

Securitometer: Developing an Overall Security Metrics for Android Device

Muhammad Rezqi

Master of Science
School of Informatics
University of Edinburgh
2016

Abstract

This dissertation presents our work on the development of Securimeter, an Android application that probe for security flaws and vulnerabilities of the device where it is installed, and provide a security score of the device. We developed three type of security test for Securimeter namely: malware classification test, app permission test, and device vulnerability test. We compiled the results of the three tests into two security metrics: the app metric and the operating system metric. The design decisions for this application were based on our research on the needs of penetration testers and IT security enthusiast. We measured the success of this project by comparing the result of our initial study that elicited the requirements for Securimeter with the result of our evaluation for the final version of Securimeter. To conclude the project, we also conducted a post-system usability and acceptability study. We found that Securimeter is regarded as useful not only to security professionals but also the general Android users. We also found that these users regarded the experience of using Securimeter as considerably satisfying in terms of its level of usefulness, information quality, and interface quality.

Acknowledgements

I would like to thank my supervisor, David Aspinall, for trusting me with this project and providing me with invaluable guidance, feedback, and advice. I would also like to thank Kami Vaniea and members of the the AppGuarden Group who have provided me with numerous feedback, literature, and supports.

Waves of gratitude go to my dearest family members and PPI Edinburgh friends who have affectionately supported me through my study. I would also like to extend my gratitude to Dinda Sarasannisa for keeping me sane during the exam and dissertation period. I am thankful to have the opportunity to work together on our dissertations and exchanging constructive inputs.

I would also like to thank Ulfah Lukman, colleagues at Deloitte Computer Forensic services, fellow IT security professionals at PwC and Ernst & Young for their abundance of help especially in assisting me to conduct the survey and evaluation of Securitometer.

Finally, I would like to express my gratitude to the Indonesian Endowment for Education (*Lembaga Pengelola Dana Pendidikan / LPDP*) who has given me the chance to study at the University of Edinburgh.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Muhammad Rezqi)

To my parents,
Khaermenawati and Syahrial,
who overcame great hardships and sacrifices
to provide me the opportunity to pursue my dreams

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	2
1.3	Achievements	3
1.3.1	The Securimeter Android Application	3
1.3.1.1	Use of CVE and CVSS for Scoring Mechanism	5
1.3.2	Porting a Python Classifier to Java for Android	5
1.3.3	Usability Aspects of Securimeter Application	5
1.4	Document Outline	6
2	Background	7
2.1	Software Vulnerability	7
2.1.1	Industry Standard: CVE and CVSS	8
2.1.1.1	Common Vulnerabilities and Exposures (CVE)	8
2.1.1.2	Common Vulnerability Scoring System (CVSS)	8
2.1.2	Android Security and Vulnerability	8
2.2	Limitations and Weakness of Android Security Model	9
2.3	Previous Work on Android Security Metrics	11
2.3.1	Security Metrics for the Android Ecosystem	11
2.3.2	Malicious Application Classifier for Android Apps	13
3	Requirement Specification	15
3.1	Requirement Elicitation	15
3.1.1	Research on Related Projects	15
3.1.1.1	Virus Total for Android	16
3.1.1.2	X-Ray	17
3.1.1.3	Android VTS	18

3.1.1.4	Risk-Ranking for Android Permissions	19
3.1.1.5	Insights from Related Projects	20
3.1.2	Brainstorming	21
3.1.3	Prototyping and Survey	22
3.1.4	Focus Group Discussion	24
3.2	Recap and Summary	26
4	Design and Implementation	29
4.1	Technology Specification	29
4.1.1	Minimum Target Platform	29
4.1.2	Solution Architecture	29
4.2	Back-end: Three Tests Performed	31
4.2.1	Malware Classification Test	31
4.2.1.1	Extracting API Calls	31
4.2.1.2	Extracting Permissions	37
4.2.1.3	The Classification	40
4.2.2	Application Permission Test	42
4.2.3	Device Vulnerability Test	43
4.2.3.1	Filtering and Executing the Test	45
4.3	Back-end: Two Scoring Mechanism	47
4.3.0.1	App Score	47
4.3.0.2	OS Score	48
4.4	Front-end: Securitometer Application	48
4.4.1	Splash Screen and Introductory Pages	49
4.4.2	Main Page	50
4.4.3	Running the Test	51
4.4.4	Displaying the Result	52
4.4.5	Reporting the Result	56
5	Evaluation	58
5.1	Feature-based Evaluation	58
5.1.1	Goals and Methodology	58
5.1.2	Results and Analysis	58
5.2	Usability and Acceptability Study	61
5.2.1	Goals and Methodology	61
5.2.2	Results and Analysis	64

5.2.2.1	Initial Survey	64
5.2.2.2	Final Survey	69
6	Conclusion	76
6.1	Concluding Remarks	76
6.2	Limitations of this Project	77
6.3	Future Work	78
A	Appendix: Screenshot of Securitometer	79
B	Appendix: Sample Exported Result from Securitometer	85
C	Appendix: Initial Survey for Securitometer Prototype	89
D	Appendix: Final Survey for the Final Version of Securitometer	109
	Bibliography	125

List of Figures

2.1	The security level of Android devices	12
3.1	Traffic light scheme for the scanning results	16
3.2	X-Ray displaying vulnerability test result	17
3.3	Android VTS displaying vulnerability test result	18
3.4	Ranking of permission according to their risk	19
3.5	Occurrence percentage of top 40 Android permissions in malicious and benign apps	20
3.6	Focus group for Securitometer in progress (blurred for privacy)	25
4.1	Securitometer components	30
4.2	dexdump ran in Android	35
4.3	Feature name and its corresponding weight	40
4.4	ExplainDroid's malware classifier result	41
4.5	Securitometer's malware classifier result	41
4.6	Interface for each vulnerability test	45
4.7	Score calculation for apps permission test	47
4.8	(a) Splash screen (b-f) Introductory pages	49
4.9	Empty main page	50
4.10	(a) App test (b) OS test	51
4.11	Securitometer progress in notification	52
4.12	Test result summaries	53
4.13	Overall test result	54
4.14	Changes made from oldest to the latest version	54
4.15	(a) App test results (b) OS test results	55
4.16	(a) App details (b) OS vulnerability details	56
4.17	Export result options	57

5.1	Methodology for usability and acceptability study	62
5.2	Demographics for initial survey	64
5.3	Classification of respondents for initial survey	65
5.4	Securitometer's potential users	67
5.5	Survey result on the app security metrics	68
5.6	Survey result on the OS security metrics	68
5.7	Respondent's level of understanding	69
5.8	Demographics for final survey	70
5.9	Final survey result on the app security metrics	70
5.10	Final survey result on the OS security metrics	71
5.11	Respondent's level of understanding in final survey	71

List of Tables

4.1	Fuzion24 vulnerability test suite	44
5.1	PSSUQ's list of questions	63
5.2	PSSUQ's questions 1-8	72
5.3	PSSUQ's questions 9-15	73
5.4	PSSUQ's questions 16-18	74
5.5	PSSUQ's question 19	74

List of source codes

1	API calls extraction in the original ExplainDroid's malware classifier .	32
2	Extract APK files of all installed apps	34
3	Excluding system apps and Securitometer itself	34
4	Executing <i>dexdump</i> in Securitometer	37
5	Permission extraction in the original ExplainDroid's malware classifier	38
6	Extracting permissions in Securitometer	39
7	Ranking of permission in JSON	43
8	Vulnerability organizer	47
9	Sample report by Securitometer	88

Chapter 1

Introduction

1.1 Motivation

As smartphone technology integrates more with every aspect of people's lives, it also starts to find its way to the corporate environment. Some companies have already started to provide their employee with these devices in order to make them more accessible and productive. Other companies decided to implement a Bring-Your-Own-Device (BYOD) policy where employees are free to use their smartphone to connect to company's network and access work resources [12][25].

Both providing company-issued smartphone and the BYOD policy present a new threat to the organisation as it can be used as a channel to leak sensitive data [42]. This is because smartphones are the next big target for hackers and malicious attackers[9][15]. In 2013 alone, about 38% smartphone users have experienced cyber attacks [4]. Considering the fact that Android phones dominated the smartphone market with a share of 82.8% [16], and the fact that about 87.7% of Android phones used around the world are exposed to at least one critical vulnerabilities [32] [9], we can infer that more than half of smartphone owners are vulnerable to cyber attacks [9]. This is why companies should be more aware of the kind of devices that are used to access their private resources.

Usually, cyber threats at the corporate level are mitigated by conducting a penetration test on the company's computing system [11]. Penetration testing is a method to evaluate the security level of computer systems by simulating an attack towards it. Traditionally, penetration testing is conducted to identify flaws and vulnerabilities in the corporate website, software, network, and database. However as people are moving away from the use of desktop computers, we should start to develop new security tools

and security procedures that are more suited and is specially made for mobile devices.

Our goal is to develop a tool, called *Securitometer*, which probe for security flaws and vulnerabilities of the device where it is installed in order to evaluate their security level. Note that our goal is very similar with that of penetration testing. While smart-phones come in many platforms, this project chose to first focus on Android since its users remain as the main target of cyber-criminals according to the latest Internet Security Threat Report by Norton [22].

1.2 Objectives

The objective of this project is to develop *Securitometer*, an Android application that will probe for security flaws and vulnerabilities of the device where it is installed, and provide a security score of the device. The security score will represent the level of risk that the owner of the device is exposed to.

We realised that the perceived security level of a mobile device is affected by many factors, e.g. how the device is used, what type of data is stored, and the collection of apps installed on the device. There are a number of individual tools that are available to use to analyse the security level of Android applications. Some tools focus on examining the app permissions, some use program analysis method, and others check for malicious features [7][29]. Our application will focus on analysing the type of resource accessed by the installed apps within the device, whether or not the installed apps behave maliciously, and on the vulnerabilities that the device are exposed to.

We believe that the most potential users of *Securitometer* come from the community of penetration testers and IT security auditors. However, *Securitometer* can also be useful to the general Android user that are enthusiastic about their device security. We created *Securitometer* with both communities as our target. At the end of this project, we conducted a survey where one of the questions is aimed to find out who seems to benefit the most from this application.

With regards to penetration testing, a company should aim to have a high average of security level for all of the mobile devices used within their network. Our aim is for this application to become a tool that can help penetration tester and security reviewer in assessing the security level of smartphones, especially Android, that are used in corporate environment.

The following examples demonstrate the sample situation where this application might be useful at:

- A company owner might be concerned about the use of corporate phones for confidential and work-related purpose. He could hire a penetration tester to install this application to a number of company-issued devices, extract the result, analyse it, write a report and make recommendations.
- A company owner might want to do a quick overview of how secure their employee's personal device are because it is commonly used for work-related communications without violating their workers' privacy. He could ask his employees to install this application, export the result, and send it to the company's security analyst.

A usability study will be conducted to Securitometer's potential users (penetration testers, IT security auditors, and security enthusiast) to find out the overall user satisfaction when using Securitometer, how useful do they think this application is, the quality of information provided in the application, and the quality of the interface. We will not be designing a new procedure on how to do penetration testing with mobile phones as the target. Securitometer's sole purpose is to be one of the security testing tools that could provide its user with the security score and overview of the device where it is installed.

1.3 Achievements

Through our work on this project, we have developed the Securitometer application, which is available for download at <http://bit.ly/2bmeRUQ>. We did not submit Securitometer to Google Play store as the nature of the application might raise security warnings from the Google Play store publishing policy [3].

1.3.1 The Securitometer Android Application

The final version of Securitometer performs 3 type of test:

1. Malware classification test
2. App permission test
3. Device vulnerability test

The above three test are then compiled into 2 scoring mechanisms, 1 score for the operating system (OS) and 1 for the collection of apps. The malware classifier and the

app permission test contribute to the score of installed apps. The device vulnerability test contributes to the score of the target device's operating system.

The followings are the 2 scoring mechanisms that are measured by the final version of Securitometer:

1. App Scoring

The process of determining the app score consists of two parts, as follow:

(a) Malware Classification Test

The first part runs a malware classifier. This classifier extracts the requested permission and API calls from each app and decides whether they are malicious or benign. If they are malicious, the app will get the lowest score and labelled as malware.

(b) App Permission Test

The second part runs if the verdict from the first part is benign. Apps are not just labelled as benign, but also given a risk level (high risk, medium risk, or low risk). This risk level is determined by comparing the permissions that they requested against a list of top 40 riskiest permissions that are commonly found in malicious apps. This list gave a risk score for each permission and is based on an experiment that has analysed huge numbers of malicious and benign apps [37]. Each app starts with a perfect 10.0 score, which is then reduced for each time any of its requested permission are listed in the top 40 riskiest permissions.

2. OS Scoring / Device Vulnerability Test

The process of determining the OS score starts by running a device vulnerability scanner. It will scan the device to detect the presence of any publicly known vulnerabilities. This scan does not exploit the vulnerabilities and is based on an existing vulnerability testing suite [3]. The device score starts with a perfect 10.0 score, which is then reduced for each time any vulnerabilities is found in the device. We made use of the fact that each vulnerability has a CVSS2 score that represents its severity.

1.3.1.1 Use of CVE and CVSS for Scoring Mechanism

Securitometer makes use of the industry standard CVE (Common Vulnerabilities and Exposures) ¹ and CVSS (Common Vulnerability Scoring System) in order to assign a security score to the target device's operating system. The motivation behind this was to exploit the familiarity aspect from penetration testers and IT security auditors. Most penetration testers and IT security auditors are already very familiar with the use of CVE and CVSS [6].

1.3.2 Porting a Python Classifier to Java for Android

Securitometer incorporated a malicious android applications classifier, called ExplainDroid, within its test suite [7]. In doing this, we ported the original classifier, which was originally coded in Python, to Java Android.

The initial motivation of incorporating this classifier was to introduce more context to the app scoring mechanism. Before we ported and incorporated the classifier to Securitometer, the app score was only determined by identifying the type of permissions that are requested by installed apps and assign risk level accordingly. By incorporating this classifier, Securitometer could make a verdict about an app, whether it is malicious or benign. If it is malicious, Securitometer will assign the lowest score to the app. If it is benign, then Securitometer will proceed in assigning risk level based on the apps' requested permissions.

The classifier requires the execution of two android binary tools from the Android SDK for features extraction: *dexdump* ², and *aapt* ³ which stands for Android Asset Packaging Tool ⁴. We manage to simulate what these binaries do in our application and completed the porting process of this classifier.

1.3.3 Usability Aspects of Securitometer Application

We ran multiple requirement elicitation activities and a number of usability test after each development iteration to ensure that Securitometer is not just functional but also pleasant to use. We take into considerations every input from each requirement elicitation activities which include: focus group discussion, brainstorming, prototyping, and

¹<https://cve.mitre.org/about/>

²<http://elinux.org/Master-android>

³http://elinux.org/Android_aapt

⁴<https://developer.android.com/studio/command-line/index.html>

research of related projects.

One of the cases where we take usability into account is when we get the result of incorporating malicious application classifier to Securitometer. Running a malware classifier that requires feature extraction from each application take away a lot of resources in running time. The test run significantly slower compared to when we only assign a risk level to apps' requested permission without running the classifier. Due to this, we decided to add a notification feature that will let the user leave Securitometer to run in the background while still be able to check the test progress in the notification bar. We also added the ability to skip the app permission test for apps that the user are confident about whether it is a threat or not to the security of the device.

1.4 Document Outline

The rest of the chapters will be organised as follow:

- Chapter 2, Background, will introduce relevant theoretical background to set the project in its proper context. We will discuss software vulnerabilities, android security ecosystem, and previous attempts in developing security metrics for Android
- Chapter 3, Requirement Specifications, will lay out the requirement elicitation activities that we conducted in order to achieve the highest appreciation for the app that we were about to develop. At the end, we also provided a numbered summary of the requirements elicited to make evaluating whether or not we have fulfilled them easier.
- Chapter 4, Design and Implementation, will present the various components which have been developed as part of the Securitometer application. We explained each component's scope, function, and design justification in details.
- Chapter 5, Evaluation, will present this project's success measurement, evaluation goals, methodologies, and the evaluation results.
- Chapter 6, Conclusion, will summarise the whole project, evaluate it critically, and lay out recommendations for future works.

Chapter 2

Background

2.1 Software Vulnerability

Software is a set of computer instructions built by programmers to achieve a predefined goal. Often referred as computer program or system, it is a common component of digital devices and has successfully become a huge part of the modern computing world.

Programmers sometimes make mistakes during the development of software. Often times, these mistakes are discovered and exploited at later times by malicious users to alter the normal behaviour of the software. These mistakes, flaws, and software weaknesses are often referred to as software vulnerability [17].

A vulnerable software offers an attack surface for users. Users who are exposed to these surface can become an attacker who utilises these vulnerabilities as an entry point. As a result of an attack, the system could be compromised, damaged, or leaking confidential and privileged information. This is why software security is becoming increasingly popular.

Software can be categorised into two types: operating system and application program (app or application for short). Operating system performs the task that keeps the device running, while application program helps users to achieve a particular task. Both operating system and application program are vulnerable to various exploits [18].

2.1.1 Industry Standard: CVE and CVSS

2.1.1.1 Common Vulnerabilities and Exposures (CVE)

Before the standardisation, most information security practitioners had their own database for software vulnerabilities. People gave their own name and their own severity level to each vulnerability that they found and manage. At that time, different companies may refer to the same problem but with a completely different name and description. This was very ineffective.

In 1999, a dictionary of common names, called CVE (Common Vulnerabilities and Exposures) was launched as a solution to these interoperable problems. CVE acts as a dictionary and provides a standardised name and identifier to all known software vulnerabilities. It is now the industry standard for naming vulnerabilities. For example, if a security practitioner mentions a CVE name called "CVE-1999-0580", everyone will understand that he is referring to the same problem of inappropriate permission in a Windows NT system. CVE dictionary is maintained by The MITRE Corporation.¹

2.1.1.2 Common Vulnerability Scoring System (CVSS)

Common Vulnerability Scoring System (CVSS) is one of the industry standards that attempts to assess the severity of software vulnerabilities. CVSS are commonly used to assist organisations in prioritising the remediation activities that are generated as a result of vulnerabilities discovery. CVSS are also used to calculate the severity score of vulnerabilities discovered on a system.

CVSS scores are calculated by taking into account the complexity to exploit the vulnerabilities and the impact of exploiting it. CVSS scores range from 0 to 10, with 10 being the most severe.²

2.1.2 Android Security and Vulnerability

Android is an increasingly popular mobile operating system which was developed as an open source project. It is based on the Linux kernel and has now become the most used operating system on various mobile devices. It was made to be very flexible and open, thus increasing its popularity [9][32].

¹<https://cve.mitre.org/about/>

²<https://nvd.nist.gov/cvss.cfm>

Google provided developers with Android SDK³, which purpose is to help developers create applications that can run on the Android platform. Google also provided Android users with a marketplace called the Google Play⁴ store where millions of applications were uploaded, ready to be downloaded and installed by a large number of Android users. While both initiatives by Google provides a very good popularity boost to Android, it also provides attackers with a lot of information regarding the platform and channels to launch attacks.

