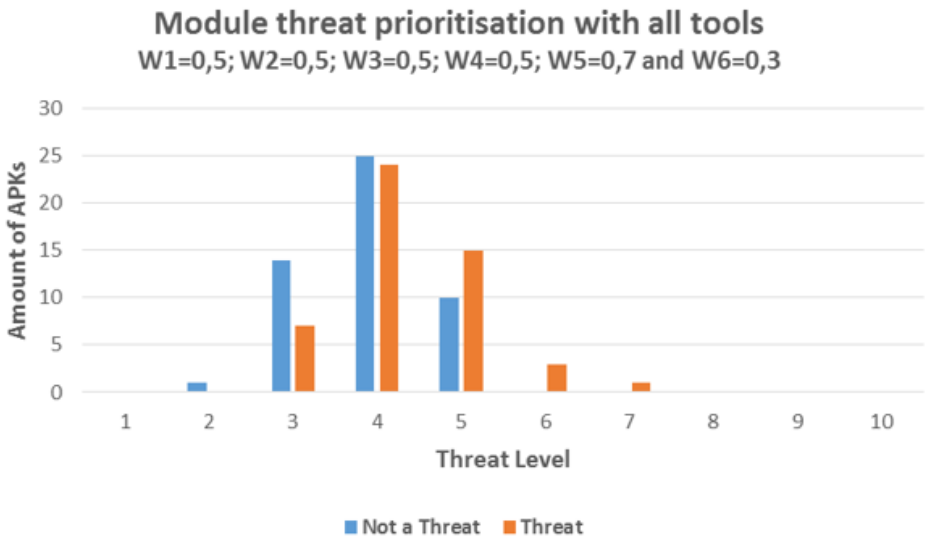


Figure 5.13: Threat prioritisation with all tools, for *W1=0,5 and W2=0,5; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

Looking at Figure 5.13, we can conclude that the APKs labelled as a "threat" are distributed across the two medium threat levels (levels 4 and 5). However, taking a closer look at threat level 4, the solution gave us 24 APKs labelled as a threat and 25 APKs labelled as not a threat. Since the amount of APKs labelled as a threat is almost the same as the ones labelled as not a threat, we can only conclude that the solution had no relevant accuracy for this intermediate threat level.

In the highest threat levels (levels 6 and 7) we have a total amount of 4 APKs labelled as a threat. This means that the module assigned the highest threat levels to 8% of the total 50 labelled threat APKs. For threat level 5, the module assigned 15 APKs labelled as a threat and 10 APKs labelled as not a threat. This means, that for this level, the solution erroneously tags 40% of the labelled "not a threat" APKs as "a threat" (10 out of 25).

The lowest threat levels (levels 2 and 3), counts on 22 APKs: 7 labelled as a "threat" and 15 as "not a threat". Which means that the module was able to identify 30% (15/50) of the total amount of APKs labelled as not a threat. In the second lowest threat level (3) there are 21 APKs, of which 7 are labelled as a threat and 14 are labelled as not a threat. This means that for the APKs scored in threat level 4 the module erroneously tags 33% of the labelled "threat" APKs as "not a threat" (7 out of 21).

5.3.1.4 Modules accuracy for *W1=0,6 and W2=0,4; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

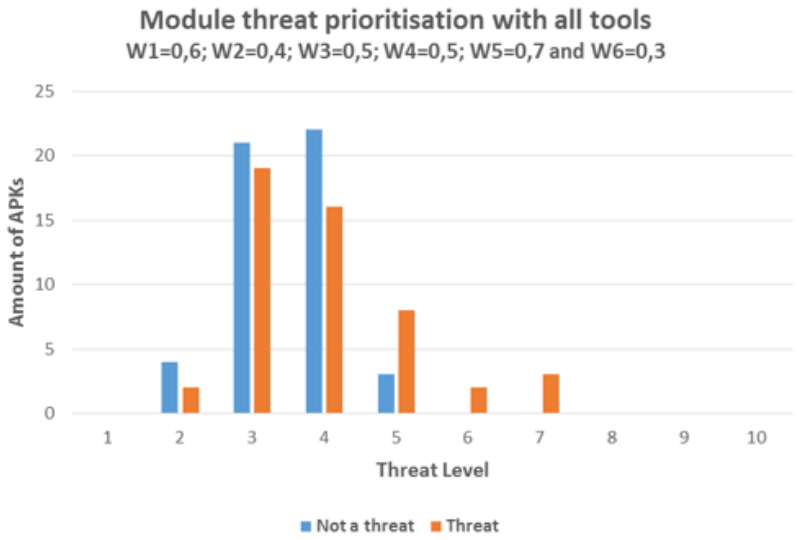


Figure 5.14: Threat prioritisation with all tools, for *W1=0,6 and W2=0,4; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

Figure 5.14 show us that the majority of the labelled threat and not a threat APKs are distributed in threat levels 3 and 4. In both levels the predominant APKs are the ones labelled as not a threat. In level 4 there are 38 APKs: 22 labelled as not a threat and 16 labelled as a threat. This means that our solution attributed a threat level of 4 to 32% of the labelled as a threat APKs and to 44% of the not a threat APKs. For threat level 3, there are 40 APKs: 21 labelled as not a threat (42% of the total amount of the APKs labelled as not a threat) and 19 labelled as a threat (38% of the total amount of the APKs labelled as a threat). Since only in these two threat levels, 3 and 4, the solution tags 70% of the APKs labelled as a threat and 86% of the APKs labelled as not a threat, we can conclude that the solution had no relevant accuracy for this intermediate threat levels.

In the highest threat levels (levels 6 and 7) we have a total amount of 5 APKs labelled as a threat. This means that the module assigned the highest threat levels to 10% of the total 50 labelled threat APKs. For threat level 5, the module assigned 8 APKs labelled as a threat and 3 APKs labelled as not a threat. This means, that for this level, the solution erroneously tags 27% of the labelled "not a threat" APKs as "a threat" (3 out of 11).

The lowest threat level (level 2), counts on 6 APKs: 2 labelled as a "threat" and 4 as "not a threat". Which means that the module was able to identify 8% (4/50) of the total amount of APKs labelled as not a threat and for the APKs scored in threat level 2 the module erroneously tags 33% of the labelled "threat" APKs as "not a threat" (2 out of 6).

5.3.1.5 Modules accuracy for *W1=0,7 and W2=0,3; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

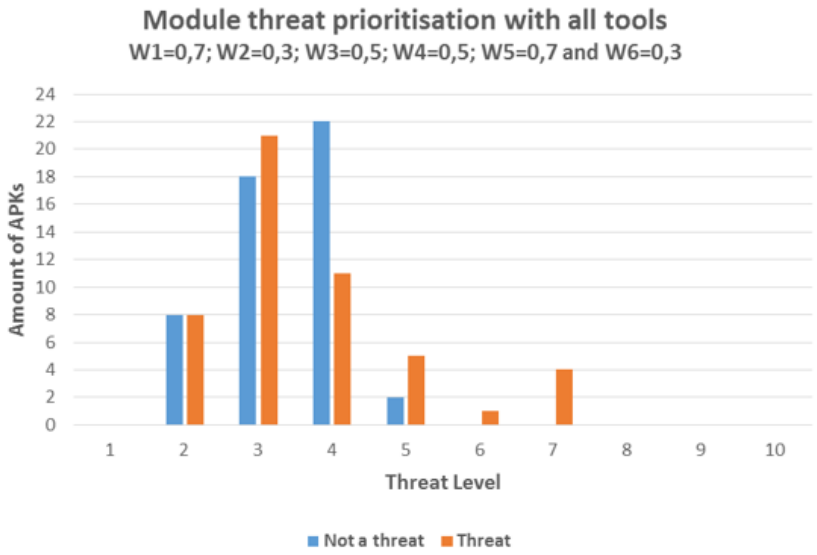


Figure 5.15: Threat prioritisation with all tools, for *W1=0,7 and W2=0,3; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

In Figure 5.15 we can see that the majority of the labelled threat and not a threat APKs are distributed in threat levels 3 and 4. In level 4 there are 33 APKs: 22 labelled as not a threat and 11 labelled as a threat. This means that our solution attributed a threat level of 4 to 22% of the labelled as a threat APKs and to 44% of the not a threat APKs. For threat level 3, there are 39 APKs: 18 labelled as not a threat (36% of the total amount of the APKs labelled as not a threat) and 21 labelled as a threat (42% of the total amount of the APKs labelled as a threat). Since only in these two threat levels, 3 and 4, the solution tags 64% of the APKs labelled as a threat and 80% of the APKs labelled as not a threat, we can conclude that the solution had no relevant accuracy for this intermediate threat levels.

In the highest threat levels (levels 6 and 7) we have a total amount of 5 APKs labelled as a threat. This means that the module assigned the highest threat levels to 10% of the total 50 labelled threat APKs. For threat level 5, the module assigned 5 APKs labelled as a threat and 2 APKs labelled as not a threat. This means, that for this level, the solution erroneously tags 29% of the labelled "not a threat" APKs as "a threat" (2 out of 7).

The lowest threat level (levels 2), counts on 16 APKs: 8 labelled as a "threat" and 8 as "not a threat". Which means that the module was able to identify 16% (8/50) of the total amount of APKs labelled as not a threat. However it also identified 16% (8/50) of the total amount of APKs labelled as a threat.

5.3.2 Evaluation of the accuracy of the module, when removing Qark

Test case description

The goal is to evaluate the accuracy of the module, following the conditions explained above and removing Qark from the tools.

Test case procedure

1. Use Sample_2 as as input for the module.
2. Assess the module accuracy for all the weight variables (see Table 5.5)

5.3.2.1 Modules accuracy for *W1=0,3 and W2=0,7; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

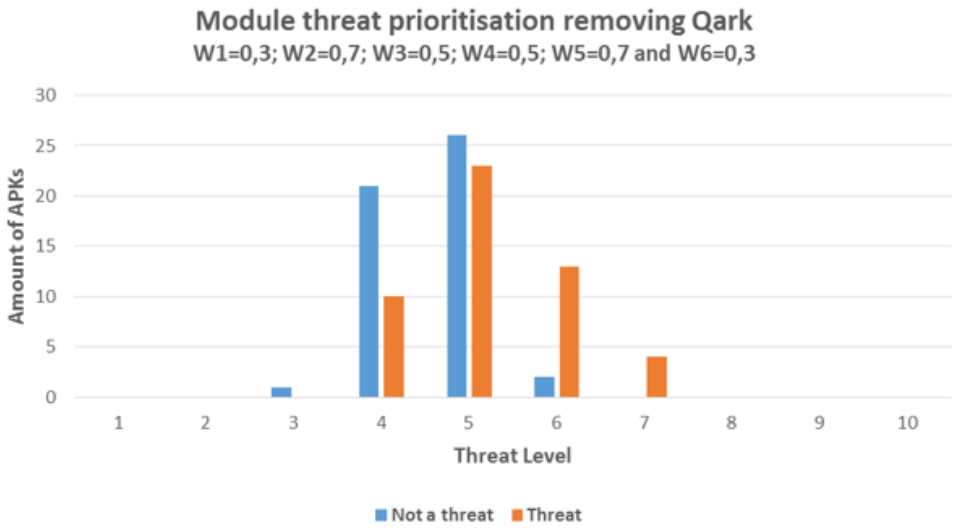


Figure 5.16: Threat prioritisation removing Qark, for *W1=0,3 and W2=0,7; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

Looking at the graphic, we can conclude that the APKs labelled as a "threat" are distributed across the 3 highest threat levels (levels 5, 6 and 7). The total number of APKs in these 3 levels is of 68. Of those 68 APKs, 40 are labelled as a "threat" and the others 28 as "not a threat". This means that the module was able to identify 80% (40/50) of the total amount of APKs labelled as a threat. We can also conclude that for the two highest threat levels, 7 and 6, the solution erroneously tags 11% of the labelled "not a threat" APKs as "a threat" (2 out of 19).

The other threat levels (levels 3 and 4), count on 32 APKs: 10 labelled as a "threat" and 22 as "not a threat". Which means that the module was able to identify 44% (22/50) of the total amount of APKs labelled as not a threat. In the second lowest threat level (4) there are 31 APKs, of which 10 are labelled as a threat and 21 are labelled as not a threat.

This means for the APKs scored in threat level 4 the module erroneously tags 48% of the labelled "threat" APKs as "not a threat" (10 out of 21).

5.3.2.2 Modules accuracy for *W1=0,4 and W2=0,6; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

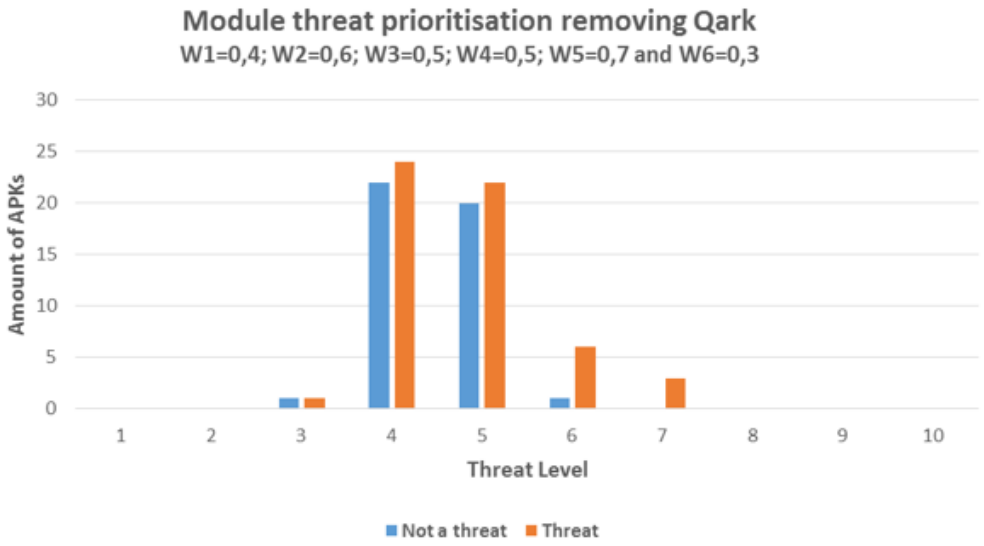


Figure 5.17: Threat prioritisation removing Qark, for *W1=0,4 and W2=0,6; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

After the analysis of the Figure 5.17, it is concluded that the APKs labelled as a "threat" are distributed across the 3 highest threat levels (levels 5, 6 and 7). The total number of APKs in these 3 levels is of 52. Of those 52 APKs, 31 are labelled as a "threat" and the others 21 as "not a threat". This means that the module was able to identify 62% (31/50) of the total amount of APKs labelled as a threat. We can also conclude that for the two highest threat levels, 7 and 6, the solution erroneously tags 10% of the labelled "not a threat" APKs as "a threat" (1 out of 10).

The other threat levels (levels 3 and 4), count on 48 APKs: 25 labelled as a "threat" and 23 as "not a threat". Which means that the module was able to identify 46% (23/50) of the total amount of APKs labelled as not a threat. In the second lowest threat level (4) there are 46 APKs, of which 24 are labelled as a threat and 22 are labelled as not a threat.

In the lowest threat level, the level 3, there are 2 APKs, of which one is labelled as a threat and one is labelled as not a threat.

5.3.2.3 Modules accuracy for *W1=0,5 and W2=0,5; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

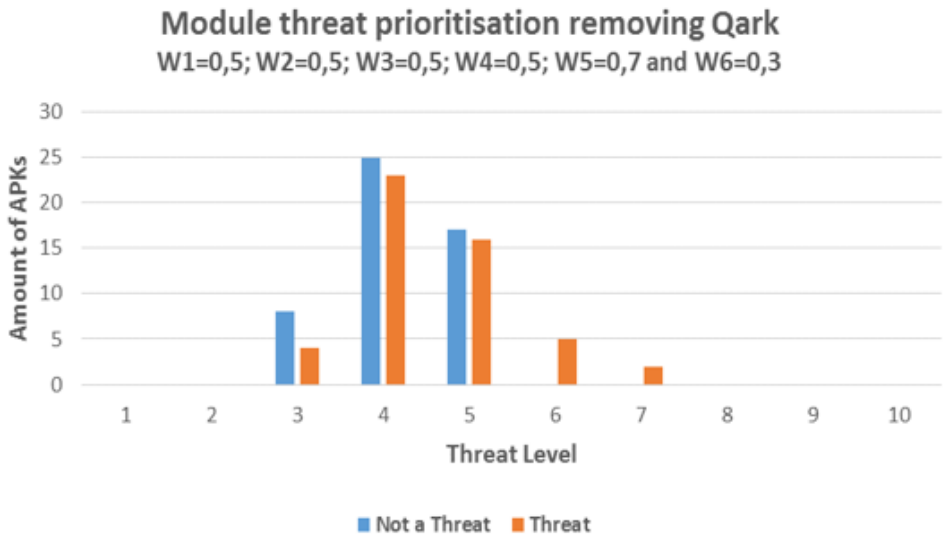


Figure 5.18: Threat prioritisation removing Qark, for *W1=0,5 and W2=0,5; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

Looking at Figure 5.18, we can conclude that the APKs labelled as a "threat" are distributed across the two medium threat levels (levels 4 and 5). However, taking a closer look at threat level 4, the solution gave us 23 APKs labelled as a threat and 25 APKs labelled as not a threat. Since the amount of APKs labelled as a threat is almost the same as the ones labelled as not a threat, we can only conclude that the solution had no relevant accuracy for this intermediate threat level.

In the highest threat levels (levels 6 and 7) we have a total amount of 7 APKs labelled as a threat. This means that the module assigned the highest threat levels to 14% of the total 50 labelled threat APKs. For threat level 5, the module assigned 16 APKs labelled as a threat and 17 APKs labelled as not a threat.

The lowest threat level (level 3), counts on 12 APKs: 4 labelled as a "threat" and 8 as "not a threat". Which means that the module was able to identify 16% (8/50) of the total amount of APKs labelled as not a threat, and that for the APKs scored in threat level 3 the module erroneously tags 33% of the labelled "threat" APKs as "not a threat" (4 out of 12).

5.3.2.4 Modules accuracy for *W1=0,6 and W2=0,4; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

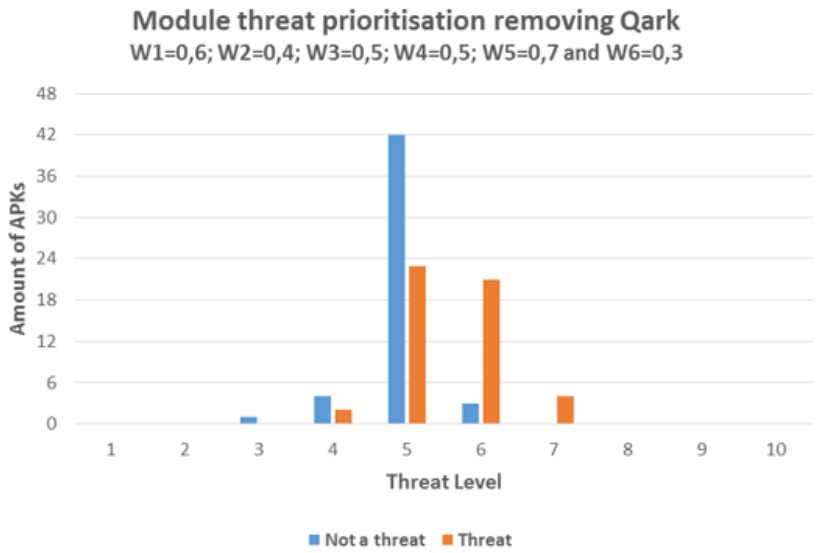


Figure 5.19: Threat prioritisation removing Qark, for *W1=0,6 and W2=0,4; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

Figure 5.19 show us that the majority of the APKs labelled as not a threat are in level 5. This threat level has a total of 65 APKs of the 100 analysed, there are 42 APKs labelled as not a threat (84%) and 23 APKs labelled as a threat (46%).

In the highest threat levels (levels 6 and 7) we have a total amount of 25 APKs labelled as a threat and 3 labelled as not a threat. This means that the module assigned the highest threat levels to 50% of the total 50 labelled threat APKs and the solution erroneously tags 11% of the labelled "not a threat" APKs as "a threat" (3 out of 28).

The lowest threat level (levels 3 and 4), counts on 7 APKs: 2 labelled as a "threat" and 5 as "not a threat". Which means that the module was able to identify 10% (5/50) of the total amount of APKs labelled as not a threat. For the APKs scored in threat level 4 the module erroneously tags 33% of the labelled "threat" APKs as "not a threat" (2 out of 6).