The concern around Android security has been going around three topics: Android malware [43] and Android vulnerabilities [32][13]. Both are a major problem to the mobile platform and has been receiving a lot of attention both in the industry and academic research.

The Android platform has an alarmingly large number of malware and is now regarded as the top mobile malware platform, overtaking the previous contender, Symbian [40]. Android malware has been successfully infecting many devices using various techniques that exploit the very characteristic of the Android platform. For example, the repackaging technique makes use of the fact that unknown APK file can be easily installed to Android application. The repackaging technique usually involves the malware author injecting or piggybacking malicious payloads to popular applications and redistribute it to alternative app market [43].

A study that tried to retrieve vulnerability data from many different Android devices concluded that almost all major Android brands are vulnerable to critical attacks [32]. The study also shows that the modifications made by different manufacturers to the original version of Android released by Google are causing the device to be more vulnerable to attacks. Additionally, the fact that patches and security updates are individually delivered by each of the phone's manufacturers also created a large gap of security level between each Android devices. This further proves that the open-source spirit of Android and the freedom to change the Android operating system to suit manufacturer's taste can have bad effects on user security too.

2.2 Limitations and Weakness of Android Security Model

The security of Android operating system and its app market model were analysed by a number of studies [31][26]. The followings are the summary of the limitations and

³<https://developer.android.com/studio/index.html>

⁴<https://play.google.com>

weaknesses found in them:

- App market

The Google Play store is not the only option for Android users when they want to download apps. In China, a ban on Google Play store to operate has been made official, resulting in the rise of numerous alternative app markets [20]. Some of the most popular ones are Baidu App Store, Tencent App Gem, and Qihoo 360 Mobile Assistant.

- App installation and execution

Installation of apps from unknown sources are made possible by the Android operating system. Users are allowed to install applications from APK files without any verification or active security checks on the app's behaviour and its file integrity [31].

- App permission model

Permission in Android dictates how application interact with the device's resources and the information contained within it. Some of the permissions in Android are declared by the Android SDK as dangerous permissions. These are the permissions that cover resources that are related to the user's private information, manipulation of the user's stored data. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app [37].

Prior to the release of Android 6.0 (API level 23), permissions are granted on an all-or-nothing basis. If a user wants to install an application, it has to accept all of the app's requested permissions. Otherwise, the installation is cancelled. After Android 6.0 is released, permissions are no longer granted during installation. They are granted to apps while the apps are running [2]. However, according to Google's data on the distribution of each version of Android, only 4.7% of active Android devices are currently running Android 6.0 or later. This shows that more than 95% of Android users are still granting permissions on an all-or-nothing basis.

In addition to this, the Android SDK allows an app to request access to resources that they don't actually need. This can result in third-party libraries abusing the granted permissions of their host apps. Another way that this can be exploited

is by making use of the Android's auto-update feature. After the user granted access to a benign application that has a bloated permission request, the app can update itself and added malicious code that can steal user's information.

- **Malware and reverse engineering of apps**

Android is the number one mobile device operating system in terms of the amount of malware that target it [40]. Malware can enter devices through many ways. The easiest way is by phishing the target to download a malware-injected app by sending them links to a fake market store. Malicious APK files could also be downloaded by the user through click-based advertising [31].

In addition to that, a new attack surface is also emerging from the use of JNI (Java Native Interface), which is supported by the Android Native Development Kit [1]. JNI allows apps to invoke native codes, which can cause more security issues if written with malicious intentions.

- **Security weaknesses and vulnerabilities**

Security flaws and vulnerabilities in Android are found in many layers [32]. Malicious attackers might try to exploit known vulnerabilities from the application installed by users, the manufacturer's pre-installed apps, the operating system layer, or the Linux layer such as the Linux kernel and its native libraries [39].

2.3 Previous Work on Android Security Metrics

In order to conduct a proper assessment to all of the above weaknesses, several attempts at analysing the security level and producing security metrics of Android devices has been conducted. We will discuss some of them in the following sections.

2.3.1 Security Metrics for the Android Ecosystem

A previous study on the security metrics for Android tried to give security scores on a scale of 10 to major brands of Android phone with regards to their vulnerability to attacks.

Figure 2.1 depicts the final security score produced by the study. The study proposed the use of a security metrics called FUM to produce the rankings in figure 2.1. The rankings correspond to the frequency of security patches delivered by the phone manufacturers and the number of vulnerabilities that the phones are exposed to.

Name	f	u	m	score (out of 10)
nexus	0.39 ± 0.00	0.48 ± 0.00	0.56 ± 0.01	5.17 ± 0.02
notnexus	0.10 ± 0.00	0.02 ± 0.00	0.53 ± 0.00	2.70 ± 0.00

Security scores for nexus

Name	f	u	m	score (out of 10)
LG	0.22 ± 0.00	0.33 ± 0.00	0.62 ± 0.01	3.97 ± 0.02
Motorola	0.18 ± 0.00	0.12 ± 0.00	0.71 ± 0.02	3.07 ± 0.02
Samsung	0.13 ± 0.00	0.04 ± 0.00	0.61 ± 0.00	2.75 ± 0.00
Sony	0.14 ± 0.00	0.19 ± 0.00	1.09 ± 0.02	2.63 ± 0.02
HTC	0.14 ± 0.00	0.10 ± 0.00	0.87 ± 0.01	2.63 ± 0.02
asus	0.20 ± 0.00	0.51 ± 0.01	6.01 ± 0.07	2.35 ± 0.02
other	0.06 ± 0.00	0.05 ± 0.00	1.04 ± 0.01	1.97 ± 0.02
alps	0.03 ± 0.00	0.19 ± 0.01	3.99 ± 0.08	0.80 ± 0.02
Symphony	0.00 ± 0.00	0.08 ± 0.00	5.00 ± 0.05	0.30 ± 0.01
walton	0.00 ± 0.00	0.09 ± 0.00	6.00 ± 0.08	0.27 ± 0.01

Security scores for manufacturers

Figure 2.1: The security level of Android devices

The FUM metric calculates security score using the following components:

1. Free, which represents the number of devices that are free from critical vulnerabilities
2. Update, which represents the number of devices that are currently running the latest version of Android supported by its manufacturer
3. Mean, which represents the average number of outstanding vulnerabilities found on the devices from the same manufacturer

The FUM metric gave 1 security score to a collection of devices. It does not produce an individual score for each identified phone. For example, we could see that in figure 2.1, LG has a security score of 3.97. According to the FUM score, this is the security score of all LG devices regardless of model number's, who owns the device, or how the device is used. The above study also gave security score to the Android ecosystem as a whole. Android as a whole scored 2.87 out of 10, which is alarmingly low.

The security metrics introduced by [32] is very effective in measuring the timely delivery of updates to fix critical vulnerabilities. It also produced a very useful security score for each major Android's brands and manufacturers and the Android ecosystem as a whole. However, these are overall scores. We could not say that the overall

security score given to the Samsung brand of Android is similar to the security score that should be given to a particular model of Samsung Android, held and used by an individual.

Every Android device is different. Devices may share the same model number, but the collection of apps installed within are obviously different depending on the users. Devices with the same model number could also differ in the type of operating system that it is running due to the choice that was usually given to users on whether or not they want to do a system upgrade. We believe that there is a need for a security metric that measures the unique security score of an individual device.

2.3.2 Malicious Application Classifier for Android Apps

In an attempt to assess the level of security in using a particular application, a lot of studies have been conducted to develop malware classifier that could detect the malicious and abnormal behaviour of Android applications.

We researched several different malware classifiers and identified the kind of features that they use as a metric to determine the level of maliciousness of an application:

1. Crowdroid, a classifier for Trojan malware, uses system calls as its target feature for analysis. The research found out that the system calls made by a genuine application are very different compared to the one made by their malware-infested counterpart [5].
2. ExplainDroid, a malware classifier that targets APK files. ExplainDroid uses permissions, API calls, and actions as its features. ExplainDroid also provides human-readable sentences that will describe the type of malware that resided within the APK file [7].
3. Kirin, extract function calls from the application's executables. This way, Kirin is able to detect malware before the application is installed [10].
4. Permission-induced Risk Malware Classifier, extracts permissions from a large number of both malicious and benign Android applications, employed three feature ranking methods (mutual information, correlation coefficient, and T-test), and came up with a ranking for individual permissions in Android according to their risk level [37].

5. Andromaly, uses machine learning approach to monitor sensor activities and CPU usages. It extracts features from the monitoring result in order to detect malicious activities in application [30].

All of the above malware classifiers have successfully assessed the malicious features of Android application and gave a verdict to the user of whether or not it is malicious. However, all of them runs on a desktop computer or at least requires the user to setup a testing environment before they can conduct their own assessment with it. End-users are becoming more aware of security and we believe that providing them with a security metric that is easy-to-use and easy-to-install will be beneficial in creating a more secure Android ecosystem.

Chapter 3

Requirement Specification

In this section, the concepts and requirements of Securimeter will be analysed and summarised to provide a comprehensive guideline for the design and implementation phase.

3.1 Requirement Elicitation

To grasp a full understanding of what is expected from an Android application that analyses the device and displays security score, we went through the requirement elicitation phase first. Aside from collecting the required functionality, requirement elicitation also helped in building the evaluation plan for this application. We explored a number of approaches in order to acquire the most comprehensive appreciation of what must be implemented within the application [8].

3.1.1 Research on Related Projects

For this project, we researched a number of similar projects and Android applications and take notes of the type of solution that they offer. These are applications that provide security scan support, including malware analysis, security reporting, and vulnerability testing. By highlighting their main features and by pointing out salient differences of their approach to mobile security scoring, this section also provides further evidence for the novelty of the concept of Securimeter.

3.1.1.1 Virus Total for Android

VirusTotal for Android is an Android application that scans the collection of apps in the device where it is installed. VirusTotal for Android performs a lookup service and check whether any of the installed apps are identified as virus.¹

VirusTotal for Android is actually a simplified version of a web application with the same name. The web application is a free malicious file detector that utilises several antivirus engines. Both the web application and Android app are owned by a subsidiary of Google [33].

VirusTotal for Android scans each installed apps, generate a hash value of each apps' APK file and then perform a hash lookup to its web service. VirusTotal displays its scan result in traffic light scheme as shown in 3.1. Apps that are regarded as malicious get a red colour mark, and apps that are considered as benign get a green colour mark. This use of traffic light colour scheme is a very good input for metrics-related apps as it has been proven to affect user's concern towards security [34].

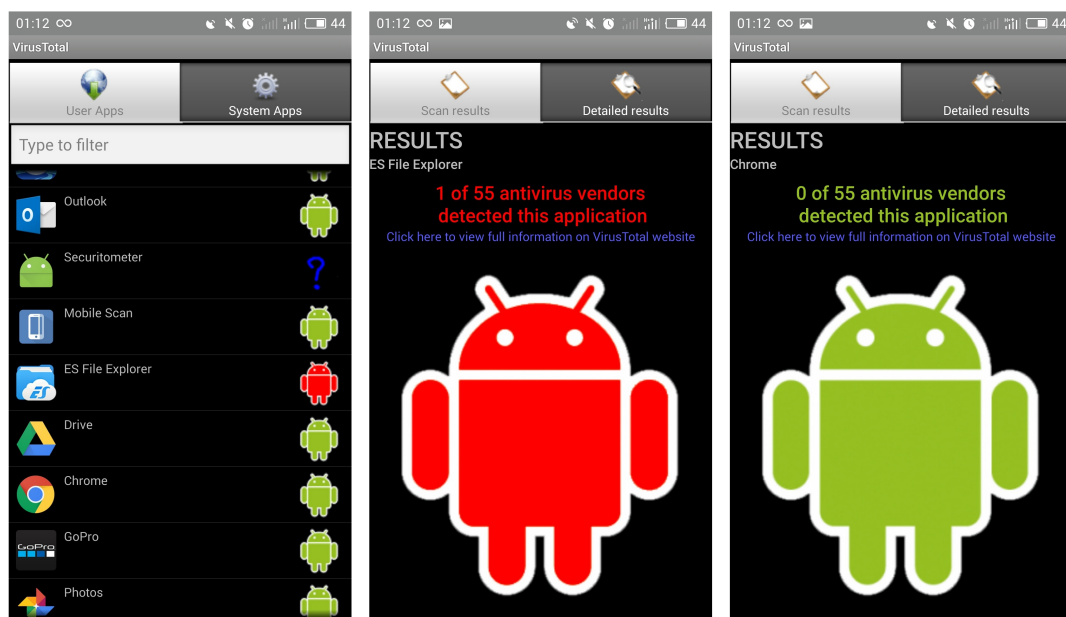


Figure 3.1: Traffic light scheme for the scanning results

If the web service does not recognise the hash value, users can choose to upload the file to VirusTotal web application so that it can perform an actual scan of the file. Since VirusTotal for Android only performs a lookup, it does not provide real-time protection.

¹<https://www.virustotal.com/en/documentation/mobile-applications/>

3.1.1.2 X-Ray

X-Ray is a mobile application, developed by Duo Security, that allows users to scan Android device for unpatched vulnerabilities that can be potentially exploited by malicious applications to gain root privileges or perform restricted actions on the device.²

X-Ray make use of the fact that a number of "privilege escalation" vulnerabilities has been present in the many releases of the original Android, and the fact that each customization attempt by manufacturers and carriers adds more to the list [27].

Once installed, X-Ray will retrieve information from the device regarding the type of system and software that are currently running. X-Ray then continue to test a number of vulnerability presence scans. When it is finished with the test, it will then display a summary of vulnerabilities that the device is exposed to, as seen in figure 3.2. X-Ray also displays its scan result in traffic light scheme.

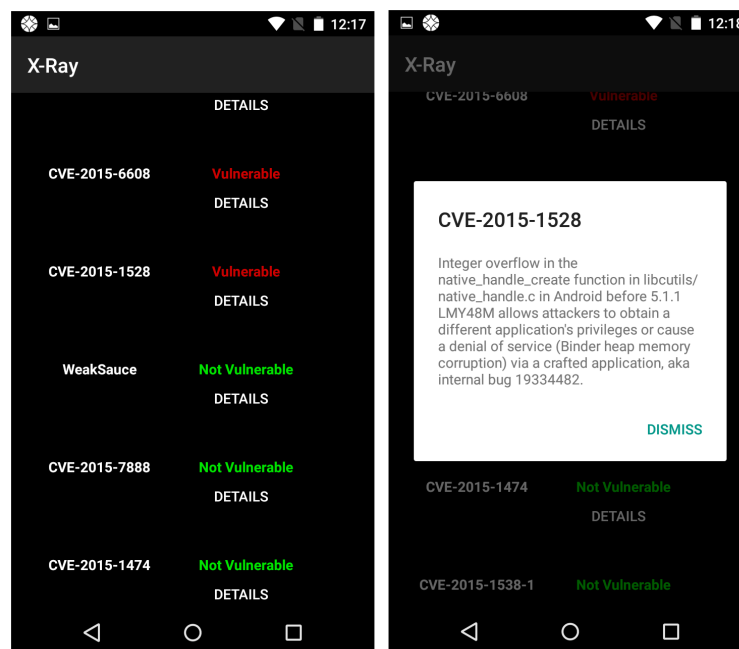


Figure 3.2: X-Ray displaying vulnerability test result

X-Ray used to work by actually attempting to exploit the vulnerabilities within the device. However, after their latest software update, X-Ray adopted the approach taken by Android VTS (will be explained in the next section), which is to safely probe for the presence of a vulnerability without ever exploiting it.

X-Ray's developers have made the project to be open-source and claimed that they

²<https://labs.duo.com/xray>

build the application with the average Android users as their target market. This is reflected in the way their user interface is designed [28].

3.1.1.3 Android VTS

Android VTS, almost very similar to the previous X-Ray, is a mobile application that allows users to scan Android device for unpatched vulnerabilities. Android VTS is developed by NowSecure Inc [21].

Android VTS's vulnerability test platform works the same way as X-Ray. It does not probe vulnerabilities by actually trying to exploit it, but just detected the presence of it. This is due to the fact that Android VTS is using the same vulnerability testing engine as X-Ray. Both X-Ray and Android VTS uses the same platform created by Ryan Welton, a Security Researcher who is also known under the name Fuzion24 [21][38]. The user interface is also very similar as seen in figure 3.3.

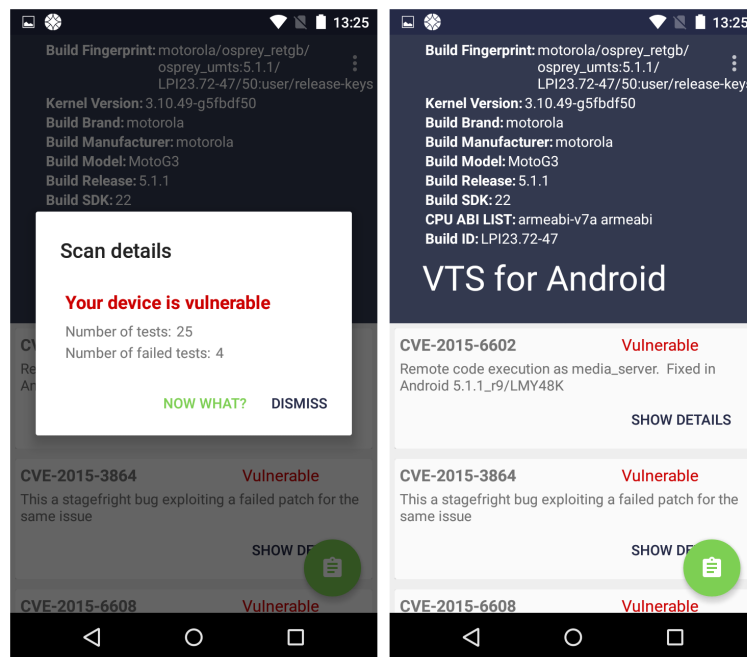


Figure 3.3: Android VTS displaying vulnerability test result

However contrary to X-Ray, Android VTS designed their application to target a more technical audience. The test results and information regarding vulnerabilities that they present to the user are more detailed and technical.

X-Ray was developed first at 2012, as an attempt to try to detect Android vulnerabilities by actually exploiting it. Android VTS came around 2015, introducing harness and removing checks that could cause instability to the device. In 2016, X-Ray adapted

Android VTS test platform and also remove the exploiting part from their test suite. Both X-Ray and Android VTS are open-sourced [23].

3.1.1.4 Risk-Ranking for Android Permissions

In an attempt to build a malware detection tool, an investigation of each Android individual permission was taken [37].

The original motivation behind this study was the fact that Android permission is very central in Android security. It also acts as an entry point in accessing private resources. The permissions requested by an app will depict the type of behaviour that it will display. It is the responsibility of both the user and the developers to fully understand the risk of granting and requesting for a combination of permissions.

In order to help user understand the risk of granting each permission, the study thoroughly analyze a number of both malicious and benign apps, employed three feature ranking methods (mutual information, correlation coefficient, and T-test), and come up with a ranking for individual permissions in Android according to their risk as seen in figure 3.4.

Rank	Corrcoef		Mutual Information	T-test
	Score	CorrCoef		
1	0.4428	READ_SMS	SEND_SMS	RESTART_PACKAGES
2	0.42	RECEIVE_SMS	RECEIVE_SMS	READ_SMS
3	0.3961	SEND_SMS	READ_SMS	READ_PHONE_STATE
4	0.1988	WRITE_SMS	READ_PHONE_STATE	SEND_SMS
5	0.1443	SET_ALARM	READ_EXTERNAL_STORAGE	READ_EXTERNAL_STORAGE
6	0.1403	RECEIVE_WAP_PUSH	WRITE_SMS	SYSTEM_ALERT_WINDOW
7	0.114	READ_PHONE_STATE	SET_ALARM	SET_ALARM
8	0.1044	READ_EXTERNAL_STORAGE	RECEIVE_BOOT_COMPLETED	RECEIVE_WAP_PUSH
9	0.0804	RESTART_PACKAGES	RECEIVE_WAP_PUSH	WRITE_SMS
10	0.0711	SYSTEM_ALERT_WINDOW	RESTART_PACKAGES	RECEIVE_SMS
11	0.0668	RECEIVE_BOOT_COMPLETED	WAKE_LOCK	RECEIVE_BOOT_COMPLETED
12	0.063	CHANGE_WIFI_STATE	ACCESS_NETWORK_STATE	CHANGE_WIFI_STATE
13	0.0611	WAKE_LOCK	WRITE_EXTERNAL_STORAGE	WAKE_LOCK
14	0.0562	DISABLE_KEYGUARD	INTERNET	DISABLE_KEYGUARD
15	0.0553	ACCESS_NETWORK_STATE	CHANGE_WIFI_STATE	ACCESS_NETWORK_STATE
16	0.0551	WRITE_SETTINGS	SYSTEM_ALERT_WINDOW	WRITE_SETTINGS
17	0.0535	READ_CONTACTS	READ_CONTACTS	READ_CONTACTS
18	0.053	RECEIVE_MMS	READ_CALL_LOG	RECEIVE_MMS
19	0.0506	WRITE_EXTERNAL_STORAGE	WRITE_SETTINGS	WRITE_EXTERNAL_STORAGE
20	0.045	EXPAND_STATUS_BAR	DISABLE_KEYGUARD	EXPAND_STATUS_BAR
21	0.0444	WRITE_CONTACTS	WRITE_CONTACTS	WRITE_CONTACTS
22	0.0415	CHANGE_NETWORK_STATE	CAMERA	CHANGE_NETWORK_STATE
23	0.0413	INTERNET	WRITE_CALL_LOG	INTERNET
24	0.0365	READ_HISTORY_BOOKMARKS	CHANGE_NETWORK_STATE	READ_HISTORY_BOOKMARKS
25	0.0346	CHANGE_CONFIGURATION	ACCESS_WIFI_STATE	CHANGE_CONFIGURATION
26	0.0344	PROCESS_OUTGOING_CALLS	RECEIVE_MMS	PROCESS_OUTGOING_CALLS
27	0.0339	GET_PACKAGE_SIZE	READ_HISTORY_BOOKMARKS	GET_PACKAGE_SIZE
28	0.0338	PERSISTENT_ACTIVITY	EXPAND_STATUS_BAR	PERSISTENT_ACTIVITY
29	0.0334	ACCESS_WIFI_STATE	GET_ACCOUNTS	ACCESS_WIFI_STATE
30	0.0329	READ_CALL_LOG	CHANGE_CONFIGURATION	READ_CALL_LOG
31	0.0309	CAMERA	PROCESS_OUTGOING_CALLS	CAMERA
32	0.0287	WRITE_HISTORY_BOOKMARKS	CALL_PHONE	WRITE_HISTORY_BOOKMARKS
33	0.0273	CALL_PHONE	WRITE_HISTORY_BOOKMARKS	CALL_PHONE
34	0.0252	SET_WALLPAPER_HINTS	GET_PACKAGE_SIZE	SET_WALLPAPER_HINTS
35	0.0249	GET_ACCOUNTS	GET_TASKS	GET_ACCOUNTS
36	0.0237	GET_TASKS	ACCESS_mock_LOCATION	GET_TASKS
37	0.0232	WRITE_CALL_LOG	PERSISTENT_ACTIVITY	WRITE_CALL_LOG
38	0.019	ADD_SYSTEM_SERVICE	ACCESS_FINE_LOCATION	ADD_SYSTEM_SERVICE
39	0.0182	ACCESS_FINE_LOCATION	SET_WALLPAPER_HINTS	ACCESS_FINE_LOCATION
40	0.0168	ACCESS_mock_LOCATION	USE_CREDENTIALS	ACCESS_mock_LOCATION

Figure 3.4: Ranking of permission according to their risk

The ranking made a very good sense and was further verified by an analysis of occurrence percentage of the top 40 riskiest permissions from the study as seen in figure 3.5.

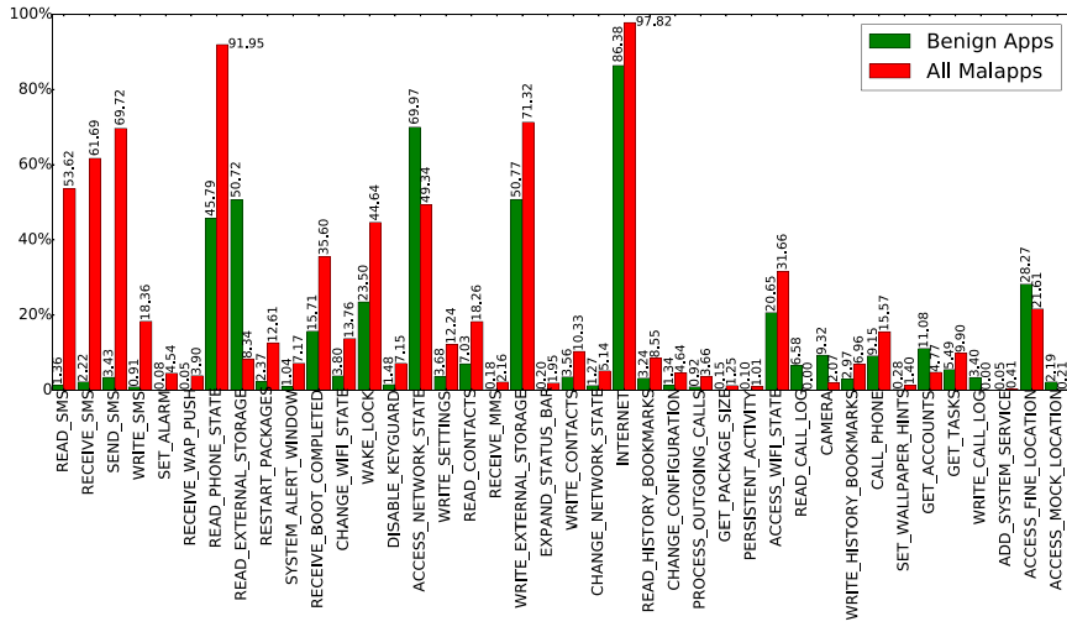


Figure 3.5: Occurrence percentage of top 40 Android permissions in malicious and benign apps

For example, permissions that are related to SMS, are always ranked at the top. This is because more than 50% of malicious apps requested this type of permission, and only about 10% or less benign apps requested for it. The huge difference of occurrence is supported by the report that stated about 68% of mobile threats are originated from premium service abuser and account data stealer. SMS-related activities are a huge contributor of these type of threats.

Another example is the fact that INTERNET, although considered as dangerous permission by Android Developers, is not ranked at the top. This is due to the fact that it is very common to find this permission in both malicious and benign apps.

3.1.1.5 Insights from Related Projects

After exploring a number of related apps and projects, we summarised the insights obtained from them to further motivate the development of Securitometer.

We could divide the topic around Android security into two main categories: operating system security, and application security. Interestingly, most related projects

seem to only focus on one area of security. X-Ray and Android VTS only focus on Android's vulnerability, VirusTotal only focuses on malicious file detection, while ExplainDroid and other research mentioned above only focus on malicious behaviour detection. In this matter, Securitometer tries to cover the scanning for both sides. Securitometer could adapt the same vulnerability test platform that is used by X-Ray and Android VTS, while at the same time employ the test for malicious application too.

In terms of target audience and the way the user interface was purposefully designed, the closest similarity found to the concept of Securitometer seems to be Android VTS because it is designed for a more technical audience. This will be a benchmark in designing the depth of information that needs to be displayed to the user regarding the test suite and the test results. Android VTS also has an export feature, which we deemed essential to be included in a security assessment tool.

3.1.2 Brainstorming

In conducting brainstorming, we mainly rely on the many subject experts around the School of Informatics, University of Edinburgh. Subject experts here include David Aspinall, this project's supervisor, the AppGuarden Group which is a research group embedded within the University of Edinburgh's Mobility and Security Group, and the weekly security and usability seminar group held by a usability expert.

The following points are captured from the brainstorming sessions:

1. Incorporating the Android malware classifier created by the AppGuarden Group into Securitometer

One of the key research in the AppGuarden group is an automatic Android malware classifier called ExplainDroid. ExplainDroid extracts the best-performing syntax-based features from an Android APK file such as permissions and API calls. It then uses several machine learning approaches to produce a classifier. The classifier's performance was compared against another type of malware classifier and it did show a dramatic improvement [7].

The ExplainDroid classifier obtained for this project was a bit more advanced than the one explained above. It takes an APK file as input, gives a verdict of whether the app is malicious or benign, and if it is indeed malicious, it also generates an explanation of the reason why it is malicious.

2. Taking user's safety into account

From the usability point of view, a lot of inputs about user's safety are generated during the brainstorming session. This includes the effort to make sure that the user understands what Securitometer does to the device and whether it will cause harm or not to the device. It was concluded that this type of explanation can be inserted in the following areas within the app:

- Introductory pages, prior to app usage
- Main page, prior to starting the test
- Result page, in a more detailed manner

3. Development and evaluation plan

We decided that the development of Securitometer should go through a number of versions. For each version, we inquire a number of potential users to evaluate its usability and acceptability. The next version of Securitometer is usually an improved version based on the inputs gathered from the previous version's evaluation.

We also decided to launch Post-Study System Usability Questionnaire (PSSUQ) for the final version. The PSSUQ measured the followings:

- Overall user satisfaction with the application
- System usefulness
- Information quality
- Interface quality

3.1.3 Prototyping and Survey

Prototyping is less useful for identifying initial requirements [8]. Because of that, we chose to conduct research and brainstorming for the initial phase of requirement elicitation. After that, we build an initial prototype, conducted a survey on it, and collect further requirement. The survey collected 60 respondents³. About 40% of them are penetration tester and security auditors, 41% are IT employee, and the rest are computer science students and security researcher.

Respondents are recruited by utilising the Securitometer developer's list of colleagues and connection with several penetration testers and IT security auditors from

³<https://goo.gl/forms/l0yoavQkc16t6n9r1>

professional service firms. The questionnaire is created using Google Forms and is available online. The survey was designed using Likert-scale and Rating-scale and was analysed accordingly [35][36].

The followings are the requirements gathered from prototyping and the survey after that:

- About 81.7% of the respondents agreed that an app that requested more dangerous permissions have more potential to be problematic in the future, compared to the one that requested less dangerous permissions.

We noted that we could use permissions ranking to assign a risk level to installed applications. In support of this, we obtained a ranking of Android permission based on a study that compares its percentage of occurrence in malicious and benign Android apps [37].

- About 88.3% of the respondents agreed that in displaying the scanning result, apps that are more likely to be problematic in the future should be displayed above the other apps. We noted that this insight should be incorporated in the way we organise our test results.
- About 95% of the respondents agreed that using traffic colour scheme to display the score helps the user understand the impact that each app has on the security level of the device. We noted that this insight should be incorporated in the way we organise our user interface.
- We asked the users to rank the factors that they believe are more important in determining Android security level. We got the following ranking:

1. Malicious app identification (score: 8.8 out of 10)
2. OS vulnerability check (score: 8.7 out of 10)
3. App permission check (score: 8.6 out of 10)
4. Checking whether phones are encrypted (score: 7.2 out of 10)
5. Checking whether passcode is used (score: 6.8 out of 10)

The last two factors represent configuration as one of the factors that determine the security level of a device. The top three factors are mostly about the collection of applications installed on the device and operating system. Supported by

the survey result, we decided to only use the top three-factor and not use configuration as our metrics due to time constraint. Further survey is required if Securitometer is to incorporate configuration to its metrics too.

- We also asked the users which type of vulnerability scanning app that they prefer. We got the following:

1. Stable vulnerability check (55%)

This type of app attempts to detect the presence of vulnerabilities without actually exploiting it, taking care to not include checks that could cause instability problems for the end user and therefore may omit checks that could cause these types of issues.

2. Exploit attempt (25%)

This type of app actually attempts to exploit the vulnerabilities which could cause instability to the device being scanned.

3. Lookup based (20%)

This type of app uses the device version/build information to look up common vulnerabilities found in that Android version. No actual vulnerability scan takes place.

We noted that stable vulnerability check are most popular both in the survey and in real world application, as seen in Android VTS and X-Ray application before.

More details about the result of this survey are provided in section 5.2.2.1.

3.1.4 Focus Group Discussion

Focus group captures requirements by gathering a number of representatives for the potential users and clients [8]. These representatives should have a broad idea regarding user's needs. We conducted focus group discussion with the School of Informatics weekly security and usability seminar group held by a usability expert. The group consists of usability experts and a number of MSc and UG4 researcher who are currently studying or have studied computer security as seen in figure 3.6. Our justification for this is that these people have the technical knowledge in both security and user interface design. They are also the people who will eventually work as security practitioner so that they represent the potential user of Securitometer.

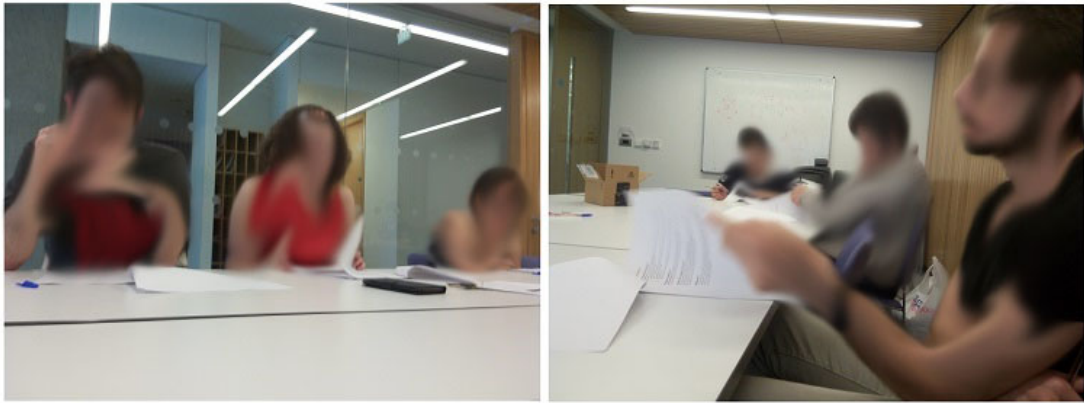


Figure 3.6: Focus group for Securitometer in progress (blurred for privacy)

Before conducting the focus group discussion, we have developed the first prototype of Securitometer. We asked them to try the first prototype, and then we handed the questionnaire that will guide the discussion process.

A lot of inputs are accepted during the discussion, however, we collected them all into groups and made the following summary of the focus group discussion:

- Since Securitometer's target users are from the technical audience, more information regarding the type of tests that are being conducted are needed.
- Introductory pages are needed to make sure that the user understands what the application do.
- Before testing the device, more information about the device itself is needed for reporting purpose.
- Result sharing is not necessarily useful for this type of app since it is only relevant if the target user is the general Android users. It also raises user's privacy concern.
- The test result should provide a summary of what caused the biggest contribution to the security score.
- The traffic colour scheme used to display security score and test result font should take into account the background colour.
- When the tests are running, a dialogue that informs users about the current progress of the test should be displayed.

- Consider providing the feature that let users run the test while completing another task.

3.2 Recap and Summary

The requirement elicitation activities that we conducted is not completely detached from each other. An activity is conducted over the insights gained from the earlier one. This was done in order to justify the novelty of Securitometer, both from the point of view of practitioners as represented by penetration tester and security professionals, as well as academicians as represented by security researcher and students.

We summarised the insights collected into the following list of requirements [14]:

1. Incorporate device vulnerability test suite

Elicited from: research, prototyping, survey

Success criteria:

- (a) The test detects vulnerabilities present in Android devices.
- (b) The test works on different CPUs.
- (c) The test does not cause instability to target device.

2. Incorporate malware classifier

Elicited from: research, brainstorming, prototyping, survey

Success criteria:

- (a) The classifier detects potential malware in target device's installed apps.
- (b) The classifier runs on the device without communicating any data to third party server.

3. Incorporate app permission test

Elicited from: research, prototyping, survey

Success criteria:

- (a) The test identifies permissions that are commonly found in a malicious application.
- (b) The test assigns a risk level to each app based on the type of permissions that it requested.

- (c) The test labels an app based on its risk into the following categories: high risk, medium risk, and low risk.

4. Include introductory pages

Elicited from: prototyping, brainstorming, survey, focus group discussion

Success criteria:

- (a) The introductory pages are shown on Securitometer's first run.
- (b) The introductory pages give an elaborate explanation to the user about what Securitometer is doing.
- (c) The introductory pages ensure users that Securitometer is not harmful to the target device.

5. OS scoring mechanism

Elicited from: prototyping, survey, focus group discussion

Success criteria:

- (a) OS score will be contributed by the number of vulnerabilities found in the device.
- (b) OS score will be displayed using the colour scheme of the traffic light.
- (c) Test that shows the device as being vulnerable, should be displayed first on the application screen.

6. App scoring mechanism

Elicited from: prototyping, survey, focus group discussion

Success criteria:

- (a) App score will be contributed by the number of malware found in the device and the risk level of the app.
- (b) App score will be displayed using the colour scheme of the traffic light.
- (c) Apps that are more likely to be problematic should be displayed first on the application screen.

7. Report mechanism

Elicited from: prototyping, brainstorming, focus group discussion

Success criteria:

- (a) Securimeter should be able to produce a report about the target device with an interchangeable format.
- (b) The report should include the app permission test result, the malware classifier result, and the vulnerability test result.

8. Progress dialogue

Elicited from: focus group discussion

Success criteria:

- (a) Securimeter should inform users of its current progress when tests are running.
- (b) User should be notified when test results are ready.
- (c) Notifications should be handled in a way that will not impose a bad user experience.

The above list of requirements will be revisited in section 5.1 as one of the ways to evaluate this project.

Chapter 4

Design and Implementation

In this chapter, we will discuss the development and implementation of Securitometer. This will consist of the 3 type of tests executed by Securitometer, the 2 scoring mechanisms displayed to the user, the user interface and other additional features developed in order to satisfy the requirements elicited in Chapter 3.