5.3.2.5 Modules accuracy for *W1=0,7 and W2=0,3; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

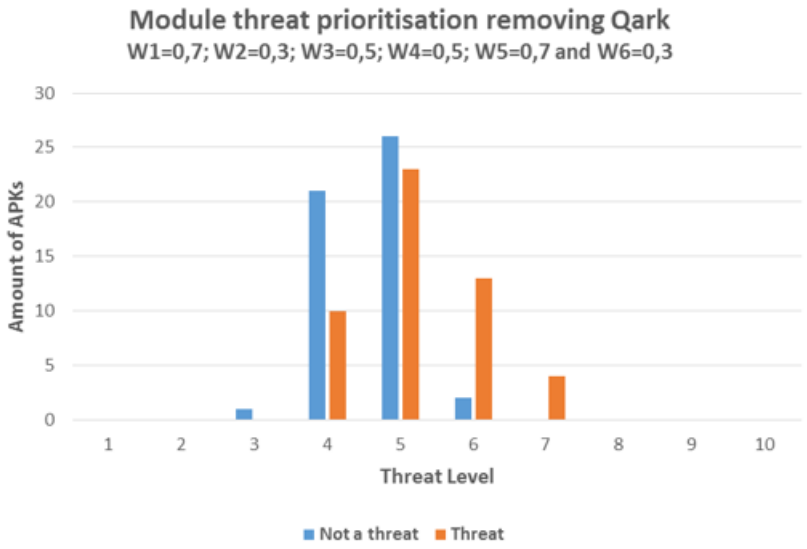


Figure 5.20: Threat prioritisation removing Qark, for *W1=0,7 and W2=0,3; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

In Figure 5.20 we can see that the majority of the labelled threat and not a threat APKs are distributed in threat level 5. In that level here are 49 APKs: 26 labelled as not a threat and 23 labelled as a threat. This means that our solution attributed a threat level of 5 to 52% of the labelled as not a threat APKs and to 46% of the labelled as a threat APKs. Since for this threat level (5) the solution tags almost half of our sample, we can conclude that the solution had no relevant accuracy for this intermediate threat level.

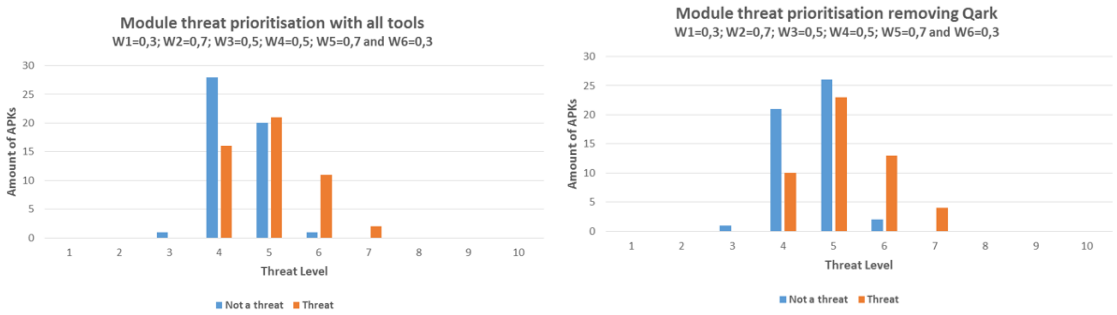
In the highest threat levels (levels 6 and 7) we have a total amount of 17 APKs labelled as a threat. This means that the module assigned the highest threat levels to 34% of the total 50 labelled threat APKs. For threat level 6, the module assigned 13 APKs labelled as a threat and 2 APKs labelled as not a threat. This means, that for this level, the solution erroneously tags 13% of the labelled "not a threat" APKs as "a threat" (2 out of 15).

The lowest threat levels (level 3 and 4), counts on 32 APKs: 10 labelled as a "threat" and 22 as "not a threat". For level 4, the module was able to identify 42% (21/50) of the total amount of APKs labelled as not a threat. However it also identified 20% (10/50) of the total amount of APKs labelled as a threat.

5.3.3 Comparative analysis of the accuracy results

After we collected all the results for the tests in subsection 5.3.1 and subsection 5.3.2 we made a comparative analysis between the two of them. We will make this comparison considering how important is the distribution of the APKs by the 10 threat levels for the decision support mechanism. When we analyse 100 APKs, the prioritised threat list that results from that analysis will be ordered by threat level. So the first 50 APKs, will be the ones with the highest threat level, thus the ones to be first analysed by the quality assurance team. Hence, the solution with more APKs labelled as a threat in the first half of the prioritisation, is more efficient since the quality assurance team will review more bad APKs in first place.

5.3.3.1 Comparative analysis for *W1=0,3 and W2=0,7; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*



(a) Accuracy with all tools

(b) Accuracy removing Qark

Figure 5.21: Modules accuracy for *W1=0,3 and W2=0,7; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

By looking at Figure 5.21a and Figure 5.21b, we can see that the first 50 APKs are distributed in threat levels 5, 6 and 7. Table 5.6 give us a summary of the amount of APKs labelled as a threat by the solution in the two different scenarios.

Threat Level	Figure 5.21a	Figure 5.21b
7	4% (2/50)	8% (4/50)
6	22% (11/50)	26% (13/50)
5	42% (21/50)	46% (23/50)
Total	68% (34/50)	80% (40/50)

Table 5.6: Comparative analysis of the highest threat levels for the labelled threat APKs

From the table above, we can assess that for the 3 highest threat levels (5, 6 and 7) the solution identifies more APKs labelled as a threat in the scenario where the slower toll is removed. However we can also see that, for the same 3 threat levels the solution loses accuracy, since for the scenario 5.21a we have 62% (34/55) of labelled threat APKs and

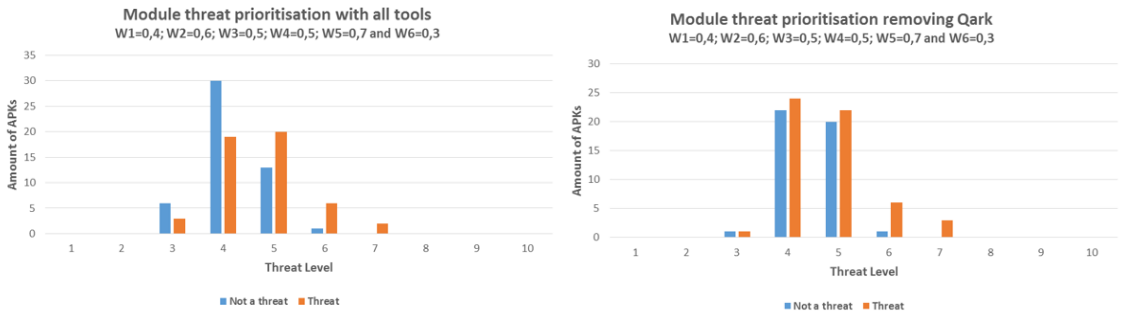
38%(21/55) of labelled as not a threat APKs; whereas for scenario 5.21b we have 59% (40/68) of labelled threat APKs and 41% (28/68) of APKs labelled as not a threat.

Threat Level	Figure 5.21a	Figure 5.21b
4	56% (28/50)	42% (21/50)
3	2% (1/50)	2% (1/50)
Total	58% (29/50)	44% (22/50)

Table 5.7: Comparative analysis of the lowest threat levels for the labelled not a threat APKs

The Table 5.7 give us a summary of the amount of APKs labelled as not a threat by the solution in the two different scenarios. We can assess that for the 2 lowest threat levels (3 and 4) the solution identifies more APKs labelled as not a threat in the scenario where all the tools are operable. However we can also see that, for the same 2 threat levels the winning solution loses accuracy, since for the scenario 5.21a we have 64%(29/45) of labelled not a threat APKs and 36%(16/45) of labelled as a threat APKs; whereas for scenario 5.21b we have 69% (22/32) of labelled not a threat APKs and 31% (10/32) of APKs labelled as a threat.

5.3.3.2 Comparative analysis for *W1=0,4 and W2=0,6; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*



(a) Accuracy with all tools

(b) Accuracy removing Qark

Figure 5.22: Modules accuracy for *W1=0,3 and W2=0,7; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

By looking at Figure 5.22a and Figure 5.22b, we can see that the first 50 APKs are distributed in threat levels 5, 6 and 7. Table 5.8 give us a summary of the amount of APKs labelled as a threat by the solution in the two different scenarios.

Threat Level	Figure 5.22a	Figure 5.22b
7	4% (2/50)	6% (3/50)
6	12% (6/50)	12% (6/50)
5	40% (20/50)	44% (22/50)
Total	56% (28/50)	62% (31/50)

Table 5.8: Comparative analysis of the highest threat levels for the labelled threat APKs

From the table above, we can assess that for the 3 highest threat levels (5, 6 and 7) the solution identifies more APKs labelled as a threat in the scenario where the slower toll is removed. However we can also see that, for the same 3 threat levels the solution loses accuracy, since for the scenario 5.21b we have 60% (31/52) of labelled threat APKs and 40%(21/52) of labelled as not a threat APKs; whereas for scenario 5.22a we have 67% (28/42) of labelled threat APKs and 33% (14/42) of APKs labelled as not a threat.

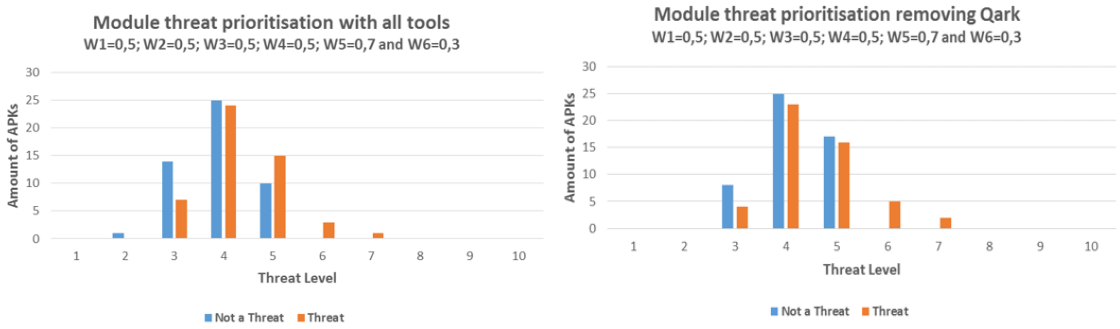
Threat Level	Figure 5.22a	Figure 5.22b
4	60% (30/50)	44% (22/50)
3	12% (6/50)	2% (1/50)
Total	72% (36/50)	46% (23/50)

Table 5.9: Comparative analysis of the lowest threat levels for the labelled not a threat APKs

The Table 5.9 give us a summary of the amount of APKs labelled as not a threat by the solution in the two different scenarios. We can assess that for the 2 lowest threat levels (3 and 4) the solution identifies more APKs labelled as not a threat in the scenario where all the tools are operable.We can also see that, for the same 2 threat levels the winning solution does not loses accuracy when compared with the other. Since for the scenario 5.22a we have 38%(22/58) of labelled threat APKs and 62%(36/58) of labelled as not a threat APKs; whereas for scenario 5.22b we have 48% (23/48) of labelled not a threat APKs and 52% (25/48) of APKs labelled as a threat.

5.3.3.3 Comparative analysis for *W1=0,5 and W2=0,5; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

Figure 5.23a and Figure 5.23b, show us that the first 50 APKs are distributed in threat levels 4, 5, 6 and 7. Table 5.10 give us a summary of the amount of APKs labelled as a threat by the solution in the two different scenarios.



(a) Accuracy with all tools

(b) Accuracy removing Qark

Figure 5.23: Modules accuracy with all the tools and removing the slower tool

From the table below, we can assess that for the 3 highest threat levels (5, 6 and 7) the solution identifies more APKs labelled as a threat in the scenario where the slower toll is removed. However we can also see that, for the same 3 threat levels the solution loses accuracy, since for the scenario 5.23b we have 58% (23/40) of labelled threat APKs and 43%(17/40) of labelled as not a threat APKs; whereas for scenario 5.23a we have 66% (19/29) of labelled threat APKs and 34% (10/29) of APKs labelled as not a threat.

Threat Level	Figure 5.23a	Figure 5.23b
7	2% (1/50)	4% (2/50)
6	6% (3/50)	10% (5/50)
5	30% (15/50)	32% (16/50)
4	48% (24/50)	46% (23/50)
Total	86% (43/50)	92% (46/50)

Table 5.10: Comparative analysis of the highest threat levels for the labelled threat APKs

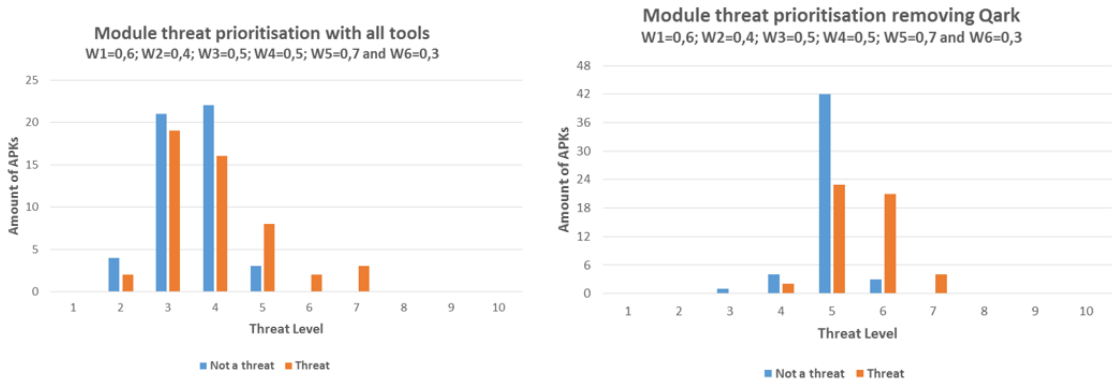
We can also conclude that the solution, for both scenarios, has no relevant accuracy for the intermediate threat level (level 4). Since the amount of APKs labelled as threat and not a threat are very close to each other. The Table 5.11 give us a summary of the amount of APKs labelled as not a threat by the solution in the two different scenarios. We can assess that for the 2 lowest threat levels (2 and 3) the solution identifies more APKs labelled as not a threat in the scenario where all the tools are operable. We can also see that, for the same 2 threat levels the winning solution does not loses accuracy when compared with the other. Since for the scenario 5.23a we have 32%(7/22) of labelled threat APKs and 68%(15/22) of labelled as not a threat APKs; whereas for scenario 5.23b we have 33% (4/12) of APKs labelled as a threat and 66% (8/12) of labelled not a threat APKs.

Threat Level	Figure 5.23a	Figure 5.23b
3	28% (14/50)	16% (8/50)
2	2% (1/50)	0% (0/50)
Total	30% (15/50)	16% (8/50)

Table 5.11: Comparative analysis of the lowest threat levels for the labelled not a threat APKs

5.3.3.4 Comparative analysis for *W1=0,6 and W2=0,4; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

After a quick analysis of the Figure 5.24a and Figure 5.24b, we can see that both have different threat level distributions for the 100 analysed APKs. We can see that for the scenario where the slower tool is removed, we have most of our APKs in threat level 5 (42 labelled as not a threat and 23 labelled as a threat). To compare the APK distribution of the first 50 APKs we have to consider that for the scenario in 5.24a, those 50 APKs are distributed between the threat levels 4, 5, 6 and 7. Whereas for the scenario in 5.24b those 50 APKs are distributed between the threat levels 5, 6 and 7.



(a) Accuracy with all tools

(b) Accuracy removing Qark

Figure 5.24: Modules accuracy with all the tools and removing the slower tool

From Table 5.12 we can assess that for the scenario where all the tools are used, there are 4 threat levels assigned to the first half of the prioritised APK list. In those threat levels (4, 5, 6 and 7), the solution erroneously tags 46% of the labelled "not a threat" APKs as "a threat" (25 out of 54). For the scenario where the slowest tool is removed, there are 3 threat levels assigned to the first half of the prioritised APK list. In those threat levels (5, 6 and 7), the solution erroneously tags 48% of the labelled "not a threat" APKs as "a threat" (45 out of 93).

Since Figure 5.24b does not have APKs in threat level 2 and the threat level 4 of Figure 5.24a still belongs to the first 50 APKS of the prioritised list, to compare the lowest threat levels between the two scenarios we have to look at: threat levels 2 and 3 for the

Threat Level	Figure 5.24a	Figure 5.24b
7	6% (3/50)	8% (4/50)
6	4% (2/50)	42% (21/50)
5	16% (8/50)	46% (23/50)
4	32% (16/50)	Will not be considered
Total	58% (29/50)	96% (48/50)

Table 5.12: Comparative analysis of the highest threat levels for the labelled threat APKs

scenario where all tools are used and to threat levels 3 and 4 for the scenario where the slowest tool is removed.

Threat Level	Figure 5.24a
3	42% (21/50)
2	8% (4/50)
Total	50% (25/50)

Table 5.13: Lowest threat levels for the scenario with all tools

For the lowest threat levels (2 and 3) in scenario 5.24a, Table 5.13 show us that the solution identified half of the APKs labelled as not a threat. However it also identified 42% (21/50) of the labelled threat APKs. We can also conclude that for the two lowest threat levels, 3 and 2, the solution erroneously tags 46% of the labelled "threat" APKs as "not a threat" (21 out of 46).

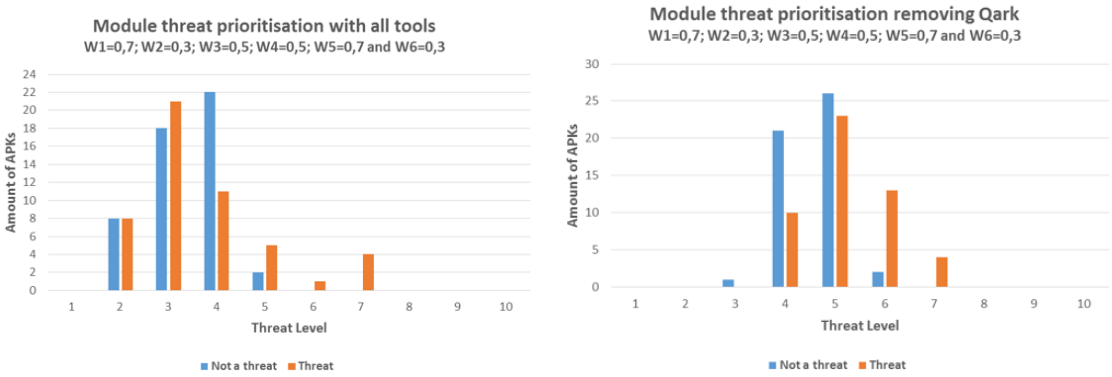
Threat Level	Figure 5.24b
4	28% (4/50)
3	2% (1/50)
Total	10% (5/50)

Table 5.14: Lowest threat levels for the scenario where the slowest tool was removed

Table 5.14 show us the results of the lowest threat levels (3 and 4) for the scenario 5.24b. We can conclude that the solution identified 10% of the APKs labelled as not a threat. However it also identified 4% (2/50) of the labelled threat APKs. We can also conclude that for the two lowest threat levels, 4 and 3, the solution erroneously tags 29% of the labelled "threat" APKs as "not a threat" (2 out of 7).

5.3.3.5 Comparative analysis for *W1=0,7 and W2=0,3; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*

After an analysis of the Figure 5.25a and Figure 5.25b, we can see that both have different threat level distributions for the 100 analysed APKs. To compare the APK distribution of the first 50 APKs we have to consider that for the scenario in 5.25a, those 50 APKs are distributed between the threat levels 4, 5, 6 and 7. Whereas for the scenario in 5.25b those 50 APKs are distributed between the threat levels 5, 6 and 7.



(a) Accuracy with all tools

(b) Accuracy removing Qark

Figure 5.25: Modules accuracy with all the tools and removing the slower tool

From Table 5.15 we can assess that for the scenario where all the tools are used, there are 4 threat levels assigned to the first half of the prioritised APK list. In those threat levels (4, 5, 6 and 7), the solution erroneously tags 53% of the labelled "not a threat" APKs as "threat" (24 out of 45). For the scenario where the slowest tool is removed, there are 3 threat levels assigned to the first half of the prioritised APK list. In those threat levels (5, 6 and 7), the solution erroneously tags 38% of the labelled "not a threat" APKs as "threat" (26 out of 68).

Threat Level	Figure 5.25a	Figure 5.25b
7	8% (4/50)	8% (4/50)
6	2% (1/50)	26% (13/50)
5	10% (5/50)	46% (23/50)
4	22% (11/50)	Will not be considered
Total	42% (21/50)	80% (40/50)

Table 5.15: Comparative analysis of the highest threat levels for the labelled threat APKs

To compare the lowest threat levels we had also to consider the different distribution of the threat levels by both scenarios. So Table 5.16 give us a summary of the amount of APKs labelled as not a threat by the solution, but having different threat levels for both scenarios. We can assess that for the threat levels 2 and 3, regarding the scenario in 5.25a,

the solution loses accuracy since it identifies more APKs labelled as a threat than APKs labelled as not a threat.