4.1 Technology Specification

4.1.1 Minimum Target Platform

One of the earliest design decision that we have to make was determining the minimum target platform where Securitometer could run. We decided that Securitometer should at least be supported on any devices running API version 16 or later of the Android SDK (Android 4.1 Jelly Bean). This should cover at least 95.2% of the devices that are currently active at the moment¹. This decision is also supported by the Google Android Developer Guide which stated that choosing to support about 90% of the active devices is good practice [2].

4.1.2 Solution Architecture

The Securitometer application consists of several components. Figure 4.1 depicts the many elements that make up the whole Securitometer application.

Securitometer runs three type of test: the malware classification test, app permission test, and device vulnerability test. These three tests are then compiled into two

¹<https://developer.android.com/studio/projects/create-project.html>

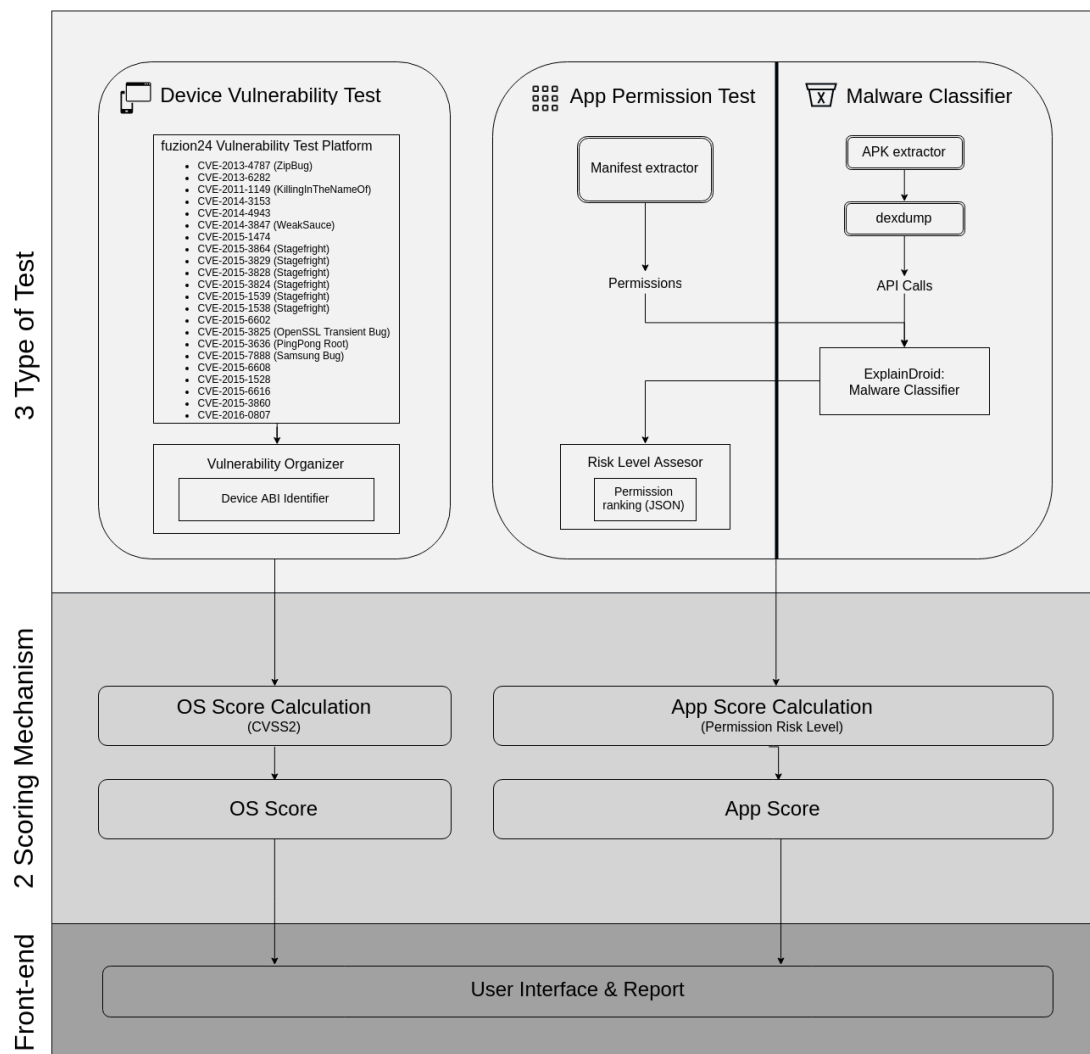


Figure 4.1: Securitometer components

scoring mechanisms, 1 score for the operating system and the other one for the collection of apps. The device vulnerability test contributes to the score of the operating system. The app permission test and malware classifier contribute to the score of the collection of apps.

The front-end of Securitometer was defined using standard Android layout mechanism which is a collection of XML files. Based on the requirement elicitation activities that we have elaborated in Chapter 3, we decided that the following pages are necessary: splash screen, introductory pages, main page before the test, overall test result page, app test result page, device vulnerability test result page, and export result. We will provide details about each component in the coming sections.

4.2 Back-end: Three Tests Performed

4.2.1 Malware Classification Test

Securitometer incorporated a malware classifier called ExplainDroid, within it [7]. ExplainDroid's malware classifier decide whether or not an APK file is malicious or benign. ExplainDroid was originally implemented in Python. We ported ExplainDroid to Java for Android and put the resulting code inside Securitometer. Hereafter, we will refer to the ported classifier as Securitometer's malware classifier.

Securitometer's malware classifier extracts the permissions and API calls from APK files, and then uses them as the feature for classification. These two are selected as features because they reflect both the requirement for resources and the ability of applications to access those resources.

The followings are the steps taken by Securitometer's malware classifier:

1. Extract permissions and API calls from each app to use as input feature for the classification problem
2. Sum the weight of each identified features
3. Decide whether an app is malicious or benign based on the sum result

The original version of ExplainDroid's malware classifier uses two Linux binaries called *dexdump* and *aapt* to extract the required input features, API calls and permission request. For Securitometer's malware classifier, we managed to simulate the extraction of those features using the Android SDK only.

Details on how we port the malware classifier will be provided in the coming sections.

4.2.1.1 Extracting API Calls

We simplified the code used for API calls extraction from the original ExplainDroid into the following:

```
def collect_api_calls(dexdump, files) :  
    apis = []  
  
    temp_file = "./stdout.txt"  
  
    re_f_invoke = r"invoke-"
```

```

re_api = r"""(Landroid|Ljava|Ljavax|Ldalvik|Ljunit|
            Lcom\|android|Lorg\| (apache|json|w3c|xml|xmlpull))
            [a-zA-Z0-9\$\-_\|\/\;\.\>\<]+:"""

for file in files:
    # using dexdump to get apis
    try :
        out = open(temp_file, "w")
        subprocess.call([dexdump, "-d", file], stdout = out)
        out.close()
    except:
        print "Failure in execution: dexdump!"
        sys.exit(0)

    out = open(temp_file, "r")

    for line in out :
        s = line.strip()
        if re.search(re_f_invoke, s) :
            m = re.search(re_api, s)
            if m :
                api = m.group()
                api = re.sub("\|", ".", api)
                api = re.sub("L", "", api)
                api = re.sub(";", "", api)
                api = re.sub(":", "", api)
                if not api in apis :
                    apis.append(api)

    out.close()

return [apis]

```

Listing 1: API calls extraction in the original ExplainDroid's malware classifier

In the original code, the APK file was fed to ExplainDroid via command line. ExplainDroid then executes *dexdump* on the APK file and parse the output as needed to build a list of API calls. ExplainDroid does not run on mobile devices, so the APK used as input has to be copied to the same machine as ExplainDroid. This makes feature extraction a lot easier because ExplainDroid and the APK file stands on a different level. ExplainDroid act as a running application and the APK file is just a static data file acting as input.

The followings are the steps defined within Securitometer to simulate *dexdump* execution in ExplainDroid:

1. Extract APK files from installed apps

The challenge here is that Securitometer targets installed apps on a device, while Securitometer itself is also an installed app. Both Securitometer and the target apps are standing on equal level, as an application running on the same device. To tackle this issue, we use a class called *PackageManager*, provided by the Android SDK². *PackageManager* assisted us in transforming the installed app from a running application into a static APK file. The classifier did not require an app to be in the state of running while it extracts their features. This makes it easier to port this functionality to Securitometer.

Securitometer scan through all installed apps using the *PackageManager* class, extract their source information, and write each of them one by one as APK file to the local storage. To be able to do this, Securitometer requires the following permission during installation: `android.permission.READ_EXTERNAL_STORAGE` and `android.permission.WRITE_EXTERNAL_STORAGE`.

The following simplified code depicts how Securitometer extracts the APK files of all installed apps:

```

ApplicationInfo apkInfo;
PackageManager manager;

//EXTRACT APK FILE
File f1 = new File(apkInfo.sourceDir);
String file_name = apkInfo.loadLabel(pm).toString();
file_name = file_name.replace(" ", "_");
String f2Path = (Context.getExternalFilesDir(Environment.
    getDataDirectory()).getAbsolutePath()).getAbsolutePath();
File f2 = new File(f2Path + "/" + file_name + ".apk");
f2.createNewFile();
//ACTUAL WRITING
InputStream in = new FileInputStream(f1);
OutputStream out = new FileOutputStream(f2);
byte[] buf = new byte[4096];
int len;
float total = 0;

```

²<https://developer.android.com/reference/android/content/pm/PackageManager.html>

```

while ((len = in.read(buf)) > 0) {
    total += len;
    out.write(buf, 0, len);
}
in.close();
out.close();

//DO SOMETHING WITH THE APK HERE
//.....

```

Listing 2: Extract APK files of all installed apps

2. Removing Android system apps from the scan

Android SDK's *PackageManager* also included system applications in the result when it scans through all installed apps in the device. System apps are written in a limited-access location (root only) to prevent deletions and are often very critical in order to keep the operating system running. Due to this reason, we decided to exclude them from our tests. We also excluded Securitometer from scanning itself. The following simplified code assisted us in filtering the target apps:

```

ApplicationInfo apkInfo;
PackageManager manager;

if ((manager.getApplicationInfo(apkInfo.packageName, 0).flags &
    ApplicationInfo.FLAG_SYSTEM) != 0)
{
    continue;
}
if (apkInfo.packageName.contains("com.mrezqi.android.securitometer"))
{
    continue;
}

```

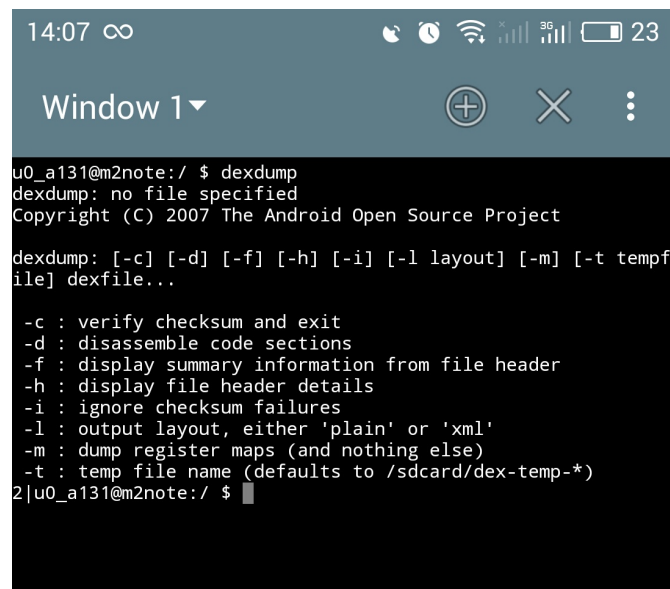
Listing 3: Excluding system apps and Securitometer itself

3. Executing *dexdump* binary inside Android device

Using another application called *Terminal Emulator for Android*³, we found out

³<https://play.google.com/store/apps/details?id=jackpal.androidterm&hl=en>

that *dexdump* is actually executable within the Android device as seen in figure 4.2.



```

14:07 ∞
Window 1
u0_a131@m2note:/ $ dexdump
dexdump: no file specified
Copyright (C) 2007 The Android Open Source Project

dexdump: [-c] [-d] [-f] [-h] [-i] [-l layout] [-m] [-t tempf
ile] dexfile...

-c : verify checksum and exit
-d : disassemble code sections
-f : display summary information from file header
-h : display file header details
-i : ignore checksum failures
-l : output layout, either 'plain' or 'xml'
-m : dump register maps (and nothing else)
-t : temp file name (defaults to /sdcard/dex-temp-*)
2|u0_a131@m2note:/ $

```

Figure 4.2: dexdump ran in Android

ExplainDroid provides its user with two type of *dexdump*: the Linux version, and the Mac version. We compared the output of Android's version of *dexdump* with that of ExplainDroid's. We installed *Terminal Emulator for Android* on each version of Android, started from Android 4.1 JellyBean (API version 16) which is our minimum target device. All of them run the same version of *dexdump* with ExplainDroid. All of the outputs are also exactly the same.

Securitometer executes *dexdump* by utilizing the Process class in Java for Android⁴. The challenge here is that dexdump output is incredibly large and it took away a lot of resources when Securitometer is trying to parse its output using the regular expression pattern matching. The parsing of API calls is a very critical phase in ExplainDroid's feature extraction, so we decided not to tamper with the amount of result that the parser collects. Instead, we focus on building an acceptable experience for the users when they are waiting for Securitometer to parse.

The following is the simplified version of Securitometer's ported code in executing dexdump, written in Java for Android:

```
HashMap<String, Integer> apk_features;
```

⁴<https://developer.android.com/reference/android/os/Process.html>

```

//DEXDUMP
Process p = null;
try {
    //EXECUTE DEXDUMP
    p = Runtime.getRuntime().exec("dexdump -d " +
        + f2.getAbsolutePath());

    //PARSE DEXDUMP OUTPUT WITHOUT WRITING IT TO FILE
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(p.getInputStream()));

    String line = reader.readLine();
    while (line != null) {
        String re_f_invoke = "invoke-";
        Pattern jav_re_f_invoke = Pattern.compile(re_f_invoke);
        String re_api = "(Landroid|Ljava|Ljavax|Ldalvik|Ljunit| " +
            "Lcom/android|Lorg/(apache|json|w3c|xml| " +
            "xmllpull))[a-zA-Z0-9$_/;>.<]+:";
        Pattern jav_re_api = Pattern.compile(re_api);

        try {
            line = line.trim();
            Matcher m_invoke = jav_re_f_invoke.matcher(line);
            if (m_invoke.find()) {
                Matcher m_api = jav_re_api.matcher(line);
                if (m_api.find()) {
                    String api = m_api.group();
                    api = api.replaceAll("/", ".");
                    api = api.replaceAll("L", "");
                    api = api.replaceAll(";", "");
                    api = api.replaceAll(":", "");

                    //NEW API CALLS FOUND
                    if (!apk_features.containsKey(api)) {
                        apk_features.put(api, 1);

                        //UPDATE NOTIFICATION MESSAGE HERE
                        //.....
                    }
                }
            }
        }
    }
}

```

```

    }
    catch (Exception e) {
        e.printStackTrace();
    }
    line = reader.readLine();
}
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (p != null) {
        p.destroy();
    }
}
}

```

Listing 4: Executing *dexdump* in Securitometer

Notice that in the code above, we put in a notification feature for the execution of *dexdump*. This is because parsing *dexdump*'s output could take a long time, and we don't want to prevent users from using their phone for that long just to complete a security scan. By providing a notification service that contains the progress of *dexdump*'s execution, users can hide the Securitometer in the background and use other apps and be notified of whether or not Securitometer has completed its scan yet. This will be discussed further in the user interface sections.

4. Delete APK file

Extracting APK files could take a huge amount space from the target device's storage. We took care to make sure that after we are done extracting features from one app, we deleted its extracted APK file to prepare more storage for the next one.

4.2.1.2 Extracting Permissions

We simplified the code used for permission request extraction from the original ExplainDroid into the following:

```

def collect_permissions(aapt, files) :
    perms = []
    pkg = ""

```

```

temp_file = "./stdout.txt"

re_f_perm = r"E: uses-permission"
re_pkg = r"A: package=\"([a-zA-Z\.\0-9\_-\$]+)\""
re_perm = r"(android\.permission\.)([A-Z\_]+)"

for file in files:
    # using aapt to get permssions and actions
    try :
        out = open(temp_file, "w")
        subprocess.call([aapt, "l", "-a", file], stdout = out)
        out.close()
    except :
        print "Failure in execution: aapt!"
        sys.exit(0)

    out = open(temp_file, "r")

    # find an "E: "
    f_E = 0

    for line in out :
        s = line.strip()
        if re.search(re_f_perm, s) :
            f_E = 1
            continue
        elif f_E == 1:
            f_E = 0
            m = re.search(re_perm, s)
            if m :
                perm = m.group()
                if not perm in perms :
                    perms.append(perm)
            elif re.search(re_pkg, s):
                m = re.search(re_pkg, s)
                pkg = m.group(1)
    out.close()

return [perms, pkg]

```

Listing 5: Permission extraction in the original ExplainDroid's malware classifier

Similar to the *dexdump* execution, the original ExplainDroid was fed APK files via command line and then it executes *aapt*. The output of *aapt* execution against the APK file is then parsed as needed to build a list of permission request.

Many steps that are taken for the API extraction part are almost similar with the steps for permission extraction. In order to save computing resources, we wrote the API extraction code and the permission extraction code within the same loop. However, for the sake of clarity, we described them as a separate entity. The followings are the steps defined within Securitometer to simulate *aapt* execution in ExplainDroid:

1. Removing Android system apps from the scan

This part was already explained in the API extraction part.

2. Extracting permission request

Extracting permission request is a lot easier to simulate within an Android application compared to *dexdump*, because of the *PackageManager* class. *PackageManager* provides a way to scan through a list of installed apps within the device, including the list of permissions requested by each app. Due to this, the process usually completes in a very little time. To make sure that the output of ExplainDroid's execution of *aapt* and Securitometer's output from *PackageManager* class are the same, we compared the output from executing both of them for 10 different apps. All of the outputs are also exactly the same.

The following is the simplified version of Securitometer's ported code in extracting installed apps' requested permissions, written in Java for Android:

```
HashMap<String, Integer> apk_features;
PackageInfo permissionInfo;

//AAPT
for (String perm : permissionInfo.requestedPermissions) {
    //NEW PERMISSION FOUND
    if (!apk_features.containsKey(perm)) {
        apk_features.put(perm, 1);
    }
}
```

Listing 6: Extracting permissions in Securitometer

4.2.1.3 The Classification

Originally, ExplainDroid is a Python-based tool which classifies whether or not an APK file is malicious or not. ExplainDroid also attempts to improve peoples understanding of potential threats hidden in APK file by presenting a sentence that will describe the type of malware that resided within the application. ExplainDroid consists of a classifier, a feature selector, a keyword selector, and a sentence selector [7]. For the development of Securitometer, we only ported the classifier component of ExplainDroid from Python to Java for Android. This is because Android SDK does not yet support the ability to filter intents from the device's installed app which is required for the keyword and sentence selector components.⁵

The following steps are taken in order to port the classification step to Java:

1. Features extraction (API calls and permissions)

This part is already explained in the previous section.

2. Importing the classification data to Securitometer

We decided to include ExplainDroid's classification data into Securitometer's asset files. These data are represented in a text file containing the name of the selected feature and its corresponding weight. When Securitometer runs, it will proceed to read these files and save its values into a *hashmap* for fast retrieval using the feature name as the key.

3. Determining whether an app is malicious or benign

ExplainDroid's malware classifier produced its classification data by assigning weights to the features extracted from its training phase. Figure 4.3 depicts a sample combination of these feature name and their corresponding weights:

android.permission.SEND_SMS	-1.887638693411285
android.telephony.gsm.SmsManager.sendTextMessage	-0.8914495819676127
android.hardware.Camera\$Parameters.setWhiteBalance	0.4244076954229288
android.permission.SET_PREFERRED_APPLICATIONS	0.5273547627688526

Figure 4.3: Feature name and its corresponding weight

The weight here represents the malicious quality of the feature intuitively. A negative weight indicates a suspicious functionality from the usage of the feature in the wild while a positive weight indicates a normal functionality.

⁵<https://code.google.com/p/android/issues/detail?id=3217>

We ported the exact code of calculating the malicious value of an app from ExplainDroid to Securitometer. We summed the weight of each extracted features and regarded that as the app's value. If the sum is negative, then we classified the app as malicious. If the sum is zero or positive then we classified the app as benign.

To verify that we have incorporated ExplainDroid into Securitometer, we tested it against the samples apps that were given by the original creator of ExplainDroid. Figure 4.4 shows the result of the original ExplainDroid's malware classifier:

```
mure@tmpn:~/explaindroid$ ./explaindroid -mod linux -apk samples/app.apk
APP: jp.naver.linecamera.android
LABEL: BENIGN
mure@tmpn:~/explaindroid$ ./explaindroid -mod linux -apk samples/demo.apk
APP: com.example.demo
LABEL: BENIGN
mure@tmpn:~/explaindroid$ ./explaindroid -mod linux -apk samples/sample.apk
APP: com.example.myfirstapp
LABEL: BENIGN
mure@tmpn:~/explaindroid$ ./explaindroid -mod linux -apk samples/flash.apk
APP: goldenshorestechnologies.brightestflashlight.free
LABEL: MALWARE
```

Figure 4.4: ExplainDroid's malware classifier result

Figure 4.5 shows the result of Securitometer's malware classifier. Securitometer's malware classifier is exactly the same as that of ExplainDroid, except that it is wholly written in Java for Android instead of Python:

<p>Brightest Flashlight Free MALWARE</p> <p>Classified as: MALWARE</p> <p>This app requested 9 out of the top 40 permissions that are most commonly exploited by malicious apps.</p> <p>Security Score: 0.00 out of 10</p> <p>SHOW DETAILS</p>	<p>demo LOW RISK</p> <p>Classified as: BENIGN</p> <p>This app did not request any permission from the top 40 most commonly exploited permissions by malicious apps.</p> <p>Security Score: 10.00 out of 10</p> <p>SHOW DETAILS</p>
<p>LINE camera LOW RISK</p> <p>Classified as: BENIGN</p> <p>This app requested 13 out of the top 40 permissions that are most commonly exploited by malicious apps.</p> <p>Security Score: 7.94 out of 10</p> <p>SHOW DETAILS</p>	<p>MyFirstApp LOW RISK</p> <p>Classified as: BENIGN</p> <p>This app did not request any permission from the top 40 most commonly exploited permissions by malicious apps.</p> <p>Security Score: 10.00 out of 10</p> <p>SHOW DETAILS</p>

Figure 4.5: Securitometer's malware classifier result

4.2.2 Application Permission Test

This test's objective is to assign a risk level to each installed apps. The risk-level are based on the type and number of permissions that it requested. Securitometer utilised a ranking of permissions obtained from a research that produces the top-40 riskiest permissions commonly found in malicious applications.

We incorporated the ranking of permission in a .JSON file as depicted below:

```
{
  "android.permission.READ_SMS": [0.4428, 1],
  "android.permission.RECEIVE_SMS": [0.4200, 2],
  "android.permission.SEND_SMS": [0.3961, 3],
  "android.permission.WRITE_SMS": [0.1988, 4],
  "com.android.alarm.permission.SET_ALARM": [0.1443, 5],
  "android.permission.RECEIVE_WAP_PUSH": [0.1403, 6],
  "android.permission.READ_PHONE_STATE": [0.1140, 7],
  "android.permission.READ_EXTERNAL_STORAGE": [0.1044, 8],
  "android.permission.RESTART_PACKAGES": [0.0804, 9],
  "android.permission.SYSTEM_ALERT_WINDOW": [0.0711, 10],
  "android.permission.RECEIVE_BOOT_COMPLETED": [0.0668, 11],
  "android.permission.CHANGE_WIFI_STATE": [0.0630, 12],
  "android.permission.WAKE_LOCK": [0.0611, 13],
  "android.permission.DISABLE_KEYGUARD": [0.0562, 14],
  "android.permission.ACCESS_NETWORK_STATE": [0.0553, 15],
  "android.permission.WRITE_SETTINGS": [0.0551, 16],
  "android.permission.READ_CONTACTS": [0.0535, 17],
  "android.permission.RECEIVE_MMS": [0.0530, 18],
  "android.permission.WRITE_EXTERNAL_STORAGE": [0.0506, 19],
  "android.permission.EXPAND_STATUS_BAR": [0.0450, 20],
  "android.permission.WRITE_CONTACTS": [0.0444, 21],
  "android.permission.CHANGE_NETWORK_STATE": [0.0415, 22],
  "android.permission.INTERNET": [0.0413, 23],
  "com.android.browser.permission.READ_HISTORY_BOOKMARK": [0.0365, 24],
  "android.permission.CHANGE_CONFIGURATION": [0.0346, 25],
  "android.permission.PROCESS_OUTGOING_CALLS": [0.0344, 26],
  "android.permission.GET_PACKAGE_SIZE": [0.0339, 27],
  "android.permission.PERSISTENT_ACTIVITY": [0.0338, 28],
  "android.permission.ACCESS_WIFI_STATE": [0.0334, 29],
  "android.permission.READ_CALL_LOG": [0.0329, 30],
  "android.permission.CAMERA": [0.0309, 31],
  "com.android.browser.permission.WRITE_HISTORY_BOOKMARK": [0.0287, 32],
  "android.permission.CALL_PHONE": [0.0273, 33],
```

```
"android.permission.SET_WALLPAPER_HINTS": [0.0252, 34],
"android.permission.GET_ACCOUNTS": [0.0249, 35],
"android.permission.GET_TASKS": [0.0237, 36],
"android.permission.WRITE_CALL_LOG": [0.0232, 37],
"android.permission.ADD_SYSTEM_SERVICE": [0.0190, 38],
"android.permission.ACCESS_FINE_LOCATION": [0.0182, 39],
"android.permission.ACCESS_MOCK_LOCATION": [0.0168, 40]
}
```

Listing 7: Ranking of permission in JSON

The key in the JSON file represents the name of the requested permission. The first value represents the risk score that users are exposed to when they granted access to that permission. The higher the risk score, the higher the risk of granting access to that permission. The second value represents the permission ranking. Permissions that are ranked on top (low number) are the riskiest.

The application permission test makes use of the list of permissions extracted from the previous test. If the test found a match between an app's requested permissions and the above ranking of permissions, it will add the value of that permission's risk score to the calculation of that app's security score.

Both the malware classifier and the application permission test results are calculated as a single score, which is the app's score. This is because both of them have the same component for their test target, the collection of installed apps on the device. Details about the score calculation are provided in the next scoring mechanism section.

4.2.3 Device Vulnerability Test

Securitometer's vulnerability test suite is implemented using the same test platform as that of Android testS and X-Ray. The test platform was developed by Ryan Welton, a Security Researcher who is also known under the name Fuzion24 [21][38]. The test platform contains a number of vulnerability detection tests written in Java for Android and C which can be compiled using Android NDK (Native Development Kit). Android NDK allows developers to use C and C++ code in their application. The official NDK guide recommended the use of NDK to developers only if they really need to squeeze out extra performance for computationally intensive apps. It also recommends NDK if developers need to reuse other developers libraries built in C or C++.

The vulnerability test suite by Fuzion24 provided Securitometer with a total of 22

different tests as seen in Table 4.1, each with their own proof of concept:

No	Popular name	CVE ID	Category	CVSS2
1	ZipBug	CVE-2013-4787	Zip bug	6.1
2	put_user/get_user	CVE-2013-6282	Kernel bug	7.2
3	KillingInTheNameOf	CVE-2011-1149	Kernel bug	7.2
4	Towelroot	CVE-2014-3153	Kernel bug	7.2
5	Linux L2TP Socket	CVE-2014-4943	Kernel bug	8.3
6	WeakSauce	CVE-2014-3847	System bug	4.9
7	Graphics Buffer Overflow	CVE-2015-1474	Graphics bug	9.3
8	StageFright	CVE-2015-3864	Media bug	10.0
9	StageFright	CVE-2015-3829	Media bug	10.0
10	StageFright	CVE-2015-3828	Media bug	10.0
11	StageFright	CVE-2015-3824	Media bug	10.0
12	StageFright	CVE-2015-1539	Media bug	10.0
13	StageFright	CVE-2015-1538	Media bug	10.0
14	StageFright 2.0	CVE-2015-6602	Media bug	9.3
15	OpenSSL Transient	CVE-2015-3825	Serialization bug	7.2
16	PingPong root	CVE-2015-3636	Kernel bug	4.9
17	Samsung Remote Code	CVE-2015-7888	System bug	9.0
18	StageFright	CVE-2015-6608	Media bug	10.0
19	GraphicsBuffer Unflatten	CVE-2015-1528	System bug	9.3
20	StageFright	CVE-2015-6616	Media bug	10.0
21	Bug 22214934	CVE-2015-3860	System bug	7.2
22	Bug 25187394	CVE-2016-0807	System bug	7.2

Table 4.1: Fuzion24 vulnerability test suite

There are actually more tests available in the platform however we decided to remove any test that will cause the target device to become unstable. We tried each provided test against 6 different Android phones with different API version and removed any test that causes the application to end abnormally.

The following are the steps taken in running Fuzion24 device vulnerability test within Securitometer:

1. Filter the type of vulnerability tests that are relevant to target device's OS

2. Execute each vulnerability test

4.2.3.1 Filtering and Executing the Test

Android used to only support one type of CPU, the ARMv5 architecture. Now with more devices being produced, the support for more type of CPUs are also increasing [1]. Different CPUs support different instruction sets. Each instruction sets has something called Application Binary Interface (ABI) which defines how binaries such as the .so files should be created and interact with the system on machine-level.

When using Android NDK, developers must specify the ABI for each CPU architecture that they want to support. Since some of the tests are written in C and compiled to support several ABIS, there is a need to filter which test are suitable for the target device Securitometer is installed in.

In Android NDK, the following ABIs are supported: armeabi, armeabi-v7a, arm64-v8a, x86, x86-64, mips, mips64 [1]. The fuzion24 vulnerability test cases are both written both in Java and C. Those that are written in C are currently compiled to support 3 ABIS namely: armeabi, armeabi-v7a, and x86.

For tests that are written in Java, we do not need to filter them based on their ABI. However, for tests that are written in C, we need to filter out tests that can't run on the target device.

The followings depict the process that Securitometer went through to filter the vulnerability test that it is going to execute on target device's OS:

1. The following Java interface are used as a blueprint for each vulnerability test:

```

<<interfaceBlock>>
VulnerabilityTest
getCVEorID(): string
runTest(): boolean
getSupportedArchitecture(): List
```

Figure 4.6: Interface for each vulnerability test

2. Each vulnerability test is enclosed in their own class and has an attribute called *List supportedArchitecture*.
3. Another class called VulnerabilityOrganizer is used to scan target device's supported ABI by utilising the *ro.product.cpu.abi* property of Android. After that,

it compares the target device's supported architecture with every test suite available. If the comparison matched, that particular test is executed.

The following simplified code depicts the vulnerability organizer's job:

```
public List<VulnerabilityTest> getTests() {
    ArrayList<VulnerabilityTest> all;
    ArrayList<VulnerabilityTest> filtered;

    //ADD ALL TEST
    all.add(new CVE_2013_4787()); all.add(new CVE_2013_6282());
    all.add(new CVE_2011_1149()); all.add(new CVE_2014_3153());
    all.add(new CVE_2014_4943()); all.add(new CVE_2014_3847());
    all.add(new CVE_2015_1474()); all.add(new CVE_2015_3864());
    all.add(new CVE_2015_3829()); all.add(new CVE_2015_3828());
    all.add(new CVE_2015_3824()); all.add(new CVE_2015_1539());
    all.add(new CVE_2015_1538()); all.add(new CVE_2015_6602());
    all.add(new CVE_2015_3825()); all.add(new CVE_2015_3636());
    all.add(new CVE_2015_7888()); all.add(new CVE_2015_6608());
    all.add(new CVE_2015_1528()); all.add(new CVE_2015_6616());
    all.add(new CVE_2015_3860()); all.add(new CVE_2015_0807());

    //GET SUPPORTED ABI
    String cpu1 = SystemUtils.getProperty("ro.product.cpu.abi");
    String cpu2 = SystemUtils.getProperty("ro.product.cpu.abi2");

    //FILTER TEST
    for(VulnerabilityTest test : all){

        if(test.getSupportedArchitectures() == null) {
            System.out.println("Error for : " + test.getCVE());
        }
        if(test.getSupportedArchitectures().contains(CPU.ALL)){
            filtered.add(test);
        } else {
            if(isArchitectureSupported(test, cpu1) &&
               isArchitectureSupported(test, cpu2)){
                filtered.add(test);
            }
        }
    }

    return filtered;
}
```

}

Listing 8: Vulnerability organizer

4.3 Back-end: Two Scoring Mechanism

As explained in the previous sections, Securitometer actually performs three type of tests. However, because the target of these tests can be categorised to just two: Apps and Operating System. We decided to compile the result of the above tests into two scoring mechanisms namely: App Score and OS Score.

4.3.0.1 App Score

The app permission test and malware classifier contribute to the score of the collection of apps. Figure 4.7 depicts the scoring mechanism for an individual app.

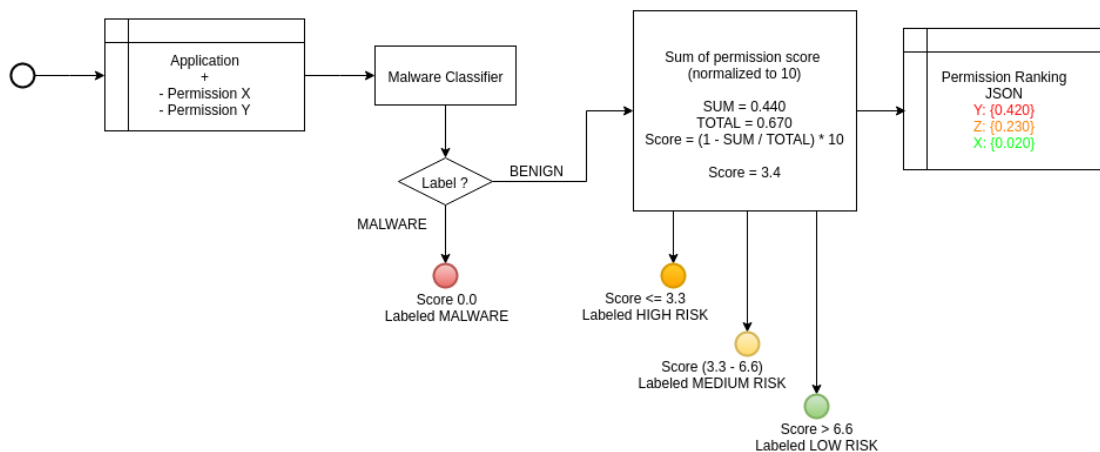


Figure 4.7: Score calculation for apps permission test

The process first started with the malware classifier. If Securitometer classified an app as a potential malware, it immediately assigned the lowest individual score to it, which is 0. If the app is classified as benign, Securitometer ran the app permission test to get the app's risk score.

Risk scores are assigned by scanning through the app's list of requested permissions. We used a ranking of Android permission based on a study that compares its percentage of occurrence in malicious and benign Android apps [37]. The top 40 riskiest permissions are listed in the ranking and given a score called risk level, as seen in figure 3.4.

To get an individual app score, we first calculated the sum of the apps permission's score. Then we divided it by the maximum score that it could get from permission rankings. The reason we multiply the risk score by 10 is because we want the security level to be on the scale of 10. We then subtract the risk score from 10 in order to give a higher score for devices that are more secure. The following is the formula to get an individual app score:

$$IndividualAppScore = 10 - (SumOfPermissionScore/MaximumScore) * 10$$

After Securimeter assigned an individual score for each app, it then calculated the average of these individual scores to display the overall score of the device's collection of apps. The following is the formula to get the overall app's score:

$$OverallAppScore = SumOfIndividualAppScore/NumberOfApps$$

4.3.0.2 OS Score

The device vulnerability test contributes to the score of the operating system. Each vulnerability within the test suite already has a CVSS2 score assigned. Since CVSS2 score has been widely adopted by notable organisations such as NIST (National Institute of Standards and Technology) in their National Vulnerability Database and the newer CVSS3 score is still in its initiation phase [24], we decided to use this score to represent the security level of target device's OS. We also want the security level to be on the scale of 10 and CVSS2 is already on the scale of 10.

The overall OS security score is then calculated by summing the CVSS2 score of each test that the OS are vulnerable to, dividing it by the total CVSS2 score of all tests that the Securimeter executed on the OS. We multiply the score by 10 to put it on the scale of 10. Then we subtract the score from 10 to give a higher score for devices that are more secure.

The following is the overall OS vulnerability score formula:

$$OverallOSScore = 10 - (VulnerabilityScore/MaximumScore) * 10$$

4.4 Front-end: Securimeter Application

In this section, we present the justification for Securimeter's user interface and the changes that it went through throughout the development process.

4.4.1 Splash Screen and Introductory Pages

The very first prototype of Securitometer has 3 pages namely: the main page, the app permission page, and the OS vulnerability test page. We did not change the number of Securitometer main pages even after we finished building the final version of Securitometer. However, during a focus group discussion to evaluate the prototype, the suggestions to add a splash screen and introductory pages for the first run was abundant. According to the suggestions, our prototype was lacking a lot of information on the type of test that it is running, and whether or not the test is harmful to the user.

We incorporated this suggestion and created a splash screen and introductory pages for the first run of Securitometer as seen in figure 4.8.