For the threat levels 3 and 4, regarding the scenario in 5.25b, the solution identifies more APKs labelled as not a threat than APKs labelled as a threat. We can also see that, for these 2 threat levels the winning solution does not loses accuracy when compared with the other. Since we have 31%(10/32) of labelled threat APKs and 69%(22/32) of labelled as not a threat APKs; whereas for scenario 5.23a we have 53% (29/55) of APKs labelled as a threat and 47% (26/55) of labelled not a threat APKs.

Threat Level	Figure 5.25a	Figure 5.25b
4	Will not be considered	16% (21/50)
3	36% (18/50)	2% (1/50)
2	16% (8/50)	0% (0/50)
Total	52% (26/50)	44% (22/50)

Table 5.16: Comparative analysis of the lowest threat levels for the labelled not a threat APKs

5.3.4 Discussion of the Results

After comparing each weight combination in the two different scenarios, we made conclusions based on the capacity of the solution to assign high threat levels to APKs that have been identified as a threat and low levels to those that have been identified as not a threat:

- For the weight combination: *W1=0,3 and W2=0,7; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*
  In graphic 5.21b, the module give us more APKs labelled as a threat for the highest threat levels (levels 5, 6 and 7). However this result has less 3% accuracy than the result in graphic 5.21a, when giving a high threat level to an APK labelled as a threat. We also verified that for the scenario where the slowest tool is removed, the solution has more accuracy than the other from 5.21a.

- For the weight combination: *W1=0,4 and W2=0,6; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*
  In graphic 5.22b, the module give us more APKs labelled as a threat for the highest threat levels (levels 5, 6 and 7). However this result has less 7% accuracy than the result in graphic 5.22a, when giving a high threat level to an APK labelled as a threat.

- For the weight combination: *W1=0,5 and W2=0,5; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*
  For both graphics, 5.23a and 5.23b, looking at threat level 4 there was no relevant

accuracy. Despite the graphic in 5.23b has more APKs labelled as a threat for the highest threat levels (levels 5, 6 and 7), it is also concluded that it loses accuracy (less 8%).

- For the weight combination: *W1=0,6 and W2=0,4; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*
  In graphic 5.24b, the module give us more APKs labelled as a threat for its highest threat levels (levels 5, 6 and 7). However this result has less 2% accuracy than the result in graphic 5.24a, when giving a high threat level to an APK labelled as a threat. We also verified that for the scenario where the slowest tool is removed, the solution has more accuracy than the other from 5.24a.

- For the weight combination: *W1=0,7 and W2=0,3; W3=0,5 and W4=0,5; W5=0,7 and W6=0,3*
  In graphic 5.25b, the module give us more APKs labelled as a threat for its highest threat levels (levels 5, 6 and 7). However this result has less 2% accuracy than the result in graphic 5.25a, when giving a high threat level to an APK labelled as a threat. We also verified that for the scenario where the slowest tool is removed, the solution has more accuracy than the other from 5.25a.

In conclusion, our analysis based on these graphics is that the proposed solution give us more APKs labelled as a threat in the highest threat levels, in the scenario where the slowest tool is removed. However when we compare the accuracy of the module on assigning threat levels between the two situations, we assessed that in the situation where all tools are operable the accuracy is higher. We could also see that the solution can inform with considerable certainty that APKs with a high threat level, calculated by the module, are bad.

To support our decision to remove Qark from the selected tools we have also contacted the developers of the tool and we were informed that we were most likely to see a lot worse performance on obfuscated code. Even if there should not be infinite loops, it would take a long time to run due to the increased amount of files and code in general.

After analysing all the facts, we have removed QARK from the modules integration.

5.4 Metrics Results

After the removal of Qark from the set of tools, we decided to make another test case with a new sample (Sample_3). The goal was to analyse the different results produced by the module when we have all the possible combinations within the set of values {0,3; 0,4; 0,5; 0,6; 0,7} and respecting the conditions of our metrics.

This new sample, Sample_3, was collected the same way as Sample_2. Therefore, this sample has 100 APKs that were manually reviewed by the quality assurance team. In the sample we have 50 APKs that passed the quality assurance test, thus they were labelled

as "not a threat". Since the other half of the sample failed the quality assurance manual analysis, that half is labelled as "a threat".

Since the analysis of the different results produced by the module will give us a better understanding on which of the weight combinations have a better accuracy result, we decided to test all 125 combinations during the analysis of the metrics results. The results of the 125 weight combinations are available in a shared folder.

We divided our analysis into 5 groups and compared, for each group, the results obtained in order to select the best one. This comparison was made based on the amount of APKs labelled as a threat, distributed in the first half of the prioritised threat level list. This also means that we have chosen the results that gave us more APKs labelled as a threat in highest threat levels. Figure 5.26, Figure 5.27, Figure 5.28, Figure 5.29 and Figure 5.30 represent the best results from each group. Below we make a descriptive analysis to each one of them.

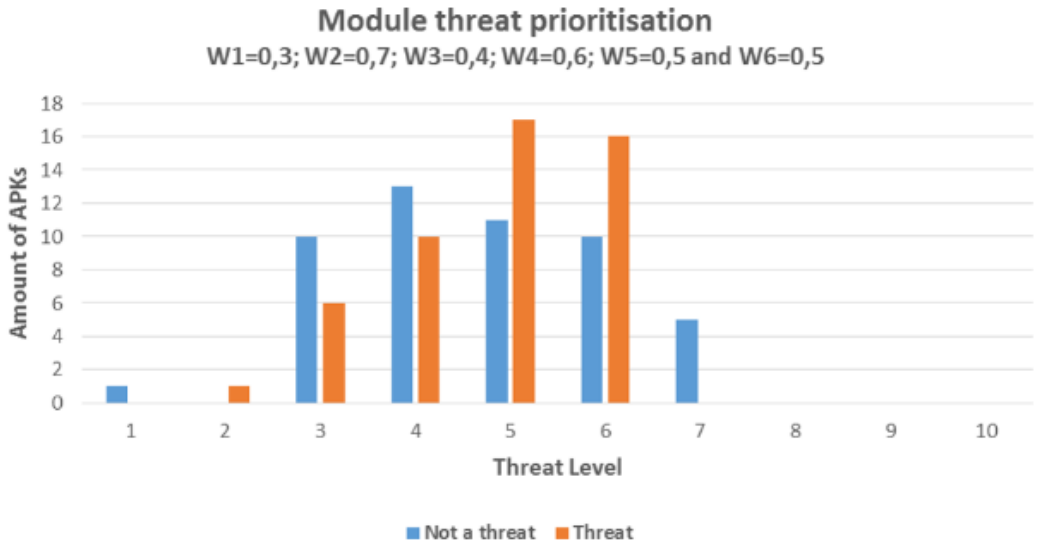


Figure 5.26: Threat prioritisation for Sample_3 *W1=0,3 and W2=0,7; W3=0,4 and W4=0,6; W5=0,5 and W6=0,5*

Looking at Figure 5.26, we observed that the first half of the prioritisation of the module is composed by 44% of the total amount of APKs labelled as not a threat and 56% of the total amount of APKs labelled as a threat. We can also observe that in the highest threat level (level 7) we have 10% of the total amount of APKs labelled as not a threat. Wherein the APKs labelled as threat only start to appear in the followed threat levels (5 and 6).

The other half of the prioritised threat list (levels from 1 to 4), is composed by 34% of APKs labelled as a threat and 48% of APKs labelled as not a threat.

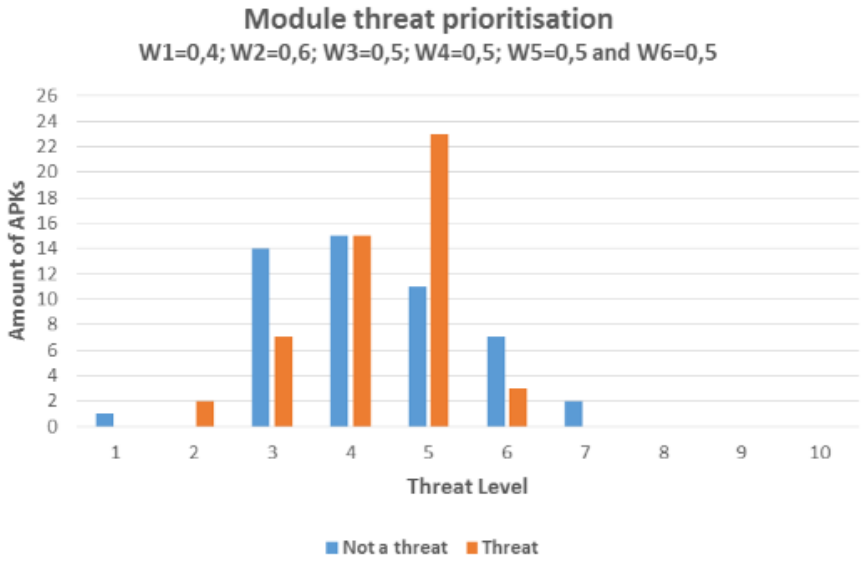


Figure 5.27: Threat prioritisation for Sample_3 *W1=0,4 and W2=0,6*; *W3=0,5 and W4=0,5*; *W5=0,5 and W6=0,5*

Figure 5.27 show us that the first half of the prioritisation of the module is composed by 52% of APKs labelled as a threat and 40% labelled as not a threat. We can also observe that in the highest threat level (level 7) we have 4% of the total amount of APKs labelled as not a threat. Wherein the APKs labelled as threat only start to appear in the followed threat levels (5 and 6).

The other half of the prioritised threat list (levels from 1 to 4), is composed by 48% of APKs labelled as a threat and 60% of APKs labelled as not a threat.

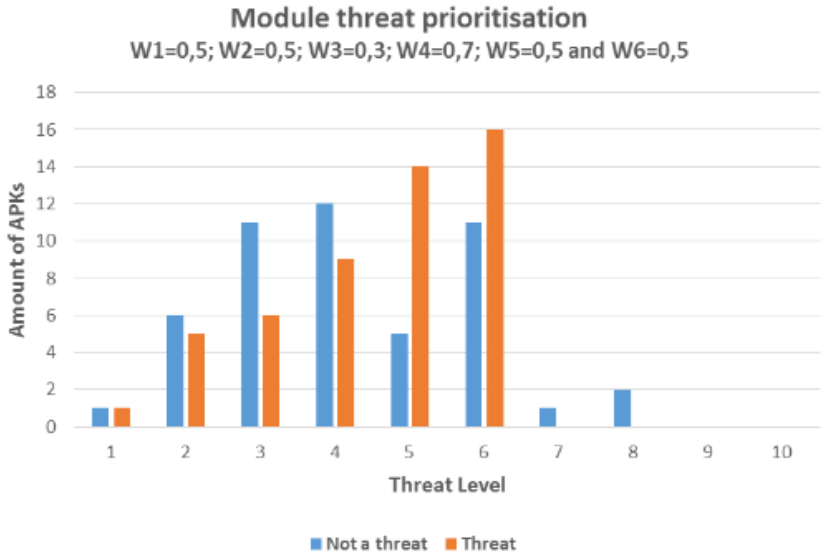


Figure 5.28: Threat prioritisation for Sample_3 *W1=0,5 and W2=0,5*; *W3=0,3 and W4=0,7*; *W5=0,5 and W6=0,5*

Figure 5.28 show us that the first half of the prioritisation of the module is composed

by 60% of APKs labelled as a threat and 38% labelled as not a threat. We can also observe that in the two highest threat levels (levels 7 and 8) we have 6% of the total amount of APKs labelled as not a threat. Wherein the APKs labelled as a threat only begging to appear in level 6, with 16 APKs labelled as a threat and 11 APKs labelled as not a threat.

The other half of the prioritised threat list (levels from 1 to 4), is composed by 40% of APKs labelled as a threat and 62% of APKs labelled as not a threat. We could also observe that the APKs labelled as a threat are distributed even in the lowest threat level.

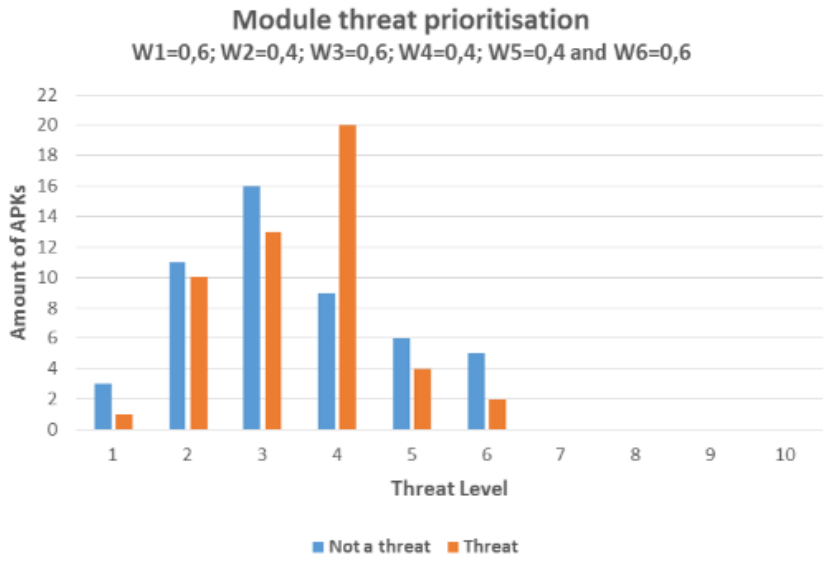


Figure 5.29: Threat prioritisation for Sample_3 *W1=0,6 and W2=0,4*; *W3=0,6 and W4=0,4*; *W5=0,4 and W6=0,6*

Looking at Figure 5.29, we observed that the first half of the prioritisation of the module (threat levels 4 to 6) is composed by 40% of APKs labelled as not a threat and 52% of APKs labelled as a threat. We can also observe that for the highest threat levels (level 5 and 6) we have 65% of APKs labelled as not a threat and 35% of APKs labelled as a threat. This means that in the highest threat levels the module gave a high score to more APKs labelled as not a threat (11 out of 17) than to APKs labelled as a threat (6 out of 17).

The other half of the prioritised threat list (levels from 1 to 3), is composed by 48% of APKs labelled as a threat and 60% of APKs labelled as not a threat. We can also see that the APKs labelled as a threat are distributed trough all the lowest threat levels; there is no clear distinction between the two types of APK.

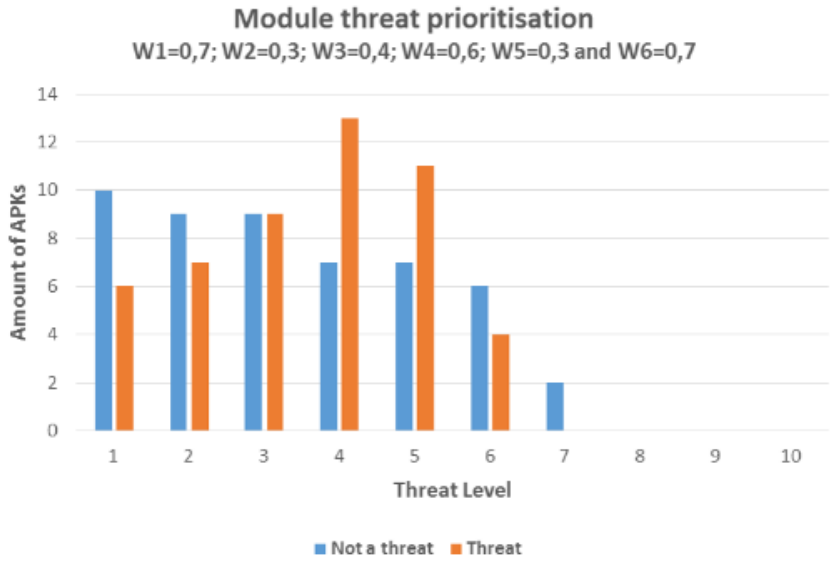


Figure 5.30: Threat prioritisation for Sample_3 *W1=0,7 and W2=0,3*; *W3=0,4 and W4=0,6*; *W5=0,3 and W6=0,7*

Figure 5.30 show us that the first half of the prioritisation of the module (threat levels 4 to 7) is composed by 44% of APKs labelled as not a threat and 56% of APKs labelled as a threat. We can also observe that for the highest threat level (level 7) we have 4% of APKs labelled as not a threat.

The other half of the prioritised threat list (levels from 1 to 3), is composed by 44% of APKs labelled as a threat and 56% of APKs labelled as not a threat. We can also see that the APKs labelled as a threat are distributed trough all the lowest threat levels.

5.5 Discussion

After an analysis of the modules accuracy results for Sample_3, we saw that for this sample our module had very bad results on assigning threat levels corresponding to the APK true value (threat or not a threat). We could even say that the solution was not able to make a distinction between the two types of APKs. Table 5.17 has a summary of the analysis of the 5 best results. There we can see the APK distribution by threat level and type (T for threat and NT for not a threat), for all the five results. By analysing the table we can see that for Figure 5.28 there are more APKs labelled as a threat in the highest threat levels (60% of the total amount of the APKs labelled as a threat), however we can not consider this as a good result since for the highest threat levels (7 and 8) there are only APKs labelled as not a threat (NT). In order to understand the nature of these results, we decided to look into our sample, to find out what went wrong in this test case.

Threat Level	Figure 5.26 T | NT	Figure 5.27 T | NT	Figure 5.28 T | NT	Figure 5.29 T | NT	Figure 5.30 T | NT
1	0 | 1	0 | 1	1 | 1	1 | 3	6 | 10
2	1 | 0	2 | 0	5 | 6	10 | 11	7 | 9
3	6 | 10	7 | 14	6 | 11	13 | 16	9 | 9
4	10 | 13	15 | 15	9 | 12	20 | 9	13 | 7
5	17 | 11	23 | 11	14 | 5	4 | 6	11 | 7
6	16 | 10	3 | 7	16 | 11	2 | 5	4 | 6
7	0 | 5	0 | 2	0 | 1	0 | 0	0 | 2
8	0 | 0	0 | 0	0 | 2	0 | 0	0 | 0
9	0 | 0	0 | 0	0 | 0	0 | 0	0 | 0
10	0 | 0	0 | 0	0 | 0	0 | 0	0 | 0

Table 5.17: Comparative analysis between the 5 best results of Sample_3 (T-Threat and NT-Not a threat)

We started by manually reviewing the APKs labelled as a threat in the sample by installing them and interacting with them. We connected our android device to a web debugging proxy application (Charles [Kar]) to visualise the data that is sent and received from the installed APK. During this process we found out that 64% of the threat sample is composed by APKs that were just a view or a button (fake); 30% were APKs that had links to advertisement (ads) and the rest 6% were APKs that were error messages (crash), e.g. the APK could not be executed because it was missing resources (see Figure 5.31).

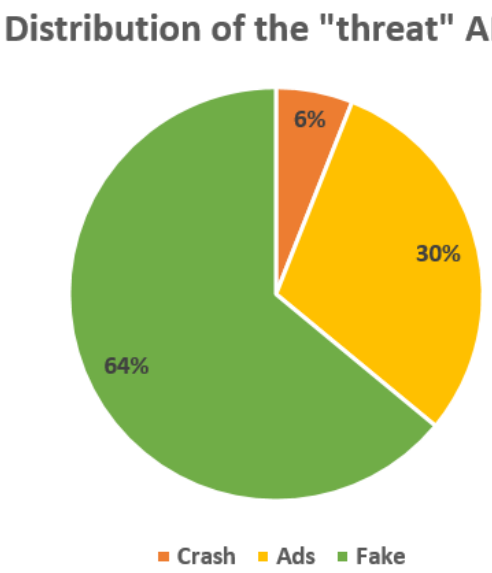


Figure 5.31: Distribution of the APKs labelled as a threat after our manual review

Since our module main goal is to find security vulnerabilities, it was understandable that in the case where an application is just a view or a button the threat level assigned to the application will be lower when compared with the labelled as not a threat APKs. In order to substantiate this conclusion we took a look at the amount of problems found in total and in average for the two types of APKs. According to Figure 5.32 for each APK labelled as "not a threat" there was in average 72,8 problems found, whereas for each APK labelled as "a threat" there was in average 52,58 problems found. The total amount of problems found for the APKs labelled as a "threat" was of 2629, while for the APKs labelled as "not a threat" was of 3640. This means that for the APKs labelled as "a threat" there was approximately less 30% problems found (which represents 1011 less problems).

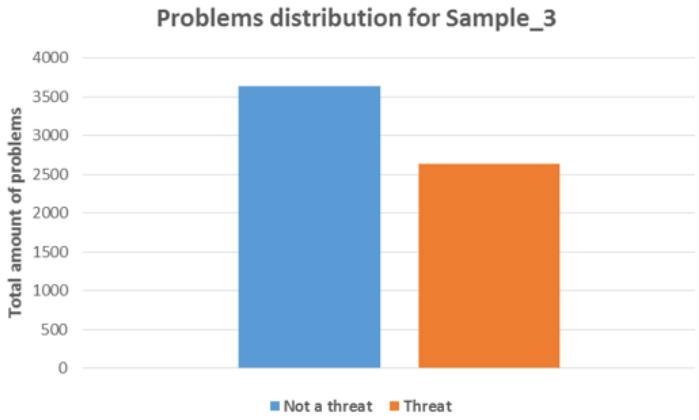


Figure 5.32: Total amount of problems found in Sample_3, distributed by "threat" and "not a threat" labels

From this setback we learned that to evaluate this module properly we should not depend on the samples reviewed by the quality assurance team. Since our threat level rating system is based on the amount of problems found, an APK that is only a button or a view will not be considered a threat, regarding the evaluation of the quality assurance team. However, we have also discovered a secondary effect of the analyse made by the module. By looking at the problems discovered by the module for each analysed APK, we found out that the module was able to identify a phenomenon called repackaging. A repackaging attack is a very common type of attacks on Android devices. In such attack, scammers do so by riding on the popularity of existing applications, embedding them with unwanted content—even malicious payloads—and masquerading them as legitimate and uploading them into app markets [Tre].

Table 5.18 represents the result of the analysis of three different APKs. If we look carefully we can see that despite all three APKs have a different md5sum (unique id), the module have identified for all of them the same problems and assigned the same severity to each problem found. After we installed each of these 3 APKs, we realised that they were the same application with only a few changes. However the goal of these applications

md5sum	Type of Problem	Tool	Gravity
223587c84e3df880935190f0b3117e73	1	MOBSF	1
	3	SUPER	0,6
	5	AndroBugs	0,5
	9	SUPER	1
	10	SUPER	0,6
d0719c7deaaa9d995922646b738ea527	1	MOBSF	1
	3	SUPER	0,6
	5	AndroBugs	0,5
	9	SUPER	1
	10	SUPER	0,6
48289c4a3f73f0cecaf5a414b2984374	1	MOBSF	1
	3	SUPER	0,6
	5	AndroBugs	0,5
	9	SUPER	1
	10	SUPER	0,6

Table 5.18: Analysis result of three different APKs

was the same, all manipulated the user to access an advertisement (see Figure 5.33).

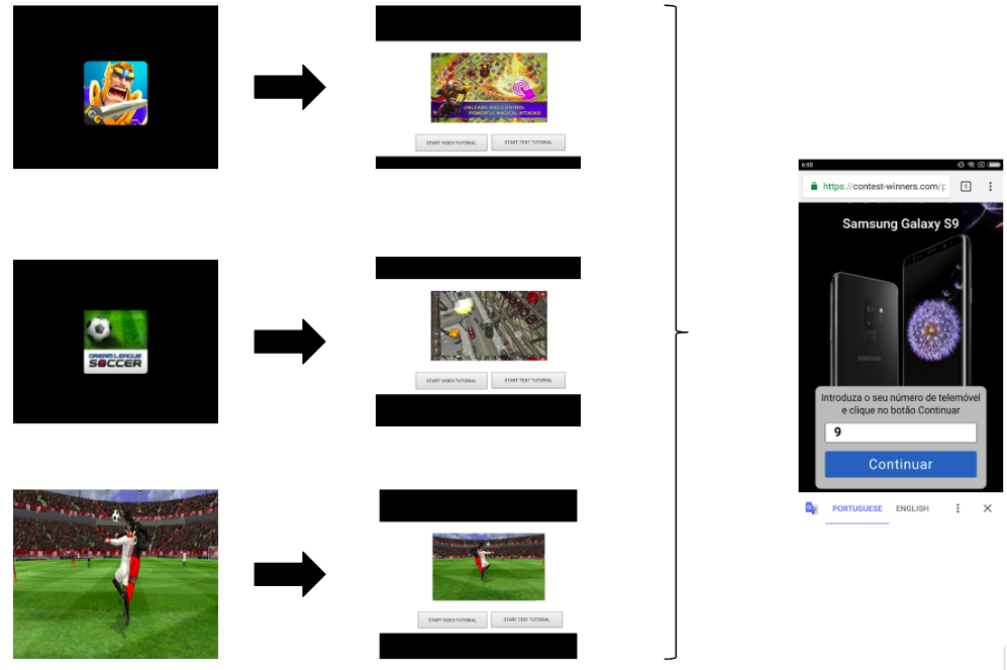


Figure 5.33: Three applications with different IDs but with the same architecture

5.5.1 Summary

In this chapter, we made several test cases in order to analyse the performance and accuracy of the model. Based on those results we made decisions that helped us adjust the architecture of the module to our goals.

We started by testing and comparing the modules execution time between a sequential and parallel arrange of the tools. We concluded that in a sequential execution of the tools, the module took 20 hours and 45 minutes to analyse 100 APKs (12 minutes and 26 seconds in average per APK); wherein a parallel execution of the tools the module took 16 hours and 47 minutes (10 minutes and 4 seconds in average per APK). We observed that the difference between the analysis time for the two scenarios, was only of 19%. So we decided to look at the analysis time distributed by each tool, in order to find the slowest tool. We found out that Qark took 64% of the total analysis time, thus we proceeded to our next test case.

In order to see if we could lower the analysis time, we compared the performance and accuracy of the module in an architecture with all tools and other without the slowest tool (Qark). We concluded that the module gave us more APKs labelled as a threat in the highest threat levels in an architecture where the slowest tool is removed, however there is a loss of its initial accuracy (gave us also more APKs labelled as not a threat in high threat levels). After contacting the developers of Qark, we were informed that we were most likely to see a lot worse performance on obfuscated code. Even if there should not be infinite loops, it would take a long time to run due to the increased amount of files and code in general. After analysing all the facts, we removed QARK from the modules integration. Despite the module timing has improved, 4 minutes is still an issue if an app store, with over 15 thousand uploads per day, wants to implement it. The enormous amount of uploaded apps would never be followed by our module, in a daily bases. So as a solution to this problem, we decided to change the process of retrieving APKs from extracting uploaded apps to extracting apps that receive negative feedback.

After the removal of Qark from the set of tools, we decided to make another test case with a new sample to understand the impact of the weight combinations on the results accuracy, so we decided to test all 125 combinations. From that analysis we had results that showed us that the module was not being able to make a proper threat level attribution to the different types of APKs. That is, to the APKs labelled as not a threat were given high threat levels and despite most of the APKs labelled as a threat having also high threat level, there was a percentage of the total amount of APKs labelled as not a threat that had low threat levels. This led us to explore the type f APKs of our sample (Sample_3).

By manually reviewing our sample, we found out that 64% of the threat sample was composed by APKs that were just a view or a button; 30% were APKs that had links to advertisement (ads) and the rest 6% were APKs that were error messages (crash), e.g. the APK could not be executed because it was missing resources. We also discovered that for

each APK labelled as “not a threat” there was in average 72,8 problems found, whereas for each APK labelled as “a threat” there was in average 52,58 problems found. Since our module main goal is to find security vulnerabilities, it was understandable that in the case where an application is just a view or a button the threat level assigned to the application will be lower when compared with the labelled as not a threat APKs. However, we also discovered a secondary effect of the analyse made by the module. By looking at the problems uncovered by the module, for each analysed APK, we found out that the module was able to identify a phenomenon called repackaging.

We concluded that the module should be evaluated with a sample reviewed not by the standards of the quality assurance team of Aptoide, but with a sample that contains APKs with the problems that the module targets.

Chapter 6

Conclusions

This dissertation had as main goal: the creation of an automated validation of applications at the app market level.

We first started by evaluating the different software verification and validation techniques. During our research we found out that this is typically done through three techniques: manual code review [Gee16], automatic analysis comprehended between dynamic [Bal99] or static [Wic+95] approaches and semi-automatic with theorem provers [Duf91]. After we weigh the advantages and disadvantages of each technique with the complexities of our own project, we chose static analysis as the selection criteria of the tools to be integrated in our system.

Before the selection of static analysis tools, we conducted a search on the different types of security vulnerabilities. According to a present study on security smells in Android [Gha+17], there are 28 symptoms in the code that signal the prospect of a security vulnerability that may lead to vulnerabilities in applications. From those 28, we chose to pursue 16.

Then we set rules to begin the process of collecting and analysing static analysis tools. We considered 8 tools but after the testing phase we only kept 5 (AndroBugs [Gita], Evicheck [Evi], MobSF [Gitc], Qark [Gitd] and SUPER [Gite]).

With the tools selected we started the implementation of an automatic tool for targeting the inspection of applications with programming malpractices (due to poor implementation/engineering choices) in the Aptoide context. The module is supported by our selected static analysis tools and work as a decision support for the quality assurance team. This support is given trough a prioritised collection of APKs, ordered by the threat it represents. The module allows the members of the quality assurance team to better manage their time by manually reviewing the applications with more security vulnerabilities. To evaluate the threat level of an APK, we created metrics based on the

hypothesis that if an APK has several severe problems reported by the static analysis tools and there is a big quantity of negative user's feedback, then it is likely that the application is harmful to the consumers.

During the testing phase of the module, we observed that QARK, in most cases, was the tool that took about 80% of the analysis time. So we decided to make a comparative analyses of the accuracy and execution time of the module when operating with all tools and when removing the slowest tool. This comparison analysis confirmed that the execution time was less when removing the slowest tool. But, other conclusions were also made: the module could inform with considerable certainty that APKs with the highest level of risk calculated by the solution were bad, but the removal of the slowest tool made the module lose some of its initial accuracy.

Although the module improves and strengthens the application validation process by uncovering problems that were not previously exposed, after we made more tests we realised that the module was not adjusted to the case where an application, considered as a threat, is just a link to a commercial or a "draft" (applications that have only a button or an image).

6.1 Contributions

The main contribution of this dissertation was the process for the definition, elaboration and evaluation of an automated validation of applications for the app stores. In the context of the Aptoide app store we modelled the automated validation as a static analysis module for the threat evaluation of Android applications. Its purpose is to effectively evaluate the threat level of an application (based on the security vulnerabilities of the application), to inform the quality assurance team on what applications are in need of a manual review. It also contributes to the enhancement of the validation capability of the Aptoide app store. The analysis system is based on 4 static analysis tools (AndroBugs [Gita], Evicheck [Evi], MobSF [Gitc] and SUPER [Gite]) and the threat rating system of an application is based on metrics developed by us. In our opinion, our model is extensible as it can be easily used to integrate with other analysis tools.

Another contribution were the metrics used for the threat rating system. These metrics helped us relate the data we obtained from the analysis of the module with a quantitative evaluation of the submitted applications.

6.2 Limitations

During the implementation of the solution, we faced two main challenges: the performance and the code obfuscation. According to the tests, in average our module could analyse one APK in 4 minutes. Even if its timing has improved, 4 minutes is still an issue if an app store, with over 15 thousand uploads per day, wants to implement it. The enormous amount of uploaded apps would never be followed by our module, in a daily

bases. In addition, since the state of the art is not yet capable of efficiently analysing apps that have obfuscated code, the performance is also dependent on the code obfuscation.

We could have multiple machines running the proposed solution, but that would just be a trade off between cost and benefit. So as a solution to this problem, we proposed that the module should only be used for apps that receive negative feedback. This way it could have a better chance to give a daily output.

The validation and accuracy of the proposed solution, had also some challenges. We evaluated the modules accuracy based on samples of applications that had been previously manually analysed by the quality assurance team. To the applications that had failed the manual analysis we labelled them as a threat and to the ones that passed the manual analysis test we labelled them as not a threat. However, during one of our tests we observed that the modules accuracy was very low and when we looked into the content of the sample we observed that the sample had a majority of "draft" applications labelled as a threat. These "draft" applications, are applications that have only a button or an image; something that does not appear to be finished and is very simplistic. So when our module analyses one of these applications, it does not find as many security vulnerabilities as it should find for an application labelled as a threat. Thus, it will believe that the application is "good". The lack of a more precise user feedback system, also posed a limitation in the process of assigning a threat level to an application.

6.3 Future Work

Future developments should include improvements to threat indication reliability and comprehensibility, experimenting our methodology with threat evaluation algorithms, and presenting a customised report of threat indicators on the basis of the specific security vulnerabilities characteristics. This way the member of the quality assurance team that starts the evaluation of a high threat level application, will know what to look for in the manual inspection.

However, the first step is to improve the accuracy of the module. There are two possible solutions: the first one is to provide a finer-grained threat category range, generating threat categories with more evident security vulnerabilities. The second solution is to integrate with more static analysis tools, capable of uncovering other security vulnerabilities. It should also be considered the introduction of a "commercial"detection module.

Another relevant development could be the indication of the threat level of an application to enhance end-user security awareness. The app store could show the app's threat level along with its recommendation value. Or could also have available a customised threat report, to show all the information about the security vulnerabilities of the application. Regarding the user feedback, the current system does not give precise information on the problems found by the end-user; so another suggestion, as future work, would be to improve the sensitivity of user feedback.

In addition, the app store could enhance the security awareness of the developer by giving a detailed report about the security vulnerabilities of the Android app package uploaded. In fact, the app store could even make a set of rules based on the problems found by the module to prevent the upload of applications with a great amount of security problems.

Bibliography

[Iso] *Accuracy (trueness and precision) of measurement methods and results — Part 1: General principles and definitions*. Standard. International Organization for Standardization, Dec. 1994.

[BC14] R. Balebako and L. Cranor. "Improving App Privacy: Nudging App Developers to Protect User Privacy." In: *IEEE Security Privacy* 12.4 (July 2014), pp. 55–58. ISSN: 1540-7993. DOI: 10.1109/MSP.2014.70.

[Bal99] T. Ball. "The Concept of Dynamic Analysis." In: *SIGSOFT Softw. Eng. Notes* 24.6 (Oct. 1999), pp. 216–234. ISSN: 0163-5948. DOI: 10.1145/318774.318944. URL: http://doi.acm.org/10.1145/318774.318944.

[Cas] Castsoftware. *Automated Code Review: Improving Security, Performance, and Maintainability of Your Software*. http://www.castsoftware.com/glossary/automated-code-review. Accessed at 25 Jan 2018.

[Chi+11] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. "Analyzing Inter-application Communication in Android." In: *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 239–252. ISBN: 978-1-4503-0643-0. DOI: 10.1145/1999995.2000018. URL: http://doi.acm.org/10.1145/1999995.2000018.

[Cim] Cimpanu, Catalin. *AndroBugs Framework Is an Android Vulnerability Analysis System. 2015*. https://news.softpedia.com/news/androbugs-framework-is-an-android-vulnerability-analysis-system-496473.shtml. Accessed 23 Feb 2018.

[Col+11] C. Collins, M. Galpin, and M. Kaeppler. *Android in Practice*. 1st. Greenwich, CT, USA: Manning Publications Co., 2011. ISBN: 1935182927, 9781935182924.

[Com] Comtact. *Facing the mobile security threat - Mobile malware statistics 2017*. http://www.comtact.co.uk/blog/facing-up-to-the-mobile-security-threat-mobile-malware-statistics-2017. Accessed at 15 Jan 2018.

[CVE] CVE Details. *Current CVSS Score Distribution For All Vulnerabilities*. https://www.cvedetails.com/. Accessed at 30 Aug 2018.

[Cyb] CyberPunk: Open Source CyberSecurity. *Mobile Security Framework: MobSF*. https://n0where.net/mobile-security-framework-mobsf. Accessed 23 Feb 2018.

[Das] Das, Kushal. *Introduction to Flask*. Accessed at 11 Sept 2018. URL: https://pymbook.readthedocs.io/en/latest/flask.html.

[Dav+11] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. "Privilege Escalation Attacks on Android." In: *Proceedings of the 13th International Conference on Information Security*. ISC'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 346–360. ISBN: 978-3-642-18177-1. URL: http://dl.acm.org/citation.cfm?id=1949317.1949356.

[Deb] Debize, Thomas. *AndroWarn : Yet Another Static Code Analyzer for malicious Android applications*. https://github.com/maaaaz/androwarn/. Accessed at 15 Feb 2018.

[Don+18] S. Dong, M. Li, W. Diao, X. Liu, J. Liu, Z. Li, F. Xu, K. Chen, X. Wang, and K. Zhang. "Understanding Android Obfuscation Techniques: A Large-Scale Investigation in the Wild." In: *CoRR* abs/1801.01633 (2018). arXiv: 1801.01633. URL: http://arxiv.org/abs/1801.01633.

[Duf91] D. A. Duffy. *Principles of Automated Theorem Proving*. New York, NY, USA: John Wiley & Sons, Inc., 1991. ISBN: 0-471-92784-8.

[Egu] Eguia, Iban. *SUPER Android Analyzer*. Accessed 27 Feb 2018. URL: https://github.com/SUPERAndroidAnalyzer/super.

[Evi] Evicheck. *Evicheck*. http://groups.inf.ed.ac.uk/security/appguarden/tools/EviCheck/. Accessed at 25 Jan 2018.

[Fal+15] L. Falsina, Y. Fratantonio, S. Zanero, C. Kruegel, G. Vigna, and F. Maggi. "Grab 'N Run: Secure and Practical Dynamic Code Loading for Android Applications." In: *Proceedings of the 31st Annual Computer Security Applications Conference*. ACSAC 2015. New York, NY, USA: ACM, 2015, pp. 201–210. ISBN: 978-1-4503-3682-6. DOI: 10.1145/2818000.2818042. URL: http://doi.acm.org/10.1145/2818000.2818042.

[Gee16] T. Gee. *What to Look for in a CodeReview*. JetBrains Technical Series, 2016.

[Gha+17] M. Ghafari, P. Gadient, and O. Nierstrasz. "Security Smells in Android." In: *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. Sept. 2017, pp. 121–130. DOI: 10.1109/SCAM.2017.24.

[Gita] GitHub. *AndroBugs*. https://github.com/AndroBugs/AndroBugs_Framework. Accessed at 25 Jan 2018.

[Gitb] GitHub. *Argus-SAF*. http://pag.arguslab.org/argus-saf. Accessed at 25 Jan 2018.

[Gitc] GitHub. *Mobile Security Framework*. https://github.com/ajinabraham/Mobile-Security-Framework. Accessed at 25 Jan 2018.

[Gitd] GitHub. *Qark*. https://github.com/linkedin/qark. Accessed at 25 Jan 2018.

[Gite] GitHub. *SUPERAndroidAnalyzer*. Accessed at 25 Jan 2018. URL: https://github.com/SUPERAndroidAnalyzer/super.

[Hwa+15] S. Hwang, S. Lee, Y. Kim, and S. Ryu. "Bittersweet ADB: Attacks and Defenses." In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS '15. New York, NY, USA: ACM, 2015, pp. 579–584. ISBN: 978-1-4503-3245-3. DOI: 10.1145/2714576.2714638. URL: http://doi.acm.org/10.1145/2714576.2714638.

[Ico] Icons 8. Accessed at 23 Sept 2018. URL: https://icons8.com/.

[Inf] InfoSecurity Europe. *Info Security Europe*. Accessed at 15 Jan 2018. URL: https://www.infosecurityeurope.com/__novadocuments/421249?v=636494122565800000.

[Int] Intel. *Dynamic Analysis vs. Static Analysis*. https://software.intel.com/en-us/inspector-user-guide-windows-dynamic-analysis-vs-static-analysis. Accessed at 7 Sept 2018.

[Ish+17] Y. Ishii, T. Watanabe, F. Kanei, Y. Takata, E. Shioji, M. Akiyama, T. Yagi, B. Sun, and T. Mori. "Understanding the Security Management of Global Third-party Android Marketplaces." In: *Proceedings of the 2Nd ACM SIGSOFT International Workshop on App Market Analytics*. WAMA 2017. Paderborn, Germany: ACM, 2017, pp. 12–18. ISBN: 978-1-4503-5158-4. DOI: 10.1145/3121264.3121267. URL: http://doi.acm.org/10.1145/3121264.3121267.

[Jet] Jet Brains. *Pycharm*. Accessed at 11 Sept 2018. URL: https://www.jetbrains.com/pycharm/.

[Jin+14] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri. "Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation." In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. New York, NY, USA: ACM, 2014, pp. 66–77. ISBN: 978-1-4503-2957-6. DOI: 10.1145/2660267.2660275. URL: http://doi.acm.org/10.1145/2660267.2660275.

[JC15] B. H. Jones and A. G. Chin. "On the efficacy of smartphone security: A critical analysis of modifications in business students' practices over time." In: *International Journal of Information Management* 35.5 (2015), pp. 561 –571. ISSN: 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2015.06.003. URL: http://www.sciencedirect.com/science/article/pii/S0268401215000596.