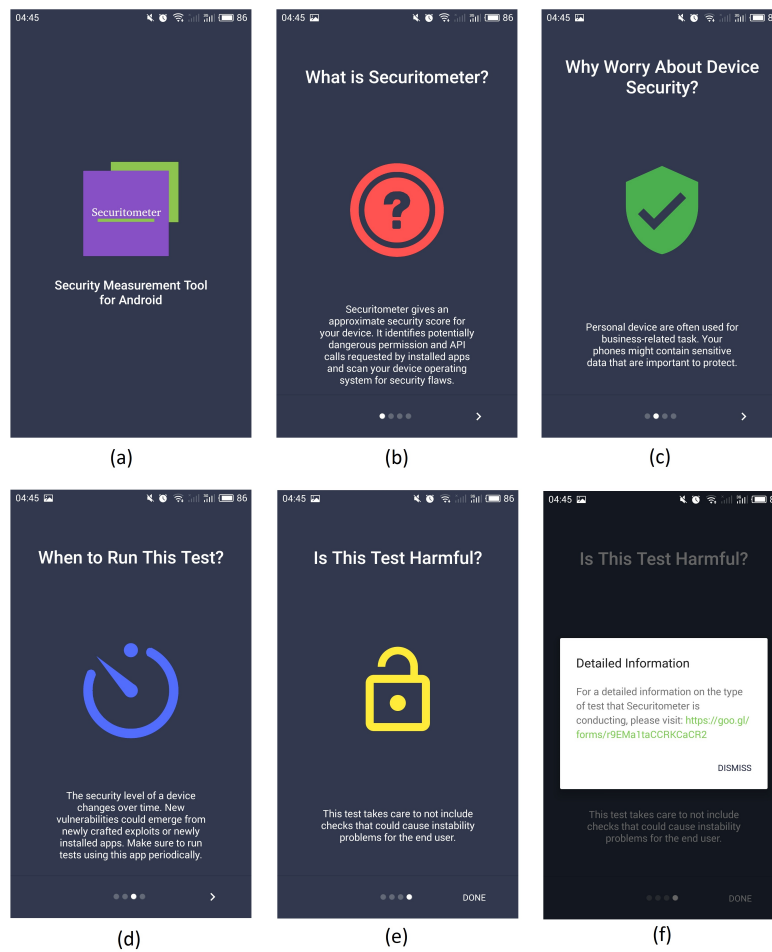


Figure 4.8: (a) Splash screen (b-f) Introductory pages

The structure of the introductory pages is originated from a study that uses the concept of 5W1H (What, Why, When, Where, Who, and How) to design a guiding system [41]. We adapted our description of Securitometer into the 5W1H structure.

The first page answers the what and the how, it introduce the user to what the purpose of Securitometer is and how it is going to achieve that purpose. The second page answers the why. It gives a reason to the user to actually consider using Securitometer. Lastly, the third page answers the when.

At the time when we presented the first prototype, participants in our focus group discussion seems to believe that Securitometer might cause harm to their device which is not the case. Our survey results for the prototype also stated that 50% of the respondents think that Securitometer is harmful. This is why we added the last introductory page to Securitometer, to make sure that users know about how Securitometer takes care not to include any tests that could cause problems for the user. We also added an additional URL that gives the user a chance to understand more about what Securitometer is doing.

4.4.2 Main Page

As seen on figure 4.9 the main page consist of device information on the top title bar, application title, and more information on the blank white screen. The start button is located at the bottom-right of the screen. This button was implemented using the Android's Floating Action Button. We believe that the bold style of the newest Android's Material Design makes the Floating Action Button strikingly hard to ignore.

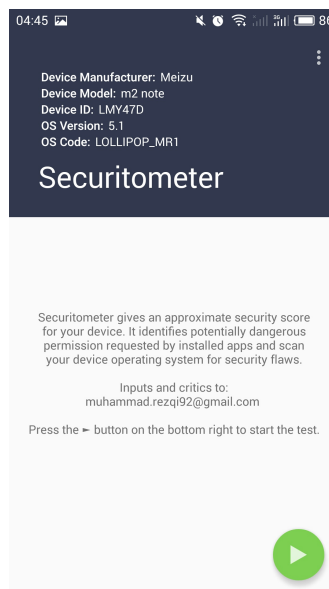


Figure 4.9: Empty main page

4.4.3 Running the Test

Incorporating the malware classifier that requires feature extraction from each application take away a lot of resources in running time. Securitometer runs slowly if the user happens to install a lot of applications on their device or the application that they install is particularly large.

To ensure that the user feels like they have freedom over the test, we added the ability to skip the app test (both malware classifier and app permission test) for apps that the user are confident about whether it is a threat or not to the security of the device as seen in figure 4.10. Users can also skip the whole app permission test if they only care about the OS vulnerability test. OS vulnerability test runs very fast so we did not give any option to skip this test. We could not offer the choice to just use the app permission test without the use of malware classifier as this could cause inconsistency in the scoring mechanism.

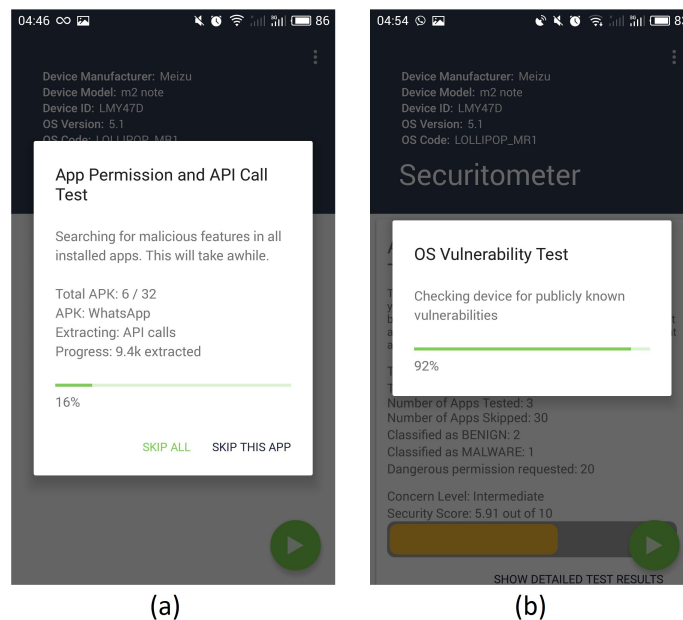


Figure 4.10: (a) App test (b) OS test

For users that would like to run a full test even though the app test took a long time, we put progress notification on the Android notification bar. Securitometer will continue to run in the background and user can still monitor its progress through the Android notification bar as seen in figure 4.11. For the progress notification, to avoid overflowing the limited space on notification bar, we truncated the extraction progress tracker by converting large numbers to smaller format by using k, M, etc as the unit

symbol.

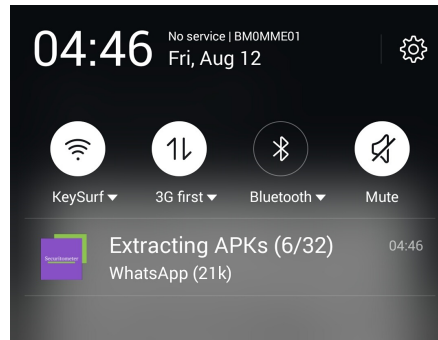


Figure 4.11: Securitometer progress in notification

We also realised that some users use the task manager in their device to kill running applications. To avoid the Securitometer notifications get stuck on the notification bar even though its process has already been killed, we run a kill notification service in the device whose only purpose is to make sure that if the Securitometer application gets killed, its notification also goes away.

4.4.4 Displaying the Result

Test results are showed in 3 different levels. They are as follow:

- Overall Test Result

This is the result that is first presented right after the test is completed. It contains a summary of the test being run and the overall score of the device.

Right after the test is completed, a dialogue containing summaries of the two tests are shown as seen in figure 4.12.

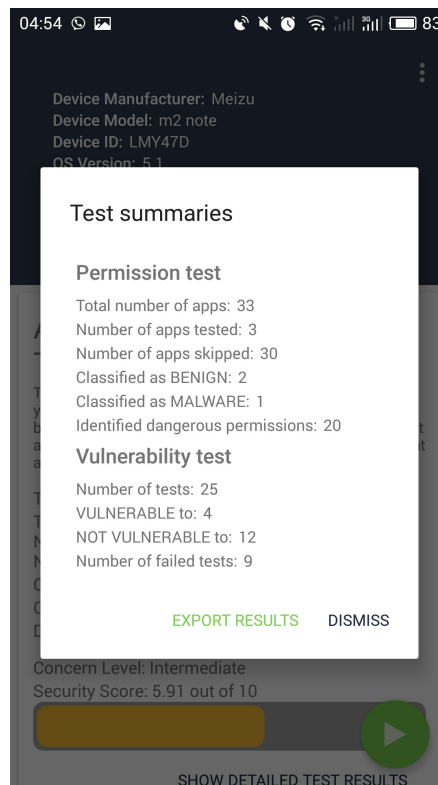


Figure 4.12: Test result summaries

After the user clicked dismiss, the summaries along with the overall score of each test are displayed in a card view as seen in 4.13. The overall score is also presented in a bar form representing the device level of security. The traffic light colour scheme is used to fill the bar as it has been proven to affect user's concern toward security [34]. The bar is filled with red if the score is below 3.3, amber is shown if the score is between 3.3 and 6.6, and green if the score is above 6.6. The range of the score is simply based on dividing the scale of 10 into 3.

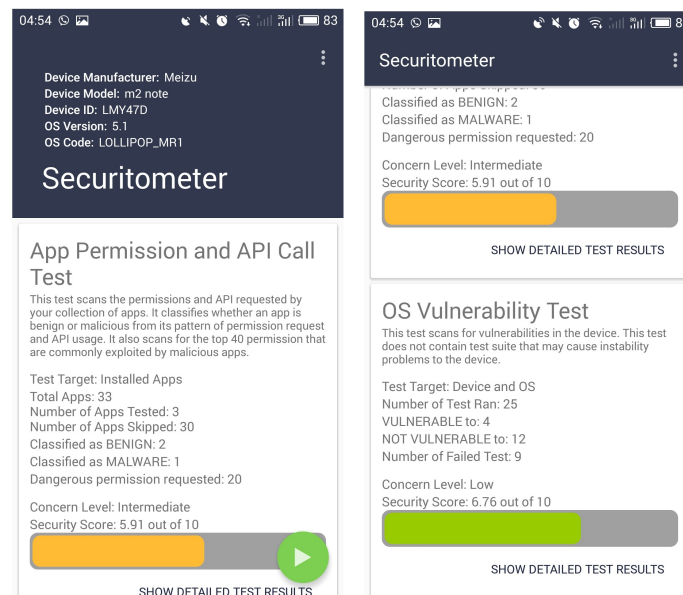


Figure 4.13: Overall test result

Prior to this look, the overall test result went through a number of slight revision as seen in figure 4.14. The changes were made to provide users with more information. Since one of our target users are security tester, we believe that more information means more things to write on the report.

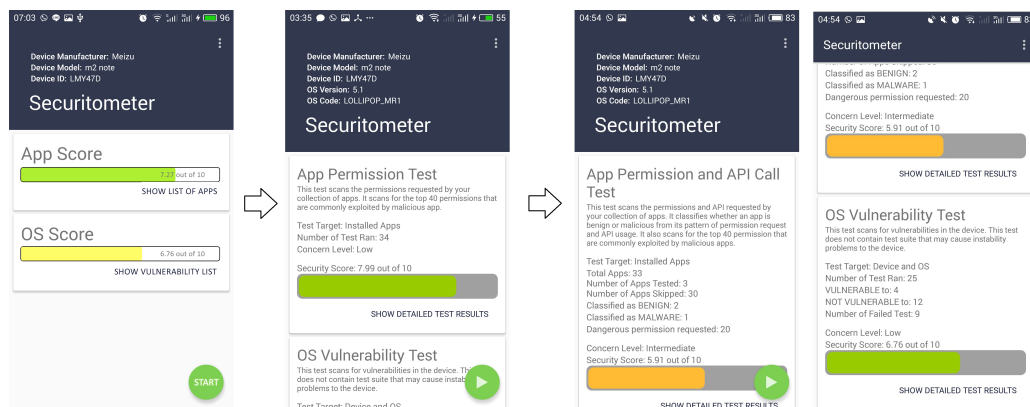


Figure 4.14: Changes made from oldest to the latest version

- Category Test Result

The second level result, or category test result, is displayed if the user clicked any of the test categories. Test category here means either the app permission test or the OS vulnerability test. Both shares a similar look to maintain consistency but contain very different information as seen in figure 4.15.

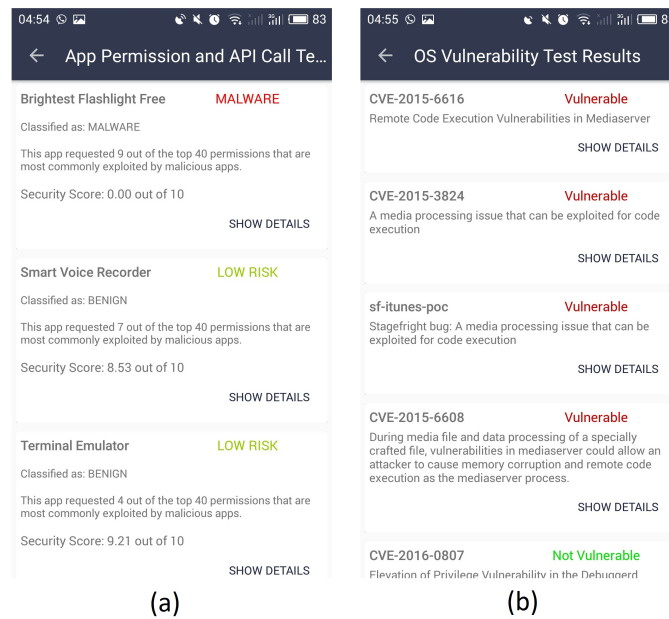


Figure 4.15: (a) App test results (b) OS test results

- Detailed Individual Result

Lastly, the detailed individual result is displayed if the users decide to click the "View Details" button on the second level result. In OS vulnerability test, this will display the details regarding the vulnerability test, including its CVE number, the description of the test, the link to its proof of concept, and its CVSS2 score. In the app permission test, this will display the list of permissions requested by the app and the risk level associated with that risk according to Securitometer's ranking of risk. The way information are organised is very different between the two type of test as seen in figure 4.16, because displaying a correct and complete information is more important here than consistency.

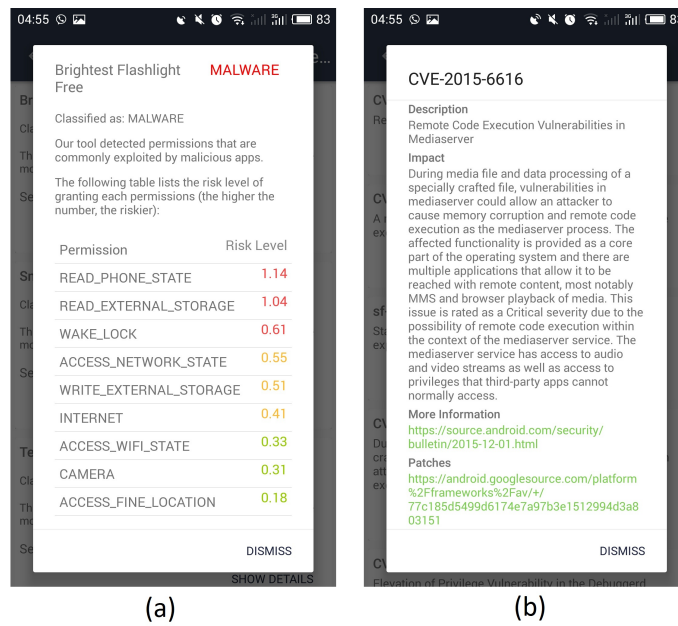


Figure 4.16: (a) App details (b) OS vulnerability details

4.4.5 Reporting the Result

Securitometer equipped itself with the capability to export its test results. Test results can be exported into a .JSON file or copied to clipboard. The options to export result is always available on the menu bar. If the user missed this option, the option to export is also offered each time a summary of test results is displayed. These options are depicted in figure 4.17:

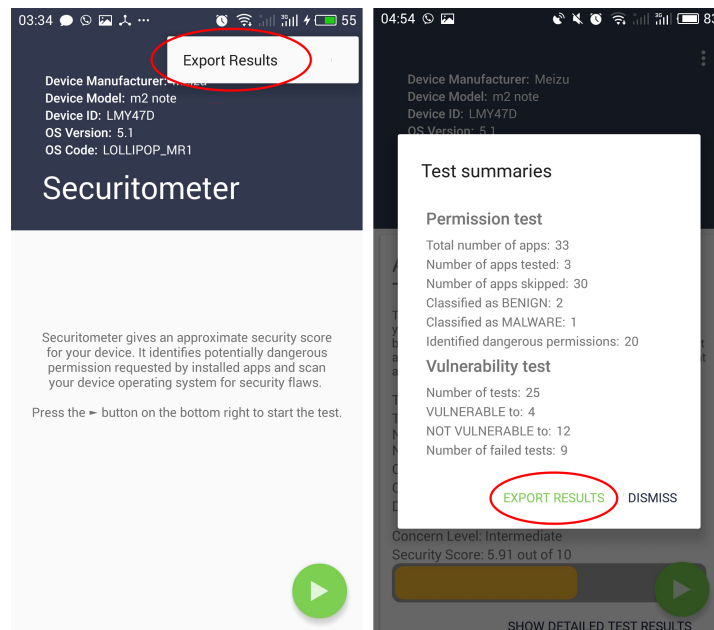


Figure 4.17: Export result options

We believe that JSON is the perfect format for Securitometer's result as it has a very good structure for parsing thus making it very interchangeable.

Chapter 5

Evaluation

In this chapter, we evaluate the success of our project in two ways: by carrying out feature-based evaluation against the requirements elicited in chapter 3, and by conducting a usability and acceptability study to potential users.

5.1 Feature-based Evaluation

5.1.1 Goals and Methodology

The main objective of this project is to develop Securitometer, an Android application that will probe for security flaws and vulnerabilities of the device where it is installed, and provide a security score of the device. The security score should represent the level of risk that the owner of the device is exposed to.

At the earliest stage of this project, we carried several requirement elicitation activities namely: research, brainstorming, focus group discussion, prototyping and survey. We have also summarised the insights that we gained from these activities. Therefore, we carry a feature-based evaluation where we assess the success of the project by comparing the final product developed against our initial list of requirements, as elaborated in section 3.2.

5.1.2 Results and Analysis

The followings are the results of comparing Securitometer's final version with its initial requirement:

1. Incorporate device vulnerability test suite

Securitometer has successfully incorporated a widely used device vulnerability test suite namely the fuzion24 vulnerability test suite as seen in section 4.2.3. We took care not to cause instability to the target device during Securitometer's vulnerability detection process. The vulnerability test suite also support different CPU architectures as seen in listing 8.

Unmet success criteria: None

2. Incorporate malware classifier

Securitometer has successfully incorporated a malware classifier called ExplainDroid [7], as one of its test suite. ExplainDroid's malware classifier decide whether or not an APK file is malicious or benign. ExplainDroid was originally implemented in Python. We ported ExplainDroid to Java for Android and put the resulting code inside Securitometer as seen in listing 2, listing 4, listing 6, and figure 4.5. By porting the whole classifier, it is able to run offline without the need to send any data to third party server.

Unmet success criteria: None

3. Incorporate app permission test

Securitometer has successfully incorporated an app permission test that utilises a ranking of permissions obtained from a research that produces the top-40 riskiest permissions commonly found in malicious applications as seen in 4.16. This test also gives a label to each app according to its risk level as seen in 4.15.

Unmet success criteria: None

4. Include introductory pages

Securitometer displays four introductory pages on its first run. We based the structure of the introductory pages from a study that uses the concept of 5W1H (What, Why, When, Where, Who, and How) to design a guiding system as seen in figure 4.8 [41]. We also make sure to add information about how Securitometer takes care not to cause any instability to its target device.

Unmet success criteria: None

5. OS scoring mechanism

We have developed a scoring mechanism for the target device's operating system. This score is calculated from the result of the device vulnerability test.

Section 4.3.0.2 provided the details including the formula used to calculate the OS score.

Securitometer uses the colour scheme of the traffic light to display the vulnerability result. Tests that state the device as vulnerable is displayed as red, not vulnerable is displayed as green, and failed test is displayed as yellow as seen in figure 4.15.

Unmet success criteria: None

6. App scoring mechanism

We have developed a scoring mechanism for the target device's collection of installed apps. This score is calculated from the result of both the malware classifier and the app permission test. Section 4.3.0.1 provided the details including the formula used to calculate the app score.

Securitometer uses the colour scheme of the traffic light to display the app test result. Tests that state the app as malicious are displayed as red. High-risk apps are displayed as dark amber. Medium-risk apps are displayed as light amber. Low-risk apps are displayed as green as seen in figure 4.15.

Unmet success criteria: None

7. Report mechanism

Securitometer has successfully equipped itself with the capability to export its test results. JSON is chosen as the format because it is easy to parse thus making it an interchangeable format.

Unmet success criteria: None

8. Progress dialogue

We realise that running the malware classifier takes a lot of resources including time. In order to ensure that Securitometer's scan does not prevent users from using their phone for a very long time, we provide a notification service that contains the current progress of the malware classifier as seen in 4.11. Users can hide the Securitometer in the background and use other apps and be notified of whether or not Securitometer has completed its scan yet.

Unmet success criteria: None

After comparing our final version of Securimeter with the initial requirements, we concluded that all goals regarding the functional aspects of Securimeter stated in section 3.2 were met.

5.2 Usability and Acceptability Study

5.2.1 Goals and Methodology

Through this evaluation, we aim to answer the following questions:

1. Who will benefit the most from Securimeter?
2. How helpful do potential users regard the security metrics used by Securimeter in evaluating the security of a device?
3. How well do potential users understand the purpose of Securimeter?
4. How useful is the system as regarded by the potential users?
5. How clear is the quality of information presented by Securimeter?
6. How do the potential users regard the quality of Securimeter's user interface?
7. Overall, how satisfied is the potential users (penetration testers and IT security auditors) with Securimeter?

Figure 5.1 depicts the methodology used in conducting usability and acceptability study for Securimeter. We conducted 2 usability and acceptability surveys to answer the above questions. The initial survey was disseminated after the first prototype of Securimeter was completed. This initial survey has two purposes: to find areas that need improvement from Securimeter, and to find out who will benefit the most from Securimeter. Due to the purpose mentioned above, this initial survey was designed to answer question 1-3.

We received 60 responses for this initial survey¹. We got the answer for questions 1-3. We also got a lot of inputs on how to improve Securimeter from this initial survey, because the questionnaire was filled with a lot of open-ended questions.

After the initial survey, We made a lot of changes and improvement to Securimeter. We incorporated the inputs we received from it and produced a final version of Securimeter as we see today.

¹<https://goo.gl/forms/0PgkHcqVZxTIGQ8z1>

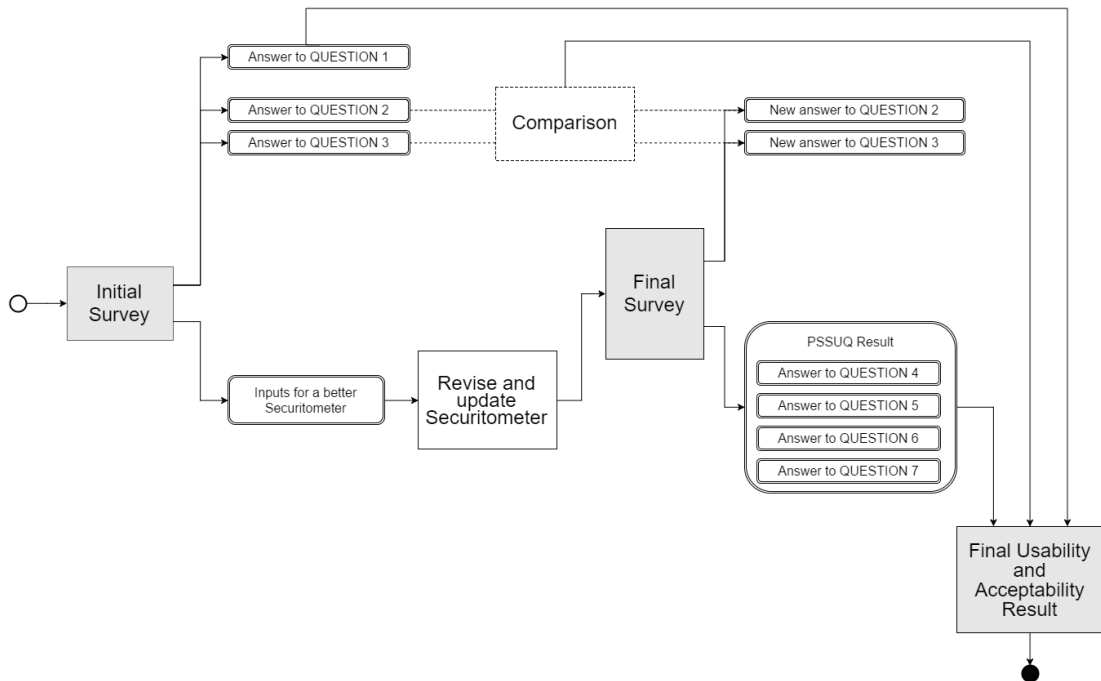


Figure 5.1: Methodology for usability and acceptability study

After the final version is completed, we decided to launch the final survey to further evaluate Securitometer's usability and acceptability aspect. Because we already got the answer to question 1, and we see that it is not necessary to enquire about this question again, we decided to exclude question 1 from the final survey. We received 40 responses for the final survey.

The final survey was designed to answer questions 2-7. We asked questions 2-3 again because we would like to see how potential users responded to them after the changes that we have made to Securitometer and compare them with their previous responses. We received 39 responses for the final survey². To answer questions 5-8, we used the standard PSSUQ (Post Study System Usability Questionnaire) [19]. PSSUQ can be used to calculate the followings:

- System usefulness, as represented by question 4.
- Information quality, as represented by question 5.
- Interface quality, as represented by question 6.
- Overall user satisfaction with the application, as represented by question 7.

²<https://goo.gl/forms/gqCMTCYntinoF1zm1>

In PSSUQ, responders have to decide their level of agreements towards the following 19 statements:

No	Question	Category
1	Overall, I am satisfied with how easy it is to use this app	Usefulness
2	It was simple to use this app	Usefulness
3	I can effectively complete my work using this app	Usefulness
4	I am able to complete my work quickly using this app	Usefulness
5	I am able to efficiently complete my work using this app	Usefulness
6	I feel comfortable using this app	Usefulness
7	It was easy to learn to use this app	Usefulness
8	I believe I became productive quickly using this app	Usefulness
9	The app gives error messages that clearly tell me how to fix problems	Information Quality
10	Whenever I make a mistake using the app, I recover easily and quickly	Information Quality
11	The information (such as online help, on-screen messages, and other documentation) provided with this app is clear	Information Quality
12	It is easy to find the information I needed	Information Quality
13	The information provided for the app is easy to understand	Information Quality
14	The information is effective in helping me complete the tasks and scenarios	Information Quality
15	The organization of information on the app screens is clear	Information Quality
16	The interface of this app is pleasant	Interface Quality
17	I like using the interface of this app	Interface Quality
18	This app has all the functions and capabilities I expect it to have	Interface Quality
19	Overall, I am satisfied with this app	Overall

Table 5.1: PSSUQ's list of questions

PSSUQ requires respondents to give their answer on a 7-point scales (lower number, means higher satisfaction with the application). We represent the 7-point scales

answer with the following category: strongly agree, agree, slightly in agreement, neutral, slightly in disagreement, disagree, strongly disagree. To conclude the final survey, we calculate the average of user's responses in PSSUQ. The average score of questions 1-19 represents overall user satisfaction with the app. The average score of questions 1-8 represents the app's level of usefulness. The average score of questions 9-15 represents the quality of information provided by the app. Lastly, the average score of questions 16-18 represents the quality of the app's interface.

Respondents for both surveys are recruited by utilising the Securitometer developer's list of colleagues and connection with several penetration testers and IT security auditors from professional service firms. Both questionnaires are created using Google Forms and are available online.

5.2.2 Results and Analysis

5.2.2.1 Initial Survey

This is a survey on the first prototype of Securitometer. The final survey about the latest version of Securitometer is elaborated in details in the next section.

The initial survey was filled by 60 respondents over the period of 15 days, from 11 July to 25 July 2016. Figure 5.2 illustrate the demographics of our respondents:

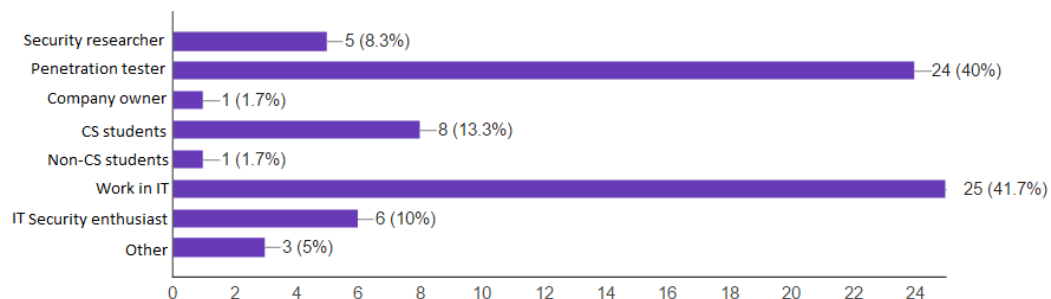


Figure 5.2: Demographics for initial survey

Our first effort in demographics is ensuring that all of the respondents are users of the Android smartphone. We then continue to ask the respondents about their occupation. We gave respondents the ability to choose more than one choice for their occupation/background thus explaining the bar chart results in figure 5.2.

The purpose of the demographic questions are to compare the respondents' background and classify the respondents into two: Android users who are IT security professionals, and Android users who are not. By utilising this classification, we can

conduct a more meaningful analysis based on the respondent's background.

For clarity, we will refer to Android users who are IT security professionals as just "IT security professionals". We will also refer to Android users who are not IT security professionals as just the "General Android users".

Based on figure 5.2 and the results that we got from our demographic questions, we depicted the distribution of our respondents according to their familiarity with professional IT security in the following diagram:

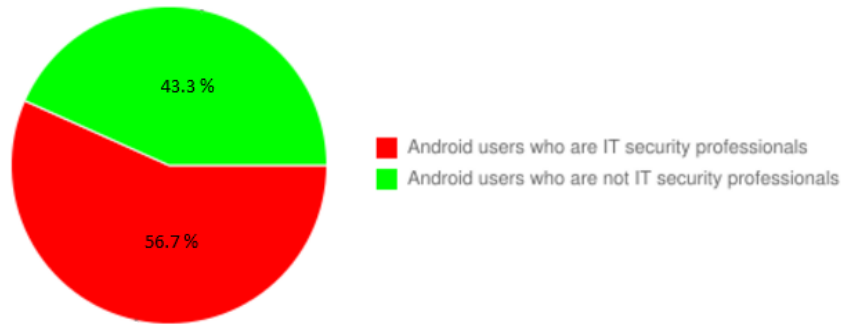


Figure 5.3: Classification of respondents for initial survey

The initial survey works like a walkthrough for the Securimeter's first prototype. It guides respondents through each page, then confirms whether respondents agree with the design choice or not. Some of the results have been elaborated before in section 3.1.3. They are as follow:

- About 81.7% of the respondents agreed with our justification for the use of app permission test. We did not see a notable difference between the answers from IT security professionals and the general Android users.
- About 95% of the respondents agreed on the use of traffic colour scheme to display the score helps users understand the impact that each app has on the security level of the device.
- Malicious app identification, OS vulnerability test, and App permission test are the top three metrics that respondents believe to represent the security level of a device.
- Stable vulnerability check is the most popular type of vulnerability testing suite (55%). We believe it is because its nature of not causing any harm to the device.

We notice that when we divide the result between IT security professionals and the general Android users, it seems that a majority of general Android users (about 76.9%) choose stable vulnerability check over the exploit attempt and lookup based (both at 11.55%). While for IT security professionals the choice varies between stable vulnerability check (about 38.2%), exploit attempt (about 35.2%), and lookup based (about 26.4%).

- About 50% of the respondents think that Securimeter could harm their devices. We did not see a notable difference between the answers from IT security professionals and the general Android users.

We also asked open-ended questions in the initial survey in order to elicit input for a better version of Securimeter. The summary of the inputs are as follow:

- Complains about the user interface

The first prototype of Securimeter is deemed as lacking a lot of aesthetic aspects. This is probably due to the fact that not a lot of thoughts are put into it during the prototype development phase. Figure 4.13 shows the comparison between the first prototype's user interface as evaluated in this survey and the latest version of Securimeter.

- Complains about the lack of information regarding the test performed

The first prototype actually already have the full main functionality of Securimeter except for the malware classifier. However, the amount of information provided to the user is deemed to be very lacking if found at all.

With regards to our objectives for conducting the initial survey, which is to answer question 1-3 from section 5.2.1. The followings are the results:

1. Who will benefit the most from Securimeter?

Figure 5.4 is the distribution of Securimeter's potential users according to our survey. About 83.3% believed Securimeter will be beneficial to penetration testers, and 73.3% believed it is useful as well to security researchers. This aligns with the motivation behind Securimeter as stated in section 1.1, which is to develop new security tools that are more suited and is specially made for mobile devices.

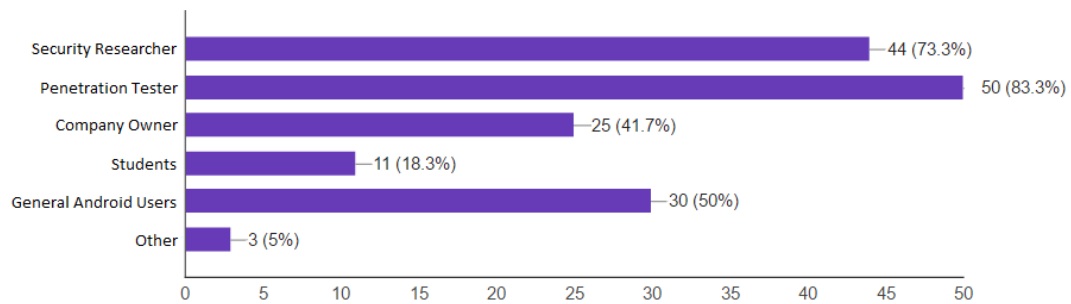


Figure 5.4: Securitometer's potential users

About 41.7% believed that company owners might be able to benefit from Securitometer too. This aligns with the sample usage scenarios that we provided in section 1.2.

One notable observation is that 50% of the total respondents believed that the general Android users could benefit the most from Securitometer. More specifically on the demographics, about 42.3% of the general Android users that answered the survey think that they will benefit the most from Securitometer. We originally thought that Securitometer's users will consist of just security professionals. This result could also be interpreted as the sign of how the need from general Android users on security awareness and security education is increasing.

2. How helpful do potential users regard the security metrics used by Securitometer in evaluating the security of a device?

- App security metric

Figure 5.5 depicts how user values the app security metric that we developed. About 11.7% respondents think that the app metrics is extremely helpful in evaluating the security of a device. Another 43.3% think that it is very helpful. This makes up a majority of 55% who think that the app metric is very helpful.

However, note that a large number of respondents (38.3%) think that the app metric is just moderately helpful. Combined with those who think that the app metric is just slightly helpful, it makes up a total of 45%. This is not to our satisfaction. This result is one of the reasons why we added an extra test to the app security metrics, which is the malware classifier. We

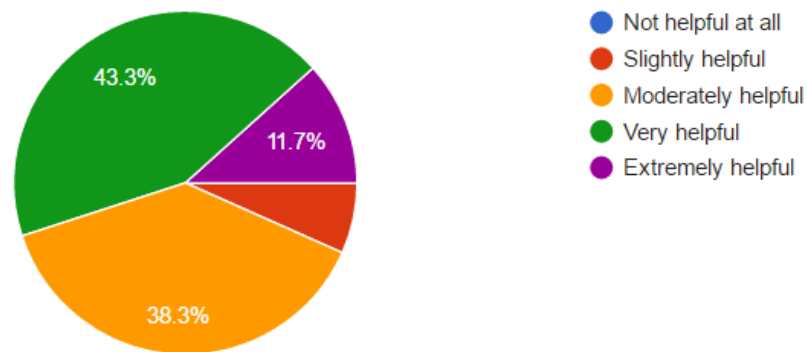


Figure 5.5: Survey result on the app security metrics

also decided to make a lot of changes to the way information are delivered to the users regarding the app metric and scoring mechanism.

- OS security metric

Figure 5.6 depicts how user values the OS security metric that we developed. About 18.3% respondents think that the OS metrics is extremely helpful in evaluating the security of a device. Another 45% think that it is very helpful. This makes up a majority of 63.3% who think that the OS metric is very helpful.

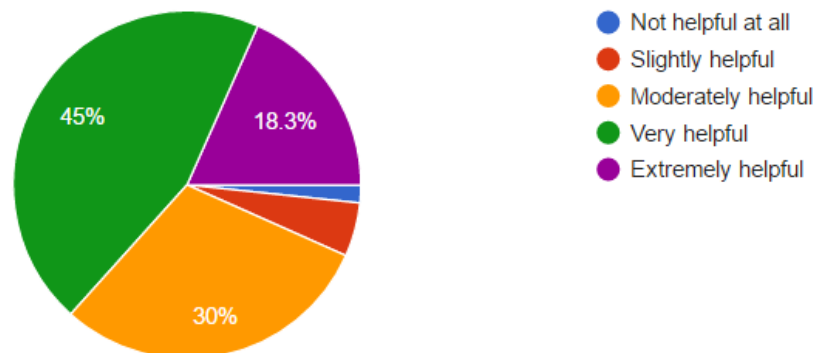


Figure 5.6: Survey result on the OS security metrics

Almost similarly with the app metric, a total of 35% of the respondents think that the OS metric is just moderately and slightly useful. We decided to incorporate the inputs requested from our open-ended questions to the way information are delivered to the user regarding the OS metric.