[Kar] Karl Von Randow. *Charles Proxy*. Accessed at 11 Sept 2018. URL: https://www.charlesproxy.com/overview/.

[KM12] S. Komatineni and D. MacLean. *Pro Android 4*. 1st. Berkely, CA, USA: Apress, 2012. ISBN: 1430239301, 9781430239307.

[Lei] Leifer, Charles. *Peewee*. Accessed at 11 Sept 2018. URL: http://docs.peewee-orm.com/en/latest/.

[Li+16] L. Li, T. F. Bissyandé, D. Octeau, and J. Klein. "DroidRA: Taming Reflection to Support Whole-program Analysis of Android Apps." In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ISSTA 2016. New York, NY, USA: ACM, 2016, pp. 318–329. ISBN: 978-1-4503-4390-9. DOI: 10.1145/2931037.2931044. URL: http://doi.acm.org/10.1145/2931037.2931044.

[Li+17] L. Li, T. F. Bissyand, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon. "Static Analysis of Android Apps: A Systematic Literature Review." In: *Inf. Softw. Technol.* 88.C (Aug. 2017), pp. 67–95. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2017.04.001. URL: https://doi.org/10.1016/j.infsof.2017.04.001.

[Lin+14] C. chi Lin, H. Li, X. Zhou, and X. Wang. "Screenmilker: How to milk your android screen for secrets." In: *In NDSS*. 2014.

[Mai] Mailgun Technologies. *The Email Service For Developers*. Accessed at 11 Sept 2018. URL: https://www.mailgun.com/.

[McA] McAfee. *Trojans, Ghosts, and More Mean Bumps Ahead for Mobile and Connected Things*. https://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2017.pdf. Accessed at 15 Jan 2018.

[ME10] P. McDaniel and W. Enck. "Not So Great Expectations: Why Application Markets Haven't Failed Security." In: *IEEE Security Privacy* 8.5 (Sept. 2010), pp. 76–78. ISSN: 1540-7993. DOI: 10.1109/MSP.2010.159.

[Mei08] R. Meier. *Professional Android Application Development*. Birmingham, UK, UK: Wrox Press Ltd., 2008. ISBN: 0470344717, 9780470344712.

[Staa] *Mobile App Usage - Statistics & Facts*. https://www.statista.com/topics/1002/mobile-app-usage. Accessed at 15 Jan 2018.

[MWR] MWR Labs. *Debuggable Apps in Android Market*. Accessed at 5 Sept 2018. URL: https://labs.mwrinfosecurity.com/blog/debuggable-apps-in-android-market.

[Nar+16] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani. "Evaluation of machine learning classifiers for mobile malware detection." In: *Soft Computing* 20.1 (Jan. 2016), pp. 343–357. ISSN: 1433-7479. DOI: 10.1007/s00500-014-1511-6. URL: https://doi.org/10.1007/s00500-014-1511-6.

[OC15] L. Onwuzurike and E. D. Cristofaro. "Danger is My Middle Name: Experimenting with SSL Vulnerabilities in Android Apps." In: *CoRR* abs/1505.00589 (2015). arXiv: 1505.00589. URL: http://arxiv.org/abs/1505.00589.

[OWAa] OWASP. *Code Review Introduction*. https://www.owasp.org/index.php/Code_Review_Introduction. Accessed at 19 Jan 2018.

[OWAb] OWASP. *Source Code Analysis Tools*. https://www.owasp.org/index.php/Source_Code_Analysis_Tools. Accessed at 15 Feb 2018.

[Pet] Peters, Sara. *AndroBugs: A Framework For Android Vulnerability Scanning. 2015.* https://www.darkreading.com/vulnerabilities---threats/androbugs-a-framework-for-android-vulnerability-scanning/d/d-id/1323000. Accessed 23 Feb 2018.

[Pir] Pires, Diogo. *Malware: Gotta catch 'em all!* Accessed at 25 Jan 2018. URL: https://blog.aptoide.com///evolution-of-aptoide-malware-detection-system/.

[Plaa] Play Store. *Always On AMOLED*. https://play.google.com/store/apps/details?id=com.tomer.alwayson. Accessed at 29 Jan 2018.

[Plab] Play Store. *Clash-Royale*. https://play.google.com/store/apps/details?id=com.supercell.clashroyale&hl=en. Accessed at 29 Jan 2018.

[Plac] Play Store. *Drink Water*. https://play.google.com/store/apps/details?id=com.aplicativoslegais.beberagua. Accessed at 29 Jan 2018.

[Plad] Play Store. *Flashlight*. https://play.google.com/store/apps/details?id=biart.com.flashlight&hl=en. Accessed at 29 Jan 2018.

[Plae] Play Store. *Freelectics Bodyweight*. https://play.google.com/store/apps/details?id=com.freeletics.lite. Accessed at 29 Jan 2018.

[Plaf] Play Store. *LOCKit*. https://play.google.com/store/apps/details?id=com.ushareit.lockit. Accessed at 29 Jan 2018.

[Plag] Play Store. *MB Way*. https://play.google.com/store/apps/details?id=pt.sibs.android.mbway&hl=pt_PT. Accessed at 29 Jan 2018.

[Plah] Play Store. *Millionaire 2018*. https://play.google.com/store/apps/details?id=com.submarineapps.umillionaire&hl=en. Accessed at 29 Jan 2018.

[Plai] Play Store. *My Passwords*. https://play.google.com/store/apps/details?id=com.er.mo.apps.mypasswords. Accessed at 29 Jan 2018.

[Plaj] Play Store. *Wallet*. https://play.google.com/store/apps/details?id=com.droid4you.application.wallet. Accessed at 29 Jan 2018.

[PVS] PVS-Studio Analyzer. *Static analysis (static code analysis)*. https://www.viva64.com/en/t/0046/. Accessed at 25 Jan 2018.

[RM18] V.-P. Ranganath and J. Mitra. "Are Free Android App Security Analysis Tools Effective in Detecting Known Vulnerabilities?" In: *ArXiv e-prints* (June 2018).

[Rou] Rouse, Margaret. *object-relational mapping (ORM)*. Accessed at 11 Sept 2018. URL: https://searchwindevelopment.techtarget.com/definition/object-relational-mapping.

[SPV14] A. B.C. K. S. Poeplau Y. Fratantonio and G. Vigna. "Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications." In: *NDSS*. 2014.

[Sch+16] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, and E. Weippl. "Protecting Software Through Obfuscation: Can It Keep Pace with Progress in Code Analysis?" In: *ACM Comput. Surv.* 49.1 (Apr. 2016), 4:1–4:37. ISSN: 0360-0300. DOI: 10.1145/2886012. URL: http://doi.acm.org/10.1145/2886012.

[Sci] Scientific American. *Godel's theorem*. https://www.scientificamerican.com/article/what-is-godels-theorem/. Accessed at 15 Feb 2018.

[Seg] Seghir, Mohamed Nassim. *EviCheck: Digital Evidence for Android*. http://groups.inf.ed.ac.uk/security/appguarden/tools/EviCheck/. Accessed 23 Feb 2018.

[Sma] SmartBear. *What is code review*. https://smartbear.com/learn/code-review/what-is-code-review/. Accessed at 25 Jan 2018.

[Staa] StackOverflow. *Developer Survey Results 2017*. Accessed at 30 Aug 2018. URL: https://insights.stackoverflow.com/survey/2017\#overview.

[Stab] Statista. *Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018*. Accessed at 30 Aug 2018. URL: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

[Stab] Statista. *Number of mobile app downloads worldwide in 2017, 2018 and 2022 (in billions)*. https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. Accessed at 15 Jan 2018.

[TM16] V. F. Taylor and I. Martinovic. "SecuRank: Starving Permission-Hungry Apps Using Contextual Permission Analysis." In: *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM '16. New York, NY, USA: ACM, 2016, pp. 43–52. ISBN: 978-1-4503-4564-4. DOI: 10.1145/2994459.2994474. URL: http://doi.acm.org/10.1145/2994459.2994474.

[TM17] V. F. Taylor and I. Martinovic. "To Update or Not to Update: Insights From a Two-Year Study of Android App Evolution." In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 45–57. ISBN: 978-1-4503-4944-4. DOI: 10.1145/3052973.3052990. URL: http://doi.acm.org/10.1145/3052973.3052990.

[Teca] Tech of Thrones. *Aptoide*. Accessed at 23 Sept 2018. URL: http://www.techofthrones.com/wp-content/uploads/2018/04/Aptoide.jpg.

[Tecb] Tech Target. *Static analysis (static code analysis)*. Accessed at 25 Jan 2018. URL: http://searchwindevelopment.techtarget.com/definition/static-analysis/.

[Tre] Trend Micro. *A Case of Misplaced Trust: How a Third-Party App Store Abuses Apple's Developer Enterprise Program to Serve Adware*. Accessed at 11 Sept 2018. URL: https://blog.trendmicro.com/trendlabs-security-intelligence/how-a-third-party-app-store-abuses-apples-developer-enterprise-program-to-serve-adware/.

[Tru] Trummer, Tony. *Introducing QARK. 2015*. https://security.linkedin.com/posts/2015/introducing-qark. Accessed 27 Feb 2018.

[Wat+17] T. Watanabe, M. Akiyama, F. Kanei, E. Shioji, Y. Takata, B. Sun, Y. Ishi, T. Shibahara, T. Yagi, and T. Mori. "Understanding the Origins of Mobile App Vulnerabilities: A Large-scale Measurement Study of Free and Paid Apps." In: *Proceedings of the 14th International Conference on Mining Software Repositories*. MSR '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 14–24. ISBN: 978-1-5386-1544-7. DOI: 10.1109/MSR.2017.23. URL: https://doi.org/10.1109/MSR.2017.23.

[Wic+95] B. A. Wichmann, A. A. Canning, D. L. Clutterbuck, L. A. Winsborrow, N. J. Ward, and D. W. R. Marsh. "Industrial perspective on static analysis." In: *Software Engineering Journal* 10.2 (Mar. 1995), pp. 69–75. ISSN: 0268-6961. DOI: 10.1049/sej.1995.0010.

[Wika] Wikipedia. *Rice's theorem*. https://en.wikipedia.org/wiki/Rice%27s_theorem. Accessed at 15 Feb 2018.

[Wikb] Wikipedia. *Turing completeness*. https://en.wikipedia.org/wiki/Turing_completeness. Accessed at 15 Feb 2018.

[Wu+16] D. Wu, D. Gao, Y. Li, and R. H. Deng. "SecComp: Towards Practically Defending Against Component Hijacking in Android Applications." In: *ArXiv e-prints* (Sept. 2016). arXiv: 1609.03322 [cs.CR].

[Wu+13] L. Wu, M. Grace, Y. Zhou, C. Wu, and J. X. "The Impact of Vendor Customizations on Android Security." In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. New York, NY, USA: ACM, 2013, pp. 623–634. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516728. URL: http://doi.acm.org/10.1145/2508859.2516728.

[Xie+11] J. Xie, H. R. Lipford, and B. Chu. "Why do programmers make security errors?" In: *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Sept. 2011, pp. 161–164. DOI: 10.1109/VLHCC.2011.6070393.

[Xu+13] L. Xu, S. F. Wu, and H. Chen. "Techniques and tools for analyzing and understanding android applications." In: *In Dissertation*. 2013.

[Xu+16] M. Xu, C. Song, Y. Ji, M.-W. Shih, K. Lu, C. Zheng, R. Duan, Y. Jang, B. Lee, C. Qian, S. Lee, and T. Kim. "Toward Engineering a Secure Android Ecosystem: A Survey of Existing Techniques." In: *ACM Comput. Surv.* 49.2 (Aug. 2016), 38:1–38:47. ISSN: 0360-0300. DOI: 10.1145/2963145. URL: http://doi.acm.org/10.1145/2963145.

Appendix A

Results of AndroBugs

A.1 Tested APKs

A.1.1 Banking

MB Way

Issue	Severity
This app has code getting the 64-bit number "Settings.Secure.ANDROID_ID". ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID. If you want to get an unique id for the device, we suggest you use "Installation" framework in the following article.	Warning
Please make sure this app has the conditions to check the validation of SSL Certificate. If it's not properly checked, it MAY allows self-signed, expired or mismatch CN certificates for SSL connection. This is a critical vulnerability and allows attackers to do MITM attacks without your knowledge. If you are transmitting users' username or password, these sensitive information may be leaking.	Warning

Table A.1: Result of AndroBugs for MB Way

Issue	Severity
ADB Backup is ENABLED for this app (default: ENABLED). ADB Backup is a good tool for backing up all of your files. If it's open for this app, people who have your phone can copy all of the sensitive data for this app in your phone (Prerequisite: 1.Unlock phone's screen 2.Open the developer mode). The sensitive data may include lifetime access token, username or password, etc.	Notice
This app is using Android SQLite databases. Prior to Android 4.0, Android has SQLite Journal Information Disclosure Vulnerability. But it can only be solved by users upgrading to Android > 4.0 and YOU CANNOT SOLVE IT BY YOURSELF	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices.Please make sure do not use "file.delete()" to delete essential files.	Notice
Found codes for checking "ApplicationInfo.FLAG_DEBUGGABLE" in AndroidManifest.xml	Notice

Table A.2: Result of AndroBugs for MB Way

Wallet

Issue	Severity
The Keystores below seem using ”byte array” or ”hard-coded cert info” to do SSL pinning (Total: 1). Please manually check.	Critical
To ensure your app is secure, always use an explicit intent when starting a Service and DO NOT declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you cannot be certain what service will respond to the intent, and the user cannot see which service starts.	Critical
This app allows Self-defined HOSTNAME VERIFIER to accept all Common Names(CN). This is a critical vulnerability and allows attackers to do MITM attacks with his valid certificate without your knowledge.	Critical
URLs that are NOT under SSL (Total:8).	Critical
This app DOES NOT check the validation of SSL Certificate. It allows self-signed, expired or mismatch CN certificates for SSL connection. This is a critical vulnerability and allows attackers to do MITM attacks without your knowledge. If you are transmitting users' username or password, these sensitive information may be leaking.	Critical
Found a critical WebView ”addJavascriptInterface” vulnerability. This method can be used to allow JavaScript to control the host application. This is a powerful feature, but also presents a security risk for applications targeted to API level JELLY_BEAN(4.2) or below, because JavaScript could use reflection to access an injected object's public fields. Use of this method in a WebView containing untrusted content could allow an attacker to manipulate the host application in unintended ways, executing Java code with the permissions of the host application.	Critical

Table A.3: Result of AndroBugs for Wallet

Issue	Severity
External storage access found (Remember DO NOT write important files to external storage)	Warning
Found ”exported” components(except for Launcher) for receiving outside applications’ actions (AndroidManifest.xml). These components can be initialised by other apps. You should add or modify the attribute to [exported=”false”] if you don’t want to. You can also protect it with a customised permission with ”signature” or higher protectionLevel and specify in ”android:permission” attribute.	Warning
This app has code getting the 64-bit number ”Settings.Secure.ANDROID_ID”. ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID.	Warning
Found ”setAllowFileAccess(true)” or not set(enabled by default) in Web-view. The attackers could inject malicious script into Web-view and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning
Found ”setJavaScriptEnabled(true)” in WebView, which could exposed to potential XSS attacks. Please check the web page code carefully and sanitise the output.	Warning

Table A.4: Result of AndroBugs for Wallet

Issue	Severity
This app is using Android SQLite databases but it's "NOT" suffering from SQLite Journal Information Disclosure Vulnerability.	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices. Please make sure do not use "file.delete()" to delete essential files.	Notice
Found using the symmetric key(PRAGMA key) to encrypt the SQLite databases.	Notice
Found codes for checking "ApplicationInfo.FLAG_DEBUGGABLE" in AndroidManifest.xml	Notice
This app is using the Internet via HTTP protocol.	Info

Table A.5: Result of AndroBugs for Wallet

A.1.2 In-app Purchases

Drink Water

Issue	Severity
Found Base64 encoding "String(s)" (Total: 3). We cannot guarantee all of the Strings are Base64 encoding and also we will not show you the decoded binary file:	Critical
SSL Connection Checking: URLs that are NOT under SSL (Total:7)	Critical

Table A.6: Result of AndroBugs for Drink Water

Issue	Severity
Dynamic code loading(DexClassLoader) found	Warning
This app has code getting the ”device id(IMEI)” but there are problems with this ”TelephonyManager.getDeviceId()” approach.	Warning
This app has code getting the 64-bit number ”Settings.Secure.ANDROID_ID”. ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID.	Warning
Found ”setAllowFileAccess(true)” or not set(enabled by default) in Web-view. The attackers could inject malicious script into Web-view and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning
Found ”setJavaScriptEnabled(true)” in WebView, which could exposed to potential XSS attacks. Please check the web page code carefully and sanitize the output.	Warning

Table A.7: Result of AndroBugs for Drink Water

Issue	Severity
ADB Backup is ENABLED for this app (default: ENABLED). ADB Backup is a good tool for backing up all of your files. If it's open for this app, people who have your phone can copy all of the sensitive data for this app in your phone (Prerequisite: 1.Unlock phone's screen 2.Open the developer mode). The sensitive data may include lifetime access token, username or password, etc.	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices. Please make sure do not use "file.delete()" to delete essential files.	Notice
This app is using Android SQLite databases but it's "NOT" suffering from SQLite Journal Information Disclosure Vulnerability.	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices. Please make sure do not use "file.delete()" to delete essential files.	Notice
This app is using the Internet via HTTP protocol.	Info

Table A.8: Result of AndroBugs for Drink Water

Freelectics Bodyweight

Issue	Severity
Security issues ”MODE_WORLD_READABLE” or ”MODE_WORLD_WRITEABLE” found.	Critical
To ensure your app is secure, always use an explicit intent when starting a Service and DO NOT declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you cannot be certain what service will respond to the intent, and the user cannot see which service starts.	Critical
URLs that are NOT under SSL (Total:62).	Critical
Found a critical WebView ”addJavascriptInterface” vulnerability. This method can be used to allow JavaScript to control the host application. This is a powerful feature, but also presents a security risk for applications targeted to API level JELLY_BEAN(4.2) or below, because JavaScript could use reflection to access an injected object’s public fields. Use of this method in a Web-view containing untrusted content could allow an attacker to manipulate the host application in unintended ways, executing Java code with the permissions of the host application.	Critical

Table A.9: Result of AndroBugs for Freelectics Bodyweight

Issue	Severity
External storage access found (Remember DO NOT write important files to external storages)	Warning
Found ”exported” components(except for Launcher) for receiving outside applications’ actions (AndroidManifest.xml). These components can be initilized by other apps. You should add or modify the attribute to [exported=”false”] if you don’t want to. You can also protect it with a customized permission with ”signature” or higher protectionLevel and specify in ”android:permission” attribute.	Warning
This app has code getting the 64-bit number ”Settings.Secure.ANDROID_ID”. ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID. If you want to get an unique id for the device, we suggest you use ”Installation” framework in the following article.	Warning
Found ”setAllowFileAccess(true)” or not set(enabled by default) in WebView. The attackers could inject malicious script into WebView and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning
Found ”setJavaScriptEnabled(true)” in WebView, which could exposed to potential XSS attacks. Please check the web page code.	Warning

Table A.10: Result of AndroBugs for Freelectics Bodyweight

Issue	Severity
Everything you delete may be recovered by any user or attacker, especially rooted devices. Please make sure do not use ”file.delete()” to delete essential files.	Notice
This app is using Android SQLite databases but it’s ”NOT” suffering from SQLite Journal Information Disclosure Vulnerability.	Notice
This app is using the Internet via HTTP protocol.	Info

Table A.11: Result of AndroBugs for Freelectics Bodyweight

A.1.3 Games

Clash-Royale

Issue	Severity
App Sandbox Permission Checking: Security issues ”MODE_WORLD_READABLE” or ”MODE_WORLD_WRITEABLE” found	Critical
SSL Connection Checking: URLs that are NOT under SSL (Total:3)	Critical
WebView RCE Vulnerability Checking: Found a critical WebView ”addJavascriptInterface” vulnerability. This method can be used to allow JavaScript to control the host application. This is a powerful feature, but also presents a security risk for applications targeted to API level JELLY_BEAN(4.2) or below, because JavaScript could use reflection to access an injected object’s public fields. Use of this method in a Web-view containing untrusted content could allow an attacker to manipulate the host application in unintended ways, executing Java code with the permissions of the host application.	Critical

Table A.12: Result of AndroBugs for Clash-Royale

Issue	Severity
External storage access found (Remember DO NOT write important files to external storages)	Warning
This app has code getting the ”device id(IMEI)” but there are problems with this ”TelephonyManager.getDeviceId()” approach.	Warning
This app has code getting the 64-bit number ”Settings.Secure.ANDROID_ID”. ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID.	Warning
Found ”setAllowFileAccess(true)” or not set(enabled by default) in Web-view. The attackers could inject malicious script into Web-view and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning
Found ”setJavaScriptEnabled(true)” in WebView, which could exposed to potential XSS attacks. Please check the web page code carefully and sanitize the output.	Warning

Table A.13: Result of AndroBugs for Clash-Royale

Issue	Severity
ADB Backup is ENABLED for this app (default: ENABLED). ADB Backup is a good tool for backing up all of your files. If it's open for this app, people who have your phone can copy all of the sensitive data for this app in your phone (Prerequisite: 1.Unlock phone's screen 2.Open the developer mode). The sensitive data may include lifetime access token, username or password, etc.	Notice
The app may has the code checking for "root" permission, mounting file-system operations or monitoring system.	Notice
This app is using Android SQLite databases but it's "NOT" suffering from SQLite Journal Information Disclosure Vulnerability.	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices. Please make sure do not use "file.delete()" to delete essential files.	Notice
This app is using the Internet via HTTP protocol.	Info

Table A.14: Result of AndroBugs for Clash-Royale

Millionaire 2018

Issue	Severity
Security issues "MODE_WORLD_READABLE" or "MODE_WORLD_WRITEABLE" found.	Critical
Found "exported" ContentProvider, allowing any other app on the device to access it (AndroidManifest.xml). You should modify the attribute to [exported="false"] or set at least "signature" protectionalLevel permission if you don't want to.	Critical
URLs that are NOT under SSL (Total:2).	Critical
Found a critical WebView "addJavascriptInterface" vulnerability. This method can be used to allow JavaScript to control the host application. This is a powerful feature, but also presents a security risk for applications targeted to API level JELLY_BEAN(4.2) or below, because JavaScript could use reflection to access an injected object's public fields. Use of this method in a Web-view containing untrusted content could allow an attacker to manipulate the host application in unintended ways, executing Java code with the permissions of the host application.	Critical

Table A.15: Result of AndroBugs for Millionaire 2018

Issue	Severity
Dynamic code loading(DexClassLoader) found.	Warning
External storage access found (Remember DO NOT write important files to external storages)	Warning
This app has code getting the 64-bit number ”Settings.Secure.ANDROID_ID”. ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID. If you want to get an unique id for the device, we suggest you use ”Installation” framework in the following article.	Warning
Please make sure this app has the conditions to check the validation of SSL Certificate. If it's not properly checked, it MAY allows self-signed, expired or mismatch CN certificates for SSL connection. This is a critical vulnerability and allows attackers to do MITM attacks without your knowledge. If you are transmitting users' username or password, these sensitive information may be leaking.	Warning
Found ”setAllowFileAccess(true)” or not set(enabled by default) in WebView. The attackers could inject malicious script into WebView and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning
Found ”setJavaScriptEnabled(true)” in WebView, which could exposed to potential XSS attacks. Please check the web page code.	Warning

Table A.16: Result of AndroBugs for Millionaire 2018

Issue	Severity
ADB Backup is ENABLED for this app (default: ENABLED). ADB Backup is a good tool for backing up all of your files. If it's open for this app, people who have your phone can copy all of the sensitive data for this app in your phone (Prerequisite: 1.Unlock phone's screen 2.Open the developer mode). The sensitive data may include lifetime access token, username or password, etc.	Notice
This app is using Android SQLite databases but it's "NOT" suffering from SQLite Journal Information Disclosure Vulnerability.	Notice
Found codes for checking "ApplicationInfo.FLAG_DEBUGGABLE" in AndroidManifest.xml	Notice
The Keystores below are "protected" by password and seem using SSL-pinning (Total: 1).	Notice
This app is using the Internet via HTTP protocol.	Info

Table A.17: Result of AndroBugs for Millionaire 2018

A.1.4 Privacy

LOCKit

Issue	Severity
This app is using critical function 'Runtime.getRuntime().exec("...")'. Please confirm that those code sections are not harmful.	Critical
This app allows Self-defined HOSTNAME VERIFIER to accept all Common Names(CN). This is a critical vulnerability and allows attackers to do MITM attacks with his valid certificate without your knowledge.	Critical
URLs that are NOT under SSL (Total:38).	Critical
This app DOES NOT check the validation of SSL Certificate. It allows self-signed, expired or mismatch CN certificates for SSL connection. This is a critical vulnerability and allows attackers to do MITM attacks without your knowledge. If you are transmitting users' username or password, these sensitive information may be leaking.	Critical
This app has very high privileges. Use it carefully. Critical use-permission found: "android.permission.MOUNT_UNMOUNT_FILESYSTEMS".	Critical
Found a critical WebView "addJavascriptInterface" vulnerability. This method can be used to allow JavaScript to control the host application. This is a powerful feature, but also presents a security risk for applications targeted to API level JELLY_BEAN(4.2) or below, because JavaScript could use reflection to access an injected object's public fields. Use of this method in a WebView containing ntrusted content could allow an attacker to manipulate the host application in unintended ways, executing Java code with the permissions of the host application.	Critical

Table A.18: Result of AndroBugs for LOCKit

Issue	Severity
External storage access found (Remember DO NOT write important files to external storages)	Warning
Found ”exported” components(except for Launcher) for receiving outside applications’ actions (AndroidManifest.xml). These components can be initilized by other apps. You should add or modify the attribute to [exported=”false”] if you don’t want to. You can also protect it with a customized permission with ”signature” or higher protectionLevel and specify in ”android:permission” attribute.	Warning
This app has code getting the ”device id(IMEI)” but there are problems with this ”TelephonyManager.getDeviceId()” approach.	Warning
This app has code getting the 64-bit number ”Settings.Secure.ANDROID_ID”. ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID. If you want to get an unique id for the device, we suggest you use ”Installation” framework in the following article.	Warning
Found ”setAllowFileAccess(true)” or not set(enabled by default) in WebView. The attackers could inject malicious script into WebView and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning
Found ”setJavaScriptEnabled(true)” in WebView, which could exposed to potential XSS attacks. Please check the web page code.	Warning

Table A.19: Result of AndroBugs for LOCKit

Issue	Severity
ADB Backup is ENABLED for this app (default: ENABLED). ADB Backup is a good tool for backing up all of your files. If it's open for this app, people who have your phone can copy all of the sensitive data for this app in your phone (Prerequisite: 1.Unlock phone's screen 2.Open the developer mode). The sensitive data may include lifetime access token, username or password, etc.	Notice
The app may has the code checking for "root" permission, mounting filesystem operations or monitoring system.	Notice
This app is using Android SQLite databases but it's "NOT" suffering from SQLite Journal Information Disclosure Vulnerability.	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices.Please make sure do not use "file.delete()" to delete essential files.	Notice
Found codes for checking "ApplicationInfo.FLAG_DEBUGGABLE" in AndroidManifest.xml	Notice
This app is using the Internet via HTTP protocol.	info

Table A.20: Result of AndroBugs for LOCKit

My Passwords - Password Manager

Issue	Severity
'Fragment' or 'Fragment for ActionbarSherlock' has a severe vulnerability prior to Android 4.4 (API 19).	Critical
Security issues "MODE_WORLD_READABLE" or "MODE_WORLD_WRITEABLE" found.	Critical
URLs that are NOT under SSL (Total:2).	Critical
This app has some internet accessing codes but does not have 'android.permission.INTERNET' use-permission in AndroidManifest.	Critical

Table A.21: Result of AndroBugs for My Passwords

Issue	Severity
External storage access found (Remember DO NOT write important files to external storages)	Warning
Found ”setAllowFileAccess(true)” or not set(enabled by default) in WebView. The attackers could inject malicious script into WebView and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning

Table A.22: Result of AndroBugs for My Passwords

Issue	Severity
ADB Backup is ENABLED for this app (default: ENABLED). ADB Backup is a good tool for backing up all of your files. If it’s open for this app, people who have your phone can copy all of the sensitive data for this app in your phone (Prerequisite: 1.Unlock phone’s screen 2.Open the developer mode). The sensitive data may include lifetime access token, username or password, etc.	Notice
This app is using Android SQLite databases but it’s ”NOT” suffering from SQLite Journal Information Disclosure Vulnerability.	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices. Please make sure do not use ”file.delete()” to delete essential files.	Notice
This app is using the Internet via HTTP protocol.	Info

Table A.23: Result of AndroBugs for My Passwords

A.1.5 Tools

Always on AMOLED

Issue	Severity
Found Base64 encoding ”String(s)” (Total: 2). We cannot guarantee all of the Strings are Base64 encoding and also we will not show you the decoded binary file.	Critical
Security issues ”MODE_WORLD_READABLE” or ”MODE_WORLD_WRITEABLE” found.	Critical
To ensure your app is secure, always use an explicit intent when starting a Service and DO NOT declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you cannot be certain what service will respond to the intent, and the user cannot see which service starts.	Critical
URLs that are NOT under SSL (Total:6).	Critical
This app should only be released and signed by device manufacturer or Google and put under ”/system/app”. If not, it may be a malicious app.	Critical

Table A.24: Result of AndroBugs for Always on AMOLED

Issue	Severity
External storage access found (Remember DO NOT write important files to external storage)	Warning
Found ”exported” components(except for Launcher) for receiving outside applications actions (AndroidManifest.xml). These components can be initialised by other apps. You should add or modify the attribute to [exported=”false”] if you don’t want to. You can also protect it with a customised permission with ”signature” or higher protection Level and specify in ”android:permission” attribute.	Warning
This app has code getting the 64-bit number ”Settings.Secure.ANDROID_ID”. ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID. If you want to get an unique id for the device, we suggest you use ”Installation” framework in the following article.	Warning
Found ”setAllowFileAccess(true)” or not set(enabled by default) in WebView. The attackers could inject malicious script into WebView and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning

Table A.25: Result of AndroBugs for Always on AMOLED

Issue	Severity
ADB Backup is ENABLED for this app (default: ENABLED). ADB Backup is a good tool for backing up all of your files. If it's open for this app, people who have your phone can copy all of the sensitive data for this app in your phone (Prerequisite: 1.Unlock phone's screen 2.Open the developer mode). The sensitive data may include lifetime access token, username or password, etc.	Notice
The app may has the code checking for "root" permission, mounting file system operations or monitoring system.	Notice
This app is using Android SQLite databases but it's "NOT" suffering from SQLite Journal Information Disclosure Vulnerability.	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices.Please make sure do not use "file.delete()" to delete essential files.	Notice
Found codes for checking "ApplicationInfo.FLAG_DEBUGGABLE" in AndroidManifest.xml	Notice
This app is using the Internet via HTTP protocol.	info

Table A.26: Result of AndroBugs for Always on AMOLED

Flashlight

Issue	Severity
App Sandbox Permission Checking: Security issues "MODE_WORLD_READABLE" or "MODE_WORLD_WRITEABLE" found	Critical
SSL Connection Checking: URLs that are NOT under SSL (Total:2)	Critical
WebView RCE Vulnerability Checking: Found a critical WebView "addJavascriptInterface" vulnerability. This method can be used to allow JavaScript to control the host application. This is a powerful feature, but also presents a security risk for applications targeted to API level JELLY_BEAN(4.2) or below, because JavaScript could use reflection to access an injected object's public fields. Use of this method in a Web-view containing untrusted content could allow an attacker to manipulate the host application in unintended ways, executing Java code with the permissions of the host application.	Critical

Table A.27: Result of AndroBugs for Flashlight

Issue	Severity
Dynamic code loading(DexClassLoader) found	Warning
External storage access found (Remember DO NOT write important files to external storage)	Warning
This app has code getting the 64-bit number "Settings.Secure.ANDROID_ID". ANDROID_ID seems a good choice for a unique device identifier. There are downsides: First, it is not 100% reliable on releases of Android prior to 2.2 (Froyo). Also, there has been at least one widely-observed bug in a popular handset from a major manufacturer, where every instance has the same ANDROID_ID.	Warning
Found "setAllowFileAccess(true)" or not set(enabled by default) in Web-view. The attackers could inject malicious script into Web-view and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access.	Warning
Found "setJavaScriptEnabled(true)" in WebView, which could exposed to potential XSS attacks. Please check the web page code carefully and sanitise the output.	Warning

Table A.28: Result of AndroBugs for Flashlight

Issue	Severity
ADB Backup is ENABLED for this app (default: ENABLED). ADB Backup is a good tool for backing up all of your files. If it's open for this app, people who have your phone can copy all of the sensitive data for this app in your phone (Prerequisite: 1.Unlock phone's screen 2.Open the developer mode). The sensitive data may include lifetime access token, username or password, etc.	Notice
This app is using Android SQLite databases. Prior to Android 4.0, Android has SQLite Journal Information Disclosure Vulnerability. But it can only be solved by users upgrading to Android > 4.0 and YOU CANNOT SOLVE IT BY YOURSELF (But you can encrypt your databases and Journals by "SQLCipher" or other libs).	Notice
Everything you delete may be recovered by any user or attacker, especially rooted devices. Please make sure do not use "file.delete()" to delete essential files.	Notice
This app is using the Internet via HTTP protocol.	Info

Table A.29: Result of AndroBugs for Flashlight

Appendix B

Results of EviCheck

B.1 Tested APKs

B.1.1 Banking

MB Way

Permission	Meaning
INTERNET	Allows applications to open network sockets.
VIBRATE	Allows access to the vibrator.

Table B.1: Result of EviCheck for MB Way

Wallet

Permission	Meaning
ACCESS_FINE_LOCATION	Allows an app to access precise location.
CHANGE_COMPONENT_ENABLED_STATE	Allows an application to change whether an application component (other than its own) is enabled or not.
VIBRATE	Allows access to the vibrator.
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming.
READ_CONTACTS	Allows an application to read the user's contacts data.
ACCESS_NETWORK_STATE	Allows applications to access information about networks.
GET_ACCOUNTS	Allows access to the list of accounts in the Accounts Service.

Table B.2: Result of EviCheck for Wallet

B.1.2 In-app Purchases

Drink Water

Permission	Meaning
CHANGE_COMPONENT_ENABLED_STATE	Allows an application to change whether an application component (other than its own) is enabled or not.
VIBRATE	Allows access to the vibrator.
READ_CONTACTS	Allows an application to read the user's contacts data.

Table B.3: Result of EviCheck for Drink Water

Freelectics Bodyweight

Permission	Meaning
READ_CONTACTS	Allows an application to read the user's contacts data.
VIBRATE	Allows access to the vibrator.
INTERNET	Allows applications to open network sockets.
ACCESS_NETWORK_STATE	Allows applications to access information about networks.

Table B.4: Result of EviCheck for Freelectics Bodyweight

B.1.3 Games

Clash-Royale

Permission	Meaning
CHANGE_WIFI_STATE	Allows applications to change Wi-Fi connectivity state.
ACCESS_NETWORK_STATE	Allows applications to access information about networks.
READ_CONTACTS	Allows an application to read the user's contacts data.
INTERNET	Allows applications to open network sockets.
VIBRATE	Allows access to the vibrator.
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming.

Table B.5: Result of EviCheck for Clash-Royale

Millionaire 2018

Permission	Meaning
READ_PHONE_State	AAllows read only access to phone state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device.
VIBRATE	Allows access to the vibrator.
READ_CONTACTS	Allows an application to read the user's contacts data.
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming.

Table B.6: Result of EviCheck for Millionaire 2018

B.1.4 Privacy

LOCKit

Permission	Meaning
GET_TASKS	This constant was deprecated in API level 21. No longer enforced.
ACCESS_NETWORK_STATE	Allows applications to access information about networks.
DISABLE_KEYGUARD	Allows applications to disable the keyguard if it is not secure.
ACCESS_FINE_LOCATION	Allows an app to access precise location.
READ_LOGS	Allows an application to read the low-level system log files.
READ_PHONE_State	AAllows read only access to phone state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device.
VIBRATE	Allows access to the vibrator.
ACCESS_WIFI_STATE	Allows applications to access information about Wi-Fi networks.
CAMERA	Required to be able to access the camera device.
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming.
INTERNET	Allows applications to open network sockets.

Table B.7: Result of EviCheck for LOCKit

My Passwords - Password Manager

Permission	Meaning
READ_CONTACTS	Allows an application to read the user's contacts data.
VIBRATE	Allows access to the vibrator.
KILL_BACKGROUND_PROCESSES	Allows an application to call killBackgroundProcesses(String).

Table B.8: Result of EviCheck for My Passwords

B.1.5 Tools

Always on AMOLED

Permission	Meaning
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming.
ACCESS_NETWORK_STATE	Allows applications to access information about networks.
CAMERA	Required to be able to access the camera device.
READ_LOGS	Allows an application to read the low-level system log files.
INTERNET	Allows applications to open network sockets.
VIBRATE	Allows access to the vibrator.
WRITE_SETTINGS	Allows an application to read or write the system settings.

Table B.9: Result of EviCheck for Always on AMOLED

Flashlight

Permission	Meaning
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming.
CAMERA	Required to be able to access the camera device.

Table B.10: Result of EviCheck for Flashlight

Appendix C

Results of Mobile Security Framework

C.1 Tested APKs

C.1.1 Banking

MB Way

Permissions

Permission	Meaning
INTERNET	Allows applications to open network sockets.
VIBRATE	Allows access to the vibrator.
RECEIVE	Unknown permission from android reference.
ACCESS_NETWORK_STATE	View network status.
C2D_MESSAGE	Allows cloud to device messaging.
WAKE_LOCK	Prevent phone from sleeping.
WRITE_SETTINGS	modify global system settings
READ_CONTACTS	Read contact data.
GET_ACCOUNTS	Discover known accounts.

Table C.1: Result of Mobile Security Framework for MB Way

Manifest Analysis

Issue	Description
Application Data can be Backed up	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.

Table C.2: Result of Mobile Security Framework for MB Way

Code Analysis

Issue	Severity
This App uses Java Hash Code. It's a weak hash function and should never be used in Secure Crypto Implementation.	High
App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	High
The App uses an insecure Random Number Generator.	High
SHA-1 is a weak hash known to have hash collisions.	High
The App logs information. Sensitive information should never be logged.	Info
This App copies data to clipboard. Sensitive data should not be copied to clipboard as other applications can access it.	Info

Table C.3: Result of Mobile Security Framework for MB Way

Wallet

Permissions

Permission	Meaning
READ_GSERVICES	Unknown permission from android reference.
USE_CREDENTIALS	Allows an application to request authentication tokens.
ACCESS_WIFI_STATE	View Wi-Fi status.
INTERNET	Allows applications to open network sockets.
ACCESS_COARSE_LOCATION	Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.
BLUETOOTH	Allows an application to view configuration of the local Bluetooth phone and to make and accept connections with paired devices.
UNINSTALL_SHORTCUT	Unknown permission from android reference.
BILLING	Unknown permission from android reference.
ACCESS_FINE_LOCATION	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.
GET_ACCOUNTS	Discover known accounts.
INSTALL_SHORTCUT	Unknown permission from android reference.
MAPS_RECEIVE	Unknown permission from android reference.
RECEIVE	Unknown permission from android reference.
ACCESS_NETWORK_STATE	View network status.
WRITE_EXTERNAL_STORAGE	Allows an application to write to the SD card.
READ_EXTERNAL_STORAGE	Allows an application to read from SD Card.

Table C.4: Result of Mobile Security Framework for Wallet

Permission	Meaning
RECEIVE_BOOT_COMPLETED	Automatically start at boot.
VIBRATE	Allows access to the vibrator.
ACTIVITY_RECOGNITION	Unknown permission from android reference.
WAKE_LOCK	Prevent phone from sleeping.
ACTIVITY_RECOGNITION	Unknown permission from android reference
CAMERA	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
CHANGE_WIFI_STATE	Allows an application to connect to and disconnect from Wi-Fi access points and to make changes to configured Wi-Fi networks.
READ_CONTACTS	Read contact data.
READ_PROFILE	Allows an application to read the user's personal profile data.