3. How well do potential users understand the purpose of Securimeter?

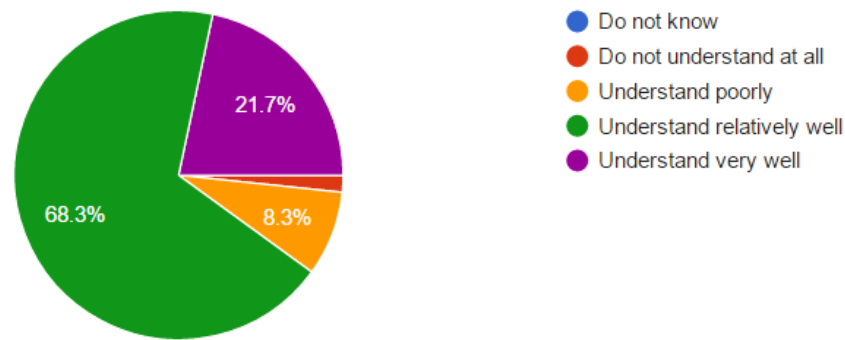


Figure 5.7: Respondent's level of understanding

Figure 5.7 is the distribution of the respondent's level of understanding about what Securitometer is about. We noticed that about 21.7% of the respondents understood the concept behind Securitometer very well. A large majority of 68.3% understood it relatively well. We consider the rest of the respondents to fail to meet our criteria of understanding Securitometer's concept. In response to this, we decided to make changes to the amount of information provided to the user as well as adding better introductory pages to Securitometer.

From this initial survey, we have obtained a solid answer to question 1 from section 5.2.1. We also received several inputs that could make Securitometer better. After proceeding to make changes and update Securitometer according to the above inputs, we believe that there is a need to revisit question 2 and question 3 again for the final survey.

5.2.2.2 Final Survey

The final survey consisted of two parts: the follow-up questions from the previous initial survey, and the PSSUQ (Post Study System Usability Questionnaire).

The final survey was filled by 39 respondents over the period of 10 days, from 1 August to 10 August 2016. We kept the original demographic questions from the initial survey and obtained the following distribution of respondents:

The distribution looks quite similar to the initial survey only with fewer respondents. This is probably because the questionnaire was live for fewer days compared to the initial survey. About 61.5% of our respondents are IT security professionals, the rest 38.5% are general Android users. Due to the similarity on the distribution of respondents' background, we decided that the decision to compare the result of this

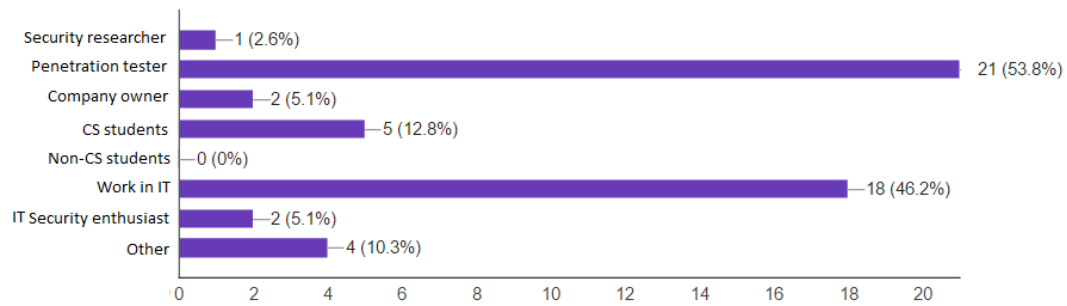


Figure 5.8: Demographics for final survey

survey and the initial survey is justified.

One of our objectives for conducting the final survey is to revisit question 2 and question 3 from section 5.2.1. The followings are the result:

1. How helpful do potential users regard the security metrics used by Securimeter in evaluating the security of a device?

- App security metric

After the initial survey completed, we added the following new features to the app metric: displaying more information about the test to the user, incorporating malware classifier to the metric, and adding a notification feature to the app permission test.

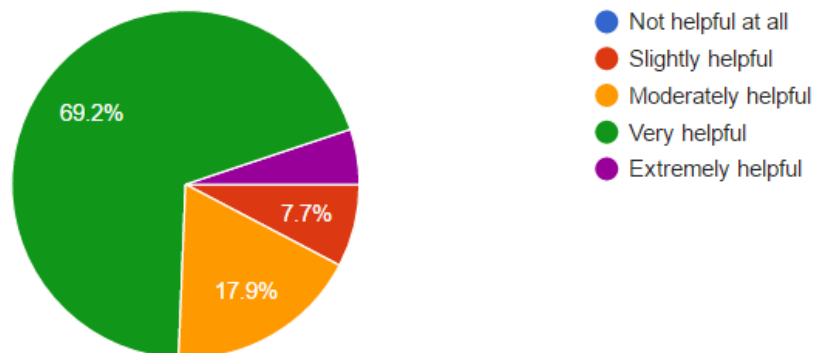


Figure 5.9: Final survey result on the app security metrics

After incorporating the new features and changes to the app security metric, we noticed an increase in the total number of respondents who think that the app metrics is either extremely helpful or very helpful (from 55% to 74.3%) as seen in figure 5.9.

- OS security metric

The following new features are added to the OS metric after the initial survey ended: displaying more information about the test, informing users that the test involved are harmless.

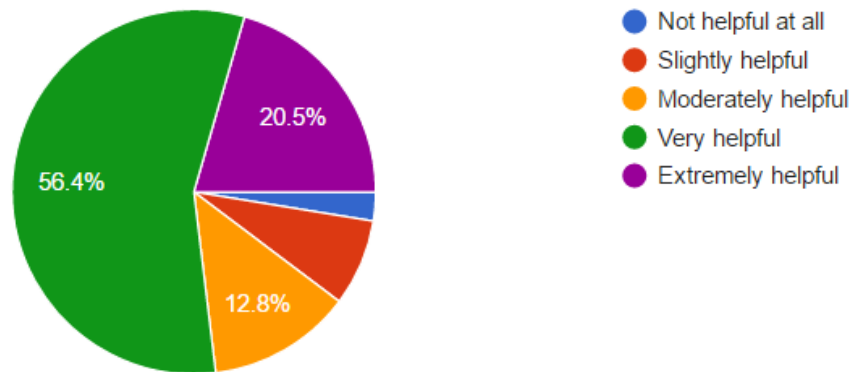


Figure 5.10: Final survey result on the OS security metrics

After incorporating the new features and changes to the OS security metric, we noticed an increase in the total number of respondents who think that the app metrics is either extremely helpful or very helpful (from 63.3% to 76.9%) as seen in figure 5.10.

2. How well do potential users understand the purpose of Securitometer?

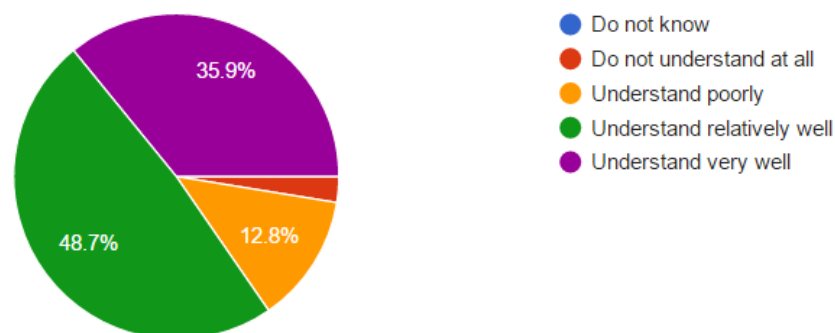


Figure 5.11: Respondent's level of understanding in final survey

Figure 5.11 is the distribution of the respondent's level of understanding about what Securitometer is about after we made major changes to its introductory pages. We noticed an increase in the total respondents who understood the concept very well (from 21.7% to 35.9%).

To answer question 4-7 from 5.2.1 we disseminated the PSSUQ along with the final survey. The PSSUQ measured four components of this project: overall user satisfaction, app usefulness, information quality, and interface quality. We will discuss the result of each component as follow:

1. How useful is the system as regarded by the potential users?

Question	Level of Agreement						
	(lower number, means higher degree)						
	1	2	3	4	5	6	7
Overall, I am satisfied with how easy it is to use this app	13	17	6	3	0	0	0
It was simple to use this app	12	20	6	1	0	0	0
I can effectively complete my work using this app	12	18	6	2	1	0	0
I am able to complete my work quickly using this app	11	18	7	2	1	0	0
I am able to efficiently complete my work using this app	12	19	5	3	0	0	0
I feel comfortable using this app	10	18	6	4	1	0	0
It was easy to learn to use this app	10	19	8	2	0	0	0
I believe I became productive quickly using this app	8	13	9	6	1	2	0

Table 5.2: PSSUQ's questions 1-8

Table 5.2 shows how the respondent's value the usefulness of Securitometer. By calculating the average of the above questions, we get a value of 2.09 which is high. We conclude that the respondents highly regarded Securitometer as a

useful application.

2. How clear is the quality of information presented by Securitometer?

Question	Level of Agreement						
	(lower number, means higher degree)						
	1	2	3	4	5	6	7
The app gives error messages that clearly tell me how to fix problems	6	12	11	6	1	3	0
Whenever I make a mistake using the app I recover easily and quickly	5	13	13	3	3	2	0
The information (such as online help, on-screen messages, and other documentation) provided with this app is clear	8	16	7	2	3	2	1
It is easy to find the information I needed	7	17	7	3	2	2	1
The information provided for the app is easy to understand	8	17	5	3	3	3	0
The information is effective in helping me complete the tasks and scenarios	9	19	5	3	1	2	0
The organization of information on the app screens is clear	10	19	5	2	1	2	0

Table 5.3: PSSUQ's questions 9-15

Table 5.3 shows how the respondent's value the quality of information provided by Securitometer. By calculating the average of the above questions, we get a value of 2.59 which is high. We conclude that the respondents valued Securito-

meter's quality of information highly.

3. How do the potential users regard the quality of Securitometer's user interface?

Question	Level of Agreement						
	(lower number, means higher degree)						
	1	2	3	4	5	6	7
The interface of this app is pleasant	12	17	8	1	0	0	1
I like using the interface of this app	12	18	6	2	0	0	1
This app has all the functions and capabilities I expect it to have	9	17	6	4	1	2	0

Table 5.4: PSSUQ's questions 16-18

Table 5.4 shows how the respondent's value the quality of Securitometer's user interface. By calculating the average of the above questions, we get a value of 2.16 which is high. We conclude that the respondents valued Securitometer's quality of information highly.

4. Overall, how satisfied is the potential users (penetration testers and IT security auditors) with Securitometer?

Question	Level of Agreement						
	(lower number, means higher degree)						
	1	2	3	4	5	6	7
Overall, I am satisfied with this app	11	20	4	2	2	0	0

Table 5.5: PSSUQ's question 19

Table 5.5 shows the value obtained for PSSUQ's question 19. To calculate the overall user satisfaction for Securitometer, we calculate the average of all questions in table 5.2, 5.3, 5.4, and 5.5. We get a value of 2.29 which is high. We conclude that overall, the respondents are highly satisfied by Securitometer.

We have compared the evaluation of Securitometer's first prototype with the eval-

uation of Securitometer's final version. Thus, we have successfully revisited question 2 and 3 from section 5.2.1. In comparing the results of both evaluation, we noticed an increasing trend in user's understanding and acceptance of the application.

We have also successfully disseminated the Post Study System Usability Questionnaire which answers question 4-7 from section 5.2.1. We concluded that the final version of Securitometer is highly regarded by the respondents both in terms of the level of usefulness, information quality, interface quality, and overall user satisfaction.

Chapter 6

Conclusion

6.1 Concluding Remarks

We took a look at the Android security ecosystem. As the leader of the smartphone market, it is doing quite poorly with at least one critical vulnerabilities found in 87.7% of these mobile operating system. Imagine having a private corporate network that is connected to hundreds of Android devices with critical vulnerabilities waiting to be exploited by malicious attackers and the system administrator has no control whatsoever with the configuration of the connected devices.

Motivated by the problems above, we set out to design and develop a new security tool that is more suited and is specially made for Android devices. Prior to developing anything, we made sure to conduct a number of requirement elicitation activities in order to acquire the most comprehensive appreciation of what must be implemented within the proposed new tool. We researched the Android's ecosystem, Android's security and vulnerabilities, penetration testing, and security tools that are related to mobile devices. We also conducted brainstorming and focus group discussion with two type of research groups: 1 group is filled with security experts to gain the technical standpoint, the other group is filled with usability experts to gain the user's viewpoint.

We developed Securitometer, an Android application that probe for security flaws and vulnerabilities of the device where it is installed, and provide a security score of the device. In developing Securitometer, we implemented three type of security test namely: malware classification test, app permission test, and device vulnerability test. We developed these three test by basing our design on well-established research and tools. The malware classification test is created by porting a malware classifier called ExplainDroid to Java for Android. The app permission test is adopted from a

research that assign risk level and rankings to Android permissions. Lastly, the device vulnerability test is adopted from fuzion24's test suite, a widely used vulnerability test suite for Android. We ported and incorporated all three test in the one Securitometer app.

In order to communicate the results of the above three test to the user, we developed two scoring mechanisms: app score and OS score. We also carefully designed and developed the user interface and user experience for Securitometer. In order to achieve the highest appreciation in this aspect, we quickly develop a prototype at the early stage of this project. We conducted user study on the prototype (focus group and survey) and incorporate the inputs that we obtained from them to the Securitometer that we now know.

In evaluating Securitometer from a software engineering point of view, we made sure to validate Securitometer's feature by conducting tests on each of its features. We made sure that it worked in the way that we intended it to by comparing its output with the output of the original research or tools that it based itself on.

In evaluating Securitometer from a usability point of view, we evaluated Securitometer by conducting surveys. We found that Securitometer is regarded as useful to not only penetration tester and security researcher, but also the general Android users. We also found that these users regarded the experience of using Securitometer as considerably satisfying in terms of its level of usefulness, information quality, and interface quality. This concludes the development and evaluation of Securitometer, the Android security metrics application.

6.2 Limitations of this Project

Although our work on Securitometer and its evaluation have yielded some positive results, the design of this project is not without flaws. The first limitation concerns the limited reach of our requirement elicitation activities. We believe that a security measurement application such as Securitometer should go through a lot of iterations and user evaluation. We only conducted two user evaluations: one for the prototype and the other one for the final version. After singling out the potential users for Securitometer from the first user evaluation, we believe that a more rigorous and targeted requirement elicitation activities need to be conducted in order to achieve the highest appreciation of the user's needs. This was not possible for us to complete due to our limited reach to the target users and the limited amount of time available for this project.

The second limitation concerns the way respondents are recruited for the final evaluation of Securimeter. Due to the limited number of security professionals out there, we recruited most respondents from the developer's professional network. Most respondents of this category might already know the identity of the developer. This might affect their sentiment in answering the survey questionnaires. To balance this, we also recruited random respondents for the category of general Android users. We believe that even though this might not be the perfect response to this limitation it should suffice to make the evaluation more objective than just recruiting friends and colleagues as respondents.

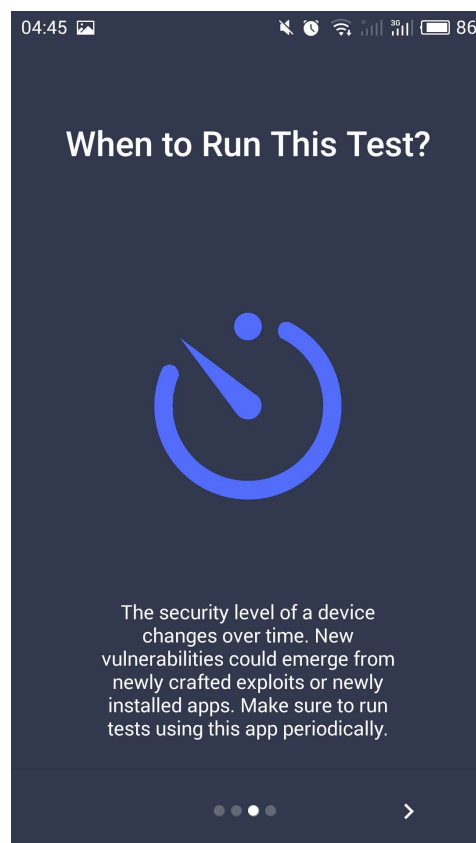
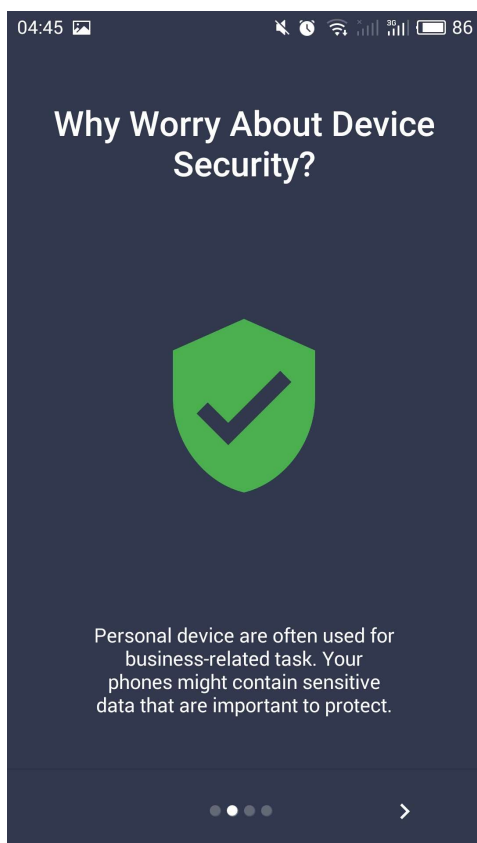
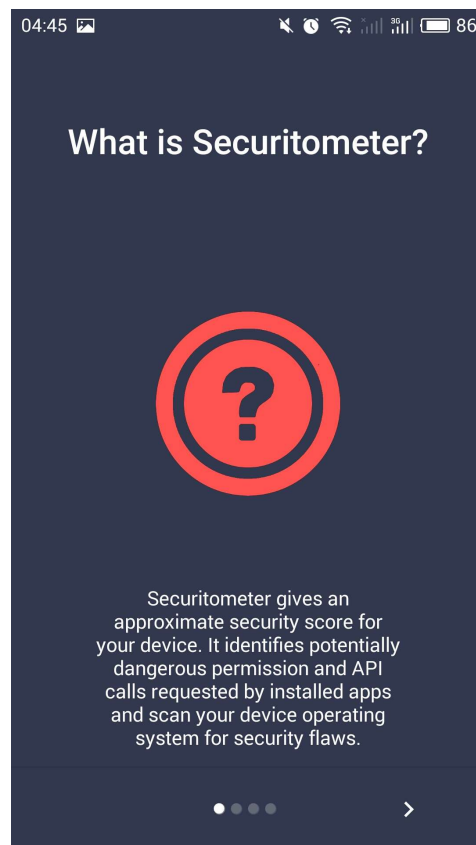