Table C.5: Result of Mobile Security Framework for Wallet

Manifest Analysis

Issue	Description
Activity is not Protected.	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
Broadcast Receiver is not Protected	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

Table C.6: Result of Mobile Security Framework for Wallet

Issue	Description
Service is not Protected	A Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Service is explicitly exported.
Service is protected by a permission, but the protection level of the permission should be checked.	A Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Table C.7: Result of Mobile Security Framework for Wallet

Code Analysis

Issue	Severity
This App uses Java Hash Code. It's a weak hash function and should never be used in Secure Crypto Implementation.	High
App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	High
App can read/write to External Storage. Any App can read data written to External Storage.	High
Files may contain hardcoded sensitive informations like usernames, passwords, keys etc.	High

Table C.8: Result of Mobile Security Framework for Wallet

Issue	Severity
SHA-1 is a weak hash known to have hash collisions.	High
App creates temp file. Sensitive information should never be written into a temp file.	High
MD5 is a weak hash known to have hash collisions.	High
The App uses an insecure Random Number Generator.	High
Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole.	Warning
IP Address disclosure	Warning
The App logs information. Sensitive information should never be logged.	Info

Table C.9: Result of Mobile Security Framework for Wallet

C.1.2 In-app Purchases

Drink Water

Permissions

Permission	Meaning
INTERNET	Allows applications to open network sockets.
ACCESS_NETWORK_STATE	View network status.
BILLING	Unknown permission from android reference.
SET_ALARM	set alarm in alarm clock.
RECEIVE_BOOT_COMPLETED	Automatically start at boot.

Table C.10: Result of Mobile Security Framework for Drink Water

Manifest Analysis

Issue	Description
Activity is not Protected.	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
Application Data can be Backed up	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
Broadcast Receiver is not Protected	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

Table C.11: Result of Mobile Security Framework for Drink Water

Code Analysis

Issue	Severity
App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	High
The App uses an insecure Random Number Generator.	High
Files may contain hardcoded sensitive informations like usernames, passwords, keys etc.	High
The App uses ECB mode in Cryptographic encryption algorithm. ECB mode is known to be weak as it results in the same ciphertext for identical blocks of plaintext.	High

Table C.12: Result of Mobile Security Framework for Drink Water

Issue	Severity
MD5 is a weak hash known to have hash collisions.	High
The App logs information. Sensitive information should never be logged.	Info

Table C.13: Result of Mobile Security Framework for Drink Water

Freelectics Bodyweight

Permissions

Permission	Meaning
WRITE_SETTINGS	modify global system settings
READ_GSERVICES	Unknown permission from android reference.
ACCESS_COARSE_LOCATION	Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.
CHANGE_BADGE	Unknown permission from android reference.
INTERNET	Allows applications to open network sockets.
WRITE	Unknown permission from android reference.
BILLING	Unknown permission from android reference.
ACCESS_FINE_LOCATION	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.
REORDER_TASKS	Allows an application to move tasks to the foreground and background. Malicious applications can force themselves to the front without your control.
ACCESS_NETWORK_STATE	View network status.
WRITE_EXTERNAL_STORAGE	Allows an application to write to the SD card.

Table C.14: Result of Mobile Security Framework for Freelectics Bodyweight

Permission	Meaning
RECEIVE_BOOT_COMPLETED	Automatically start at boot.
C2D_MESSAGE	Allows cloud to device messaging.
GET_ACCOUNTS	Discover known accounts.
WAKE_LOCK	Prevent phone from sleeping.
ACCESS_WIFI_STATE	View Wi-Fi status.
READ_CONTACTS	Read contact data.
VIBRATE	Allows access to the vibrator.

Table C.15: Result of Mobile Security Framework for Freelectics Bodyweight

Manifest Analysis

Issue	Description
Activity is not Protected.	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
Broadcast Receiver is not Protected	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
Service is protected by a permission, but the protection level of the permission should be checked.	A Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Table C.16: Result of Mobile Security Framework for Freelectics Bodyweight

C.1.3 Games

Clash-Royale

Permissions

Permission	Meaning
VIBRATE	Allows access to the vibrator.
C2D_MESSAGE	Allows cloud to device messaging.
ACCESS_NETWORK_STATE	View network status.
RECEIVE	Unknown permission from android reference.
WAKE_LOCK	Prevent phone from sleeping.
CHANGE_WIFI_STATE	Allows an application to connect to and disconnect from Wi-Fi access points and to make changes to configured Wi-Fi networks.
INTERNET	Allows applications to open network sockets.
WRITE_EXTERNAL_STORAGE	Allows an application to write to the SD card.
BILLING	Unknown permission from android reference.

Table C.17: Result of Mobile Security Framework for Clash-Royale

Manifest Analysis

Issue	Description
Application Data can be Backed up	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
Broadcast Receiver is not Protected	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

Table C.18: Result of Mobile Security Framework for Clash-Royale

Code Analysis

Issue	Severity
This App uses Java Hash Code. It's a weak hash function and should never be used in Secure Crypto Implementation.	High
App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	High
App can read/write to External Storage. Any App can read data written to External Storage.	High
Files may contain hardcoded sensitive informations like usernames, passwords, keys etc.	High
The App logs information. Sensitive information should never be logged.	Info
The App uses ECB mode in Cryptographic encryption algorithm. ECB mode is known to be weak as it results in the same ciphertext for identical blocks of plaintext.	High
SHA-1 is a weak hash known to have hash collisions.	High
This App copies data to clipboard. Sensitive data should not be copied to clipboard as other applications can access it.	Info
The App uses an insecure Random Number Generator.	High
SHA-1 is a weak hash known to have hash collisions.	High
MD5 is a weak hash known to have hash collisions.	High
Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole.	Warning

Table C.19: Result of Mobile Security Framework for Clash-Royale

Millionaire 2018

Permissions

Permission	Meaning
INTERNET	Allows applications to open network sockets.
ACCESS_NETWORK_STATE	View network status.

Table C.20: Result of Mobile Security Framework for Millionaire 2018

Manifest Analysis

Issue	Description
Activity is not Protected.	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
Application Data can be Backed up	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
Service is protected by a permission, but the protection level of the permission should be checked.	A Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Table C.21: Result of Mobile Security Framework for Millionaire 2018

Code Analysis

Issue	Severity
This App uses Java Hash Code. It's a weak hash function and should never be used in Secure Crypto Implementation.	High
The App uses an insecure Random Number Generator.	High
Files may contain hardcoded sensitive informations like usernames, passwords, keys etc.	High
The App logs information. Sensitive information should never be logged.	Info
SHA-1 is a weak hash known to have hash collisions.	High
IP Address disclosure	Warning
This App may have root detection capabilities.	Secure
Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole.	Warning
MD5 is a weak hash known to have hash collisions.	High

Table C.22: Result of Mobile Security Framework for Millionaire 2018

C.1.4 Privacy

LOCKit

Permissions

Permission	Meaning
CHANGE_NETWORK_STATE	Allows an application to change the state of network connectivity.
DISABLE_KEYGUARD	Allows an application to disable the key lock and any associated password security. A legitimate example of this is the phone disabling the key lock when receiving an incoming phone call, then re-enabling the key lock when the call is finished.
READ_LOGS	Allows an application to read from the system's various log files. This allows it to discover general information about what you are doing with the phone, potentially including personal or private information.
CAMERA	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
INTERNET	Allows applications to open network sockets.
EXPAND_STATUS_BAR	Allows application to expand or collapse the status bar.
ACCESS_FINE_LOCATION	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.
WAKE_LOCK	Prevent phone from sleeping.
ACCESS_NETWORK_STATE	View network status.
GET_TASKS	Allows application to retrieve information about currently and recently running tasks. May allow malicious applications to discover private information about other applications.
WRITE_EXTERNAL_STORAGE	Allows an application to write to the SD card.
READ_EXTERNAL_STORAGE	Allows an application to read from SD Card.

Table C.23: Result of Mobile Security Framework for LOCKit

Permission	Meaning
RECEIVE_BOOT_COMPLETED	Automatically start at boot.
BROADCAST_STICKY	Allows an application to send sticky broadcasts, which remain after the broadcast ends. Malicious applications can make the phone slow or unstable by causing it to use too much memory.
WRITE_SETTINGS	modify global system settings
FLASHLIGHT	Allows the application to control the flashlight
READ_PHONE_STATE	read phone state and identity
VIBRATE	Allows access to the vibrator.
SYSTEM_ALERT_WINDOW	lows an application to show system-alert windows. Malicious applications can take over the entire screen of the phone.
KILL_BACKGROUND_PROCESSES	Allows an application to kill background processes of other applications, even if memory is not low.
ACCESS_WIFI_STATE	View Wi-Fi status.
PACKAGE_USAGE_STATS	Allows the modification of collected component usage statistics. Not for use by common applications.
CHANGE_WIFI_STATE	Allows an application to connect to and disconnect from Wi-Fi access points and to make changes to configured Wi-Fi networks.
MOUNT_UNMOUNT_FILESYSTEMS	Allows the application to mount and unmount file systems for removable storage.

Table C.24: Result of Mobile Security Framework for LOCKit

Manifest Analysis

Issue	Description
Application Data can be Backed up	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.

Table C.25: Result of Mobile Security Framework for LOCKit

Issue	Description
Activity is not Protected.	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
Broadcast Receiver is not Protected	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

Table C.26: Result of Mobile Security Framework for LOCKit

C.1.5 Tools

Always on AMOLED

Permissions

Permission	Meaning
DISABLE_KEYGUARD	Allows an application to disable the key lock and any associated password security. A legitimate example of this is the phone disabling the key lock when receiving an incoming phone call, then re-enabling the key lock when the call is finished.
DUMP	Allows application to retrieve internal status of the system. Malicious applications may retrieve a wide variety of private and secure information that they should never commonly need.
DEVICE_POWER	Allows the application to turn the phone on or off.
INTERNET	Allows applications to open network sockets.
CHANGE_CONFIGURATION	Allows an application to change the current configuration, such as the locale or overall font size.
PACKAGE_USAGE_STATS	Allows the modification of collected component usage statistics. Not for use by common applications.

Table C.27: Result of Mobile Security Framework for Always on AMOLED

Permission	Meaning
BILLING	Unknown permission from android reference.
RECEIVE	Unknown permission from android reference.
ACCESS_NETWORK_STATE	View network status.
GET_TASKS	Allows application to retrieve information about currently and recently running tasks. May allow malicious applications to discover private information about other applications.
RECEIVE_BOOT_COMPLETED	Automatically start at boot.
BIND_DEVICE_ADMIN	Allows the holder to send intents to a device administrator. Should never be needed for common applications.
WRITE_SETTINGS	modify global system settings
READ_PHONE_STATE	read phone state and identity
C2D_MESSAGE	Allows cloud to device messaging.
WRITE_SECURE_SETTINGS	Allows an application to modify the system's secure settings data. Not for use by common applications.
VIBRATE	Allows access to the vibrator.
SYSTEM_ALERT_WINDOW	lows an application to show system-alert windows. Malicious applications can take over the entire screen of the phone.
KILL_BACKGROUND_PROCESSES	Allows an application to kill background processes of other applications, even if memory is not low.
CAMERA	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
WAKE_LOCK	Prevent phone from sleeping.

Table C.28: Result of Mobile Security Framework for Always on AMOLED

Manifest Analysis

Issue	Description
Application Data can be Backed up	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
Activity is not Protected.	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
Broadcast Receiver is not Protected	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
Service is not Protected	A Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Service is explicitly exported.

Table C.29: Result of Mobile Security Framework for Always on AMOLED

Code Analysis

Issue	Severity
This App uses Java Hash Code. It's a weak hash function and should never be used in Secure Crypto Implementation.	High
The App uses an insecure Random Number Generator.	High
The App logs information. Sensitive information should never be logged.	Info

Table C.30: Result of Mobile Security Framework for Always on AMOLED

Flashlight

Permissions

Permission	Meaning
WAKE_LOCK	Prevent phone from sleeping.
CAMERA	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
FLASHLIGHT	Allows the application to control the flashlight
INTERNET	Allows applications to open network sockets.
ACCESS_NETWORK_STATE	View network status.

Table C.31: Result of Mobile Security Framework for Flashlight

Manifest Analysis

Issue	Description
Application Data can be Backed up	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
Broadcast Receiver is not Protected	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

Table C.32: Result of Mobile Security Framework for Flashlight

Code Analysis

Issue	Severity
The App logs information. Sensitive information should never be logged.	Info

Table C.33: Result of Mobile Security Framework for Flashlight

Appendix D

Results of Qark

D.1 Tested APKs

D.1.1 Banking

MB Way

Analysis to Manifest

Issue	Severity
Backups enabled: Potential for data theft via local attacks via adb backup, if the device has USB debugging enabled (not common).	Warning

Table D.1: Result of Qark for MB Way

Analysis to App Components

Issue	Severity
The following receiver are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one.	Warning
The following activity are exported, but not protected by any permissions. Failing to protect activity could leave them vulnerable to attack by malicious apps. The activity should be reviewed for vulnerabilities, such as injection and information leakage.	Warning

Table D.2: Result of Qark for MB Way

Analysis to Crypto Bugs

Issue	Severity
setSeed should not be called with SecureRandom, as it is insecure. Specifying a fixed seed will cause the instance to return a predictable sequence of numbers. This may be useful for testing but it is not appropriate for secure use.	Potential Vulnerability
generateSeed should not be called with SecureRandom, as it is insecure. Specifying a fixed seed will cause the instance to return a predictable sequence of numbers. This may be useful for testing but it is not appropriate for secure use.	Potential Vulnerability

Table D.3: Result of Qark for MB Way

Analysis to Pending Intents

Issue	Severity
Empty Pending instance found. For security reasons, the Intent you supply here should almost always be an explicit intent that is specify an explicit component to be delivered to through Intent.setClass. A malicious application could potentially intercept, redirect and/or modify this Intent. Pending Intents retain the UID of your application and all related permissions, allowing another application to act as yours.	Warning

Table D.4: Result of Qark for MB Way

Analysis to Plugin Issues

Issue	Severity
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
The Content provider API provides a method call The framework does no permission checking on this entry into the content provider besides the basic ability for the application to get access to the provider at all. Any implementation of this method must do its own permission checks on incoming calls to make sure they are allowed.Failure to do so will allow unauthorized components to interact with the content provider.	Info
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info

Table D.5: Result of Qark for MB Way

Wallet

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following receiver are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one.	Warning

Table D.6: Result of Qark for Wallet

Analysis to X.509 Issues

Issue	Severity
Instance of checkServerTrusted, with no body found. This means this application is likely vulnerable to Man-In-The-Middle attacks.	Warning

Table D.7: Result of Qark for Wallet

Analysis to Crypto Bugs

Issue	Severity
getInstance should not be called with ECB as the cipher mode, as it is insecure	Potential Vulnerability

Table D.8: Result of Qark for Wallet

Analysis to Pending Intents

Issue	Severity
Implicit Intent: localIntent used to create instance of PendingIntent. A malicious application could potentially intercept, redirect and/or modify (in a limited manner) this Intent. Pending Intents retain the UID of your application and all related permissions, allowing another application to act as yours.	Potential Vulnerability

Table D.9: Result of Qark for Wallet

Analysis to Plugin Issues

Issue	Severity
Reading files stored on External Storage makes it vulnerable to data injection attacks Note that this code does no error checking and there is no security enforced with these files. For example, any application holding WRITE_EXTERNAL_STORAGE can write to these files.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
The Content provider API provides a method call The framework does no permission checking on this entry into the content provider besides the basic ability for the application to get access to the provider at all. Any implementation of this method must do its own permission checks on incoming calls to make sure they are allowed.Failure to do so will allow unauthorised components to interact with the content provider.	Info
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
Application dynamically loads an external class through DexClassLoader Filepat.	Info
File System Access is by default enabled setAllowFileAccess() and setAllowContentAccess() are by default true. This should be set to false to restrict access to local data since it is used to display content from locally stored HTML or fetch HTML and other content from the server.	Info
Application contains hardcoded http url	Info

Table D.10: Result of Qark for Wallet

Issue	Severity
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info
Be careful with use of Check permission function App maybe vulnerable to Privilege escalation or Confused Deputy Attack. This function can grant access to malicious application, lacking the appropriate permission, by assuming your applications permissions. This means a malicious application, without appropriate permissions, can bypass its permission check by using your application permission to get access to otherwise denied resources. Use - checkCallingPermission instead.	Info

Table D.11: Result of Qark for Wallet

D.1.2 In-app Purchases

Drink Water

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning

Table D.12: Result of Qark for Drink Water

Analysis to Crypto Bugs

Issue	Severity
getInstance should not be called with ECB as the cipher mode, as it is insecure	Potential Vulnerability

Table D.13: Result of Qark for Drink Water

Analysis to Pending Intents

Issue	Severity
Implicit Intent: localIntent used to create instance of PendingIntent. A malicious application could potentially intercept, redirect and/or modify (in a limited manner) this Intent. Pending Intents retain the UID of your application and all related permissions, allowing another application to act as yours.	Potential Vulnerability

Table D.14: Result of Qark for Drink Water

Analysis to Plugin Issues

Issue	Severity
Reading files stored on External Storage makes it vulnerable to data injection attacks Note that this code does no error checking and there is no security enforced with these files. For example, any application holding WRITE_EXTERNAL_STORAGE can write to these files.	Warning
Improper implementation of shouldInterceptRequest method Returning null allows any URL to load in the web-view.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
Application dynamically loads an external class through DexClassLoader Filepat.	Info
File System Access is by default enabled setAllowFileAccess() and setAllowContentAccess() are by default true. This should be set to false to restrict access to local data since it is used to display content from locally stored HTML or fetch HTML and other content from the server.	Info

Table D.15: Result of Qark for Drink Water

Issue	Severity
Application contains hardcoded http ur	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info
Access of phone number or IMEI, is detected in file.	Info

Table D.16: Result of Qark for Drink Water

Freelectics Bodyweight

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following receiver are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one.	Warning

Table D.17: Result of Qark for Freelectics Bodyweigh

Analysis to Pending Intents

Issue	Severity
Implicit Intent: localIntent used to create instance of PendingIntent. A malicious application could potentially intercept, redirect and/or modify (in a limited manner) this Intent. Pending Intents retain the UID of your application and all related permissions, allowing another application to act as yours.	Potential Vulnerability

Table D.18: Result of Qark for Freelectics Bodyweigh

Analysis to Plugin Issues

Issue	Severity
Reading files stored on External Storage makes it vulnerable to data injection attacks Note that this code does no error checking and there is no security enforced with these files. For example, any application holding WRITE_EXTERNAL_STORAGE can write to these files.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
Improper implementation of shouldInterceptRequest method Returning null allows any url to load in the web-view.	Warning
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info
Be careful with use of Check permission function App maybe vulnerable to Privilege escalation or Confused Deputy Attack. This function can grant access to malicious application, lacking the appropriate permission, by assuming your applications permissions. This means a malicious application, without appropriate permissions, can bypass its permission check by using your applicationpermission to get access to otherwise denied resources. Use - checkCallingPermission instead.	Info

Table D.19: Result of Qark for Freelectics Bodyweigh

Issue	Severity
The Content provider API provides a method call The framework does no permission checking on this entry into the content provider besides the basic ability for the application to get access to the provider at all. Any implementation of this method must do its own permission checks on incoming calls to make sure they are allowed.Failure to do so will allow unauthorized components to interact with the content provider.	Info
File System Access is by default enabled setAllowFileAccess() and setAllowContentAccess() are by default true. This should be set to false to restrict access to local data since it is used to display content from locally stored HTML or fetch HTML and other content from the server.	Info

Table D.20: Result of Qark for Freelectics Bodyweigh

D.1.3 Games

Clash-Royale

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following receiver are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following activity are exported, but not protected by any permissions. Failing to protect activity could leave them vulnerable to attack by malicious apps. The activity should be reviewed for vulnerabilities, such as injection and information leakage.	Warning

Table D.21: Result of Qark for Clash-Royale

Analysis to Crypto Bugs

Issue	Severity
getInstance should not be called with ECB as the cipher mode, as it is insecure	Potential Vulnerability

Table D.22: Result of Qark for Clash-Royale

Analysis to Pending Intents

Issue	Severity
Implicit Intent: localIntent used to create instance of PendingIntent. A malicious application could potentially intercept, redirect and/or modify (in a limited manner) this Intent. Pending Intents retain the UID of your application and all related permissions, allowing another application to act as yours.	Potential Vulnerability

Table D.23: Result of Qark for Clash-Royale

Analysis to Plugin Issues

Issue	Severity
Reading files stored on External Storage makes it vulnerable to data injection attacks Note that this code does no error checking and there is no security enforced with these files. For example, any application holding WRITE_EXTERNAL_STORAGE can write to these files.	Warning
Improper implementation of shouldInterceptRequest method Returning null allows any URL to load in the web-view.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info

Table D.24: Result of Qark for Clash-Royale

Issue	Severity
Be careful with use of Check permission function App maybe vulnerable to Privilege escalation or Confused Deputy Attack. This function can grant access to malicious application, lacking the appropriate permission, by assuming your applications permissions. This means a malicious application, without appropriate permissions, can bypass its permission check by using your application permission to get access to otherwise denied resources. Use - checkCallingPermission instead.	Info
File System Access is by default enabled setAllowFileAccess() and setAllowContentAccess() are by default true. This should be set to false to restrict access to local data since it is used to display content from locally stored HTML or fetch HTML and other content from the server.	Info
The Content provider API provides a method call The framework does no permission checking on this entry into the content provider besides the basic ability for the application to get access to the provider at all. Any implementation of this method must do its own permission checks on incoming calls to make sure they are allowed.Failure to do so will allow unauthorised components to interact with the content provider.	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info

Table D.25: Result of Qark for Clash-Royale

Millionaire 2018

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following receiver are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one.	Warning

Table D.26: Result of Qark for Millionaire 2018

Analysis to X.509 Issues

Issue	Severity
Potential Man-In-The-Middle vulnerability - There appear to be SSLSession (boolean) objects which are not checked using the HostnameVerifier.verify method. You should manually inspect these.	Warning

Table D.27: Result of Qark for Millionaire 2018

Analysis to Pending Intents

Issue	Severity
Implicit Intent: localIntent used to create instance of PendingIntent. A malicious application could potentially intercept, redirect and/or modify (in a limited manner) this Intent. Pending Intents retain the UID of your application and all related permissions, allowing another application to act as yours.	Potential Vulnerability

Table D.28: Result of Qark for Millionaire 2018

Analysis to Plugin Issues

Issue	Severity
Reading files stored on External Storage makes it vulnerable to data injection attacks Note that this code does no error checking and there is no security enforced with these files. For example, any application holding WRITE_EXTERNAL_STORAGE can write to these files.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning

Table D.29: Result of Qark for Millionaire 2018

Issue	Severity
Improper implementation of shouldInterceptRequest method Returning null allows any url to load in the web-view.	Warning
Usage of setMixedContentMode is found In this mode, the WebView will allow a secure origin to load content from any other origin, even if that origin is insecure. This is the least secure mode of operation for the WebView, and where possible apps should not set this mode.	Warning
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info
Be careful with use of Check permission function App maybe vulnerable to Privilege escalation or Confused Deputy Attack. This function can grant access to malicious application, lacking the appropriate permission, by assuming your applications permissions. This means a malicious application, without appropriate permissions, can bypass its permission check by using your applicationpermission to get access to otherwise denied resources. Use - checkCallingPermission instead.	Info
The Content provider API provides a method call The framework does no permission checking on this entry into the content provider besides the basic ability for the application to get access to the provider at all. Any implementation of this method must do its own permission checks on incoming calls to make sure they are allowed.Failure to do so will allow unauthorized components to interact with the content provider.	Info

Table D.30: Result of Qark for Millionaire 2018

Issue	Severity
File System Access is by default enabled setAllowFileAccess() and setAllowContentAccess() are by default true. This should be set to false to restrict access to local data since it is used to display content from locally stored HTML or fetch HTML and other content from the server.	Info
Application dynamically loads an external class through DexClassLoader.Even though this may not be a security issue always, be careful with what you are loading.	Info

Table D.31: Result of Qark for Millionaire 2018

D.1.4 Privacy

LOCKit

Analysis to Manifest

Issue	Severity
Backups enabled: Potential for data theft via local attacks via adb backup, if the device has USB debugging enabled (not common).	Warning

Table D.32: Result of Qark for LOCKit

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following receiver are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one.	Warning
The following service are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one.	Warning
The following activity-alias are exported, but not protected by any permissions. Failing to protect activity-alias could leave them vulnerable to attack by malicious apps. The activity-alias should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following activity are exported, but not protected by any permissions. Failing to protect activity could leave them vulnerable to attack by malicious apps. The activity should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
These are exported, but the associated Intents can only be sent by SYSTEM level apps. They could still potentially be vulnerable, if the Intent carries data that is tainted (2nd order injection)	Info
PROTECTED BROADCASTS	Info

Table D.33: Result of Qark for LOCKit

Analysis to X.509 Issues

Issue	Severity
Instance of checkServerTrusted, with no body found. This means this application is likely vulnerable to Man-In-The-Middle attacks. This can be confirmed using the free version of Burpsuite.	Warning

Table D.34: Result of Qark for LOCKit

Analysis to Crypto Bugs

Issue	Severity
getInstance should not be called with ECB as the cipher mode, as it is insecure.	Potential Vulnerability

Table D.35: Result of Qark for LOCKit

Analysis to Plugin Issues

Issue	Severity
Reading files stored on External Storage makes it vulnerable to data injection attacks Note that this code does no error checking and there is no security enforced with these files. For example, any application holding WRITE_EXTERNAL_STORAGE can write to these files.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
Web-view is loading http URLs If Web-view is allowing to load clear-text content from the Internet then it would be open to various forms of attack such as MiTM.	Warning
Access of phone number or IMEI, is detected in file	Info
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info
Application contains hard-coded http URL.	Info
File System Access is by default enabled setAllowFileAccess() and setAllowContentAccess() are by default true. This should be set to false to restrict access to local data since it is used to display content from locally stored HTML or fetch HTML and other content from the server.	Info

Table D.36: Result of Qark for LOCKit

My Passwords - Password Manager

Analysis to Manifest

Issue	Severity
Backups enabled: Potential for data theft via local attacks via adb backup, if the device has USB debugging enabled (not common).	Warning

Table D.37: Result of Qark for My Passwords

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following receiver are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one.	Warning

Table D.38: Result of Qark for My Passwords

Analysis to Pending Intents

Issue	Severity
Implicit Intent: localIntent used to create instance of PendingIntent. A malicious application could potentially intercept, redirect and/or modify (in a limited manner) this Intent. Pending Intents retain the UID of your application and all related permissions, allowing another application to act as yours.	Potential Vulnerability

Table D.39: Result of Qark for My Passwords

Analysis to Plugin Issues

Issue	Severity
Reading files stored on External Storage makes it vulnerable to data injection attacks Note that this code does no error checking and there is no security enforced with these files. For example, any application holding WRITE_EXTERNAL_STORAGE can write to these files.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
The Content provider API provides a method call The framework does no permission checking on this entry into the content provider besides the basic ability for the application to get access to the provider at all. Any implementation of this method must do its own permission checks on incoming calls to make sure they are allowed.Failure to do so will allow unauthorized components to interact with the content provider.	Info
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info
Be careful with use of Check permission function App maybe vulnerable to Privilege escalation or Confused Deputy Attack. This function can grant access to malicious application, lacking the appropriate permission, by assuming your applications permissions. This means a malicious application, without appropriate permissions, can bypass its permission check by using your applicationpermission to get access to otherwise denied resources. Use - checkCallingPermission instead.	Info

Table D.40: Result of Qark for My Passwords

D.1.5 Tools

Always on AMOLED

Analysis to Manifest

Issue	Severity
Backups enabled: Potential for data theft via local attacks via adb backup, if the device has USB debugging enabled (not common).	Warning

Table D.41: Result of Qark for Always on AMOLE

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following receiver are exported and protected by a permission, but the permission can be obtained by malicious apps installed prior to this one.	Warning

Table D.42: Result of Qark for Always on AMOLE

Analysis to Pending Intents

Issue	Severity
Implicit Intent: localIntent used to create instance of PendingIntent. A malicious application could potentially intercept, redirect and/or modify (in a limited manner) this Intent. Pending Intents retain the UID of your application and all related permissions, allowing another application to act as yours.	Potential Vulnerability

Table D.43: Result of Qark for Always on AMOLE

Analysis to Plugin Issues

Issue	Severity
Reading files stored on External Storage makes it vulnerable to data injection attacks Note that this code does no error checking and there is no security enforced with these files. For example, any application holding WRITE_EXTERNAL_STORAGE can write to these files.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info
Application contains hard-coded http URL.	Info
Be careful with use of Check permission function App maybe vulnerable to Privilege escalation or Confused Deputy Attack. This function can grant access to malicious application, lacking the appropriate permission, by assuming your applications permissions. This means a malicious application, without appropriate permissions, can bypass its permission check by using your applicationpermission to get access to otherwise denied resources. Use - checkCallingPermission instead.	Info

Table D.44: Result of Qark for Always on AMOLE

Flashlight

Analysis to App Components

Issue	Severity
The following receiver are exported, but not protected by any permissions. Failing to protect receiver could leave them vulnerable to attack by malicious apps. The receiver should be reviewed for vulnerabilities, such as injection and information leakage.	Warning
The following activity are exported, but not protected by any permissions. Failing to protect activity could leave them vulnerable to attack by malicious apps. The activity should be reviewed for vulnerabilities, such as injection and information leakage.	Warning

Table D.45: Result of Qark for Flashlight

Analysis to Plugin Issues

Issue	Severity
Usage of setMixedContentMode is found In this mode, the WebView will allow a secure origin to load content from any other origin, even if that origin is insecure. This is the least secure mode of operation for the WebView, and where possible apps should not set this mode.	Warning
Application dynamically registers a broadcast receiver Application that register a broadcast receiver dynamically is vulnerable to granting unrestricted access to the broadcast receiver. The receiver will be called with any broadcast Intent that matches filter.	Warning
Verbose logs are detected This may allow potential leakage of information from Android applications. Verbose logs should never be compiled into an application except during development.	Info
File System Access is by default enabled setAllowFileAccess() and setAllowContentAccess() are by default true. This should be set to false to restrict access to local data since it is used to display content from locally stored HTML or fetch HTML and other content from the server.	Info
The Content provider API provides a method call The framework does no permission checking on this entry into the content provider besides the basic ability for the application to get access to the provider at all. Any implementation of this method must do its own permission checks on incoming calls to make sure they are allowed.Failure to do so will allow unauthorized components to interact with the content provider.	Info
Application dynamically loads an external class through DexClassLoader. Even though this may not be a security issue always, be careful with what you are loading.	Info
Debug logs are detected This may allow potential leakage of information from Android applications. Debug logs should never be compiled into an application except during development.	Info

Table D.46: Result of Qark for Flashlight

Appendix E

Results of Super Android Analyser

E.1 Tested APKs

E.1.1 Banking

MB Way

Issue	Severity
This application is vulnerable to SQL injection. Any data stored in database can be exposed as any attacker is able to retrieve, modify and delete the stored information.	Critical vulnerability
Using weak algorithms allows an attacker to break the cyphered communications gaining access to plain text content.	High critically vulnerability
This option allows backups of the application data via adb. Malicious people with physical access could use adb to get private data of your app into their PC.	Medium critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression.	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability
Even if the application can create its own permissions, it's discouraged, since it can lead to misunderstanding between developers.	Low critically vulnerability

Table E.1: Result of Super Android Analyser for MB Way

Issue	Severity
Exported activity was found. It can be used by other applications.	Warning
Exported receiver was found. It can be used by other applications.	Warning
Exported service was found. It can be used by other applications.	Warning
The application needs a large heap. This is not a vulnerability as such, but could be in devices with small heap. Check if the large heap is actually needed.	Warning

Table E.2: Result of Super Android Analyser for MB Way

Wallet

Issue	Severity
Insecure application SSL implementation. This application accepts all certificates, including self signed by default. This is a critical issue as Man in the Middle attacks may be performed.	Critical vulnerability
This application is vulnerable to SQL injection. Any data stored in database can be exposed as any attacker is able to retrieve, modify and delete the stored information.	Critical vulnerability
Webview insecure implementation. This issue could allow to a remote attacker to code execution in WebView and performing Cross Site Scripting attacks.	Critical vulnerability
The application could execute system command.	High critically vulnerability
Applications is creating temp files. Sensitive information should never be written in temp files.	High critically vulnerability
Using weak algorithms allows an attacker to break the ciphered communications gaining access to plain text content.	High critically vulnerability
Application can read/write in external storage. Any app can read data written in external storage.	High critically vulnerability
The exceptions thrown by a method should be specific. Generic Exception type could could not be safe and lead to silent error suppression.	Low critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression.	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sleep Method is used with vars as arguments. If those vars are modified it could force the application to stop indefinitely.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability

Table E.3: Result of Super Android Analyser for Wallet

Issue	Severity
This application is using Base64 encoding. This is not a secure method to encode data.	Warning
The decompilation of the source code could lead to the disclosure of private email information.	Warning
Exported activity was found. It can be used by other applications.	Warning
Exported receiver was found. It can be used by other applications.	Warning
The application is recording the device network operator. This process might be performed without the user's knowledge.	Warning
The application is recording the device network operator name. This process might be performed without the user's knowledge.	Warning
The decompilation of the source code could lead to the disclosure of private IPs.	Warning
The application needs a large heap. This is not a vulnerability as such, but could be in devices with small heap. Check if the large heap is actually needed.	Warning
The decompilation of the source code could lead to the disclosure of private URLs.	Warning

Table E.4: Result of Super Android Analyser for Wallet

E.1.2 In-app Purchases

Drink Water

Issue	Severity
Using weak algorithms allows an attacker to break the ciphered communications gaining access to plain text content.	High critically vulnerability
This option allows backups of the application data via adb. Malicious people with physical access could use adb to get private data of your app into their PC.	Medium critically vulnerability
The exceptions thrown by a method should be specific. Generic Exception type could could not be safe and lead to silent error suppression.	Low critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability

Table E.5: Result of Super Android Analyser for Drink Water

Issue	Severity
The This application is using Base64 encoding and decoding. This is not a secure method to encode data.	Warning
The decompilation of the source code could lead to the disclosure of hard-coded certificate or key-store.	Warning
The decompilation of the source code could lead to the disclosure of private email information.	Warning
Exported activity was found. It can be used by other applications.	Warning
Exported receiver was found. It can be used by other applications.	Warning
The application is recording the device network operator. This process might be performed without the user's knowledge.	Warning
The application is recording the SIM serial. This process might be performed without the user's knowledge.	Warning
The decompilation of the source code could lead to the disclosure of private URLs.	Warning

Table E.6: Result of Super Android Analyser for Drink Water

Freelectics Bodyweight

Issue	Severity
Applications is creating temp files. Sensitive information should never be written in temp files.	High critically vulnerability
Using weak algorithms allows an attacker to break the ciphered communications gaining access to plain text content.	High critically vulnerability
Application can read/write in external storage. Any app can read data written in external storage.	High critically vulnerability
The exceptions thrown by a method should be specific. Generic Exception type could could not be safe and lead to silent error suppression.	Low critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sleep Method is used with vars as arguments. If those vars are modified it could force the application to stop indefinitely.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability
Even if the application can create its own permissions, it's discouraged, since it can lead to misunderstanding between developers.	Low critically vulnerability

Table E.7: Result of Super Android Analyser for Freelectics Bodyweight

Issue	Severity
This application is using Base64 decoding.	Warnings
The decompilation of the source code could lead to the disclosure of private email information.	Warnings
Exported activity was found. It can be used by other applications.	Warnings
Exported receiver was found. It can be used by other applications.	Warnings
Exported service was found. It can be used by other applications.	Warnings
The decompilation of the source code could lead to the disclosure of private IPs.	Warnings
The decompilation of the source code could lead to the disclosure of private URLs.	Warnings

Table E.8: Result of Super Android Analyser for Freelectics Bodyweight

E.1.3 Games

Clash-Royale

Issue	Severity
Webview insecure implementation. This issue could allow to a remote attacker to code execution in WebView and performing Cross Site Scripting attacks.	Critical vulnerability
Using weak algorithms allows an attacker to break the ciphered communications gaining access to plain text content.	High critically vulnerability
Application can read/write in external storage. Any app can read data written in external storage.	High critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability

Table E.9: Result of Super Android Analyser for Clash-Royale

Issue	Severity
The This application is using Base64 encoding and decoding. This is not a secure method to encode data.	Warning
Exported activity was found. It can be used by other applications.	Warning
Exported receiver was found. It can be used by other applications.	Warning
The decompilation of the source code could lead to the disclosure of private IPs.	Warning
The decompilation of the source code could lead to the disclosure of private URLs.	Warning

Table E.10: Result of Super Android Analyser for Clash-Royale

Millionaire 2018

Issue	Severity
Web-view insecure implementation. This issue could allow to a remote attacker to code execution in Web-view and performing Cross Site Scripting attacks.	Critical vulnerability
This applications is performing checks for rooted device. This could be use to execute specific code if the device is rooted to take control of it.	High critically vulnerability
Using weak algorithms allows an attacker to break the ciphered communications gaining access to plain text content.	High critically vulnerability
The exceptions thrown by a method should be specific. Generic Exception type could could not be safe and lead to silent error suppression.	Low critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sleep Method is used with vars as arguments. If those vars are modified it could force the application to stop indefinitely.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability

Table E.11: Result of Super Android Analyser for Millionaire 2018

Issue	Severity
This application is using Base64 encoding. This is not a secure method to encode data.	Warning
This application is using Base64 decoding.	Warning
Exported activity was found. It can be used by other applications.	Warning
Exported provider was found. It can be used by other applications.	Warning
The decompilation of the source code could lead to the disclosure of private IPs.	Warning
The decompilation of the source code could lead to the disclosure of private URLs.	Warning

Table E.12: Result of Super Android Analyser for Millionaire 2018

E.1.4 Privacy

LOCKit

Issue	Severity
Insecure application SSL implementation. This application accepts all certificates, including self signed by default. This is a critical issue as Man in the Middle attacks may be performed.	Critical vulnerability
This application is vulnerable to SQL injection. Any data stored in database can be exposed as any attacker is able to retrieve, modify and delete the stored information.	Critical vulnerability
This applications is performing checks for rooted device. This could be use to execute specific code if the device is rooted to take control of it.	High critically vulnerability
Using weak algorithms allows an attacker to break the ciphered communications gaining access to plain text content.	High critically vulnerability
Application can read/write in external storage. Any app can read data written in external storage.	High critically vulnerability
This option allows backups of the application data via adb. Malicious people with physical access could use adb to get private data of your app into their PC.	Medium critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sleep Method is used with vars as arguments. If those vars are modified it could force the application to stop indefinitely.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability

Table E.13: Result of Super Android Analyser for LOCKit

Issue	Severity
This application is using Base64 encoding. This is not a secure method to encode data.	Warning
This application is using Base64 decoding.	Warning
The decompilation of the source code could lead to the disclosure of hard-coded certificate or keystore.	Warning
The decompilation of the source code could lead to the disclosure of private email information.	Warning
Exported activity was found. It can be used by other applications.	Warning
Exported activity was found. It can be used by other applications.	Warning
Exported activity-alias was found. It can be used by other applications.	Warning
Exported receiver was found. It can be used by other applications.	Warning
Exported service was found. It can be used by other applications.	Warning
The application is recording the device network operator name. This process might be performed without the user's knowledge.	Warning
The decompilation of the source code could lead to the disclosure of private IPs.	Warning
The application needs a large heap. This is not a vulnerability as such, but could be in devices with small heap. Check if the large heap is actually needed.	Warning
The decompilation of the source code could lead to the disclosure of private URLs.	Warning

Table E.14: Result of Super Android Analyser for LOCKit

My Passwords - Password Manager

Issue	Severity
Applications is creating temp files. Sensitive information should never be written in temp files.	High critically vulnerability
Application can read/write in external storage. Any app can read data written in external storage.	High critically vulnerability
This option allows backups of the application data via adb. Malicious people with physical access could use adb to get private data of your app into their PC.	Medium critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability

Table E.15: Result of Super Android Analyser for My Passwords

Issue	Severity
The decompilation of the source code could lead to the disclosure of private email information.	Warning
Exported activity was found. It can be used by other applications.	Warning
The decompilation of the source code could lead to the disclosure of private URLs.	Warning

Table E.16: Result of Super Android Analyser for My Passwords

E.1.5 Tools

Always on AMOLED

Issue	Severity
This option allows backups of the application data via adb. Malicious people with physical access could use adb to get private data of your app into their PC.	Medium critically vulnerability
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression	Low critically vulnerability
This method is not as random as it is supposed to be. It should not be use to generate OTP codes.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability
Even if the application can create its own permissions, it's discouraged, since it can lead to misunderstanding between developers.	Low critically vulnerability

Table E.17: Result of Super Android Analyser for Always on AMOLED

Issue	Severity
This application is using Base64 encoding. This is not a secure method to encode data.	Warning
This application is using Base64 decoding.	Warning
Exported activity was found. It can be used by other applications.	Warning
Exported receiver was found. It can be used by other applications.	Warning
Exported service was found. It can be used by other applications.	Warning
The decompilation of the source code could lead to the disclosure of private URLs.	Warning

Table E.18: Result of Super Android Analyser for Always on AMOLED

Flashlight

Issue	Severity
Exception catching should be specific. Generic Exception type could not be safe and lead to silent error suppression.	Low critically vulnerability
Sensitive information should never be logged since it can lead to that information being disclosed.	Low critically vulnerability

Table E.19: Result of Super Android Analyser for Flashlight

Issue	Severity
The decompilation of the source code could lead to the disclosure of private email information.	Warning
Exported activity was found. It can be used by other applications.	Warning

Table E.20: Result of Super Android Analyser for Flashlight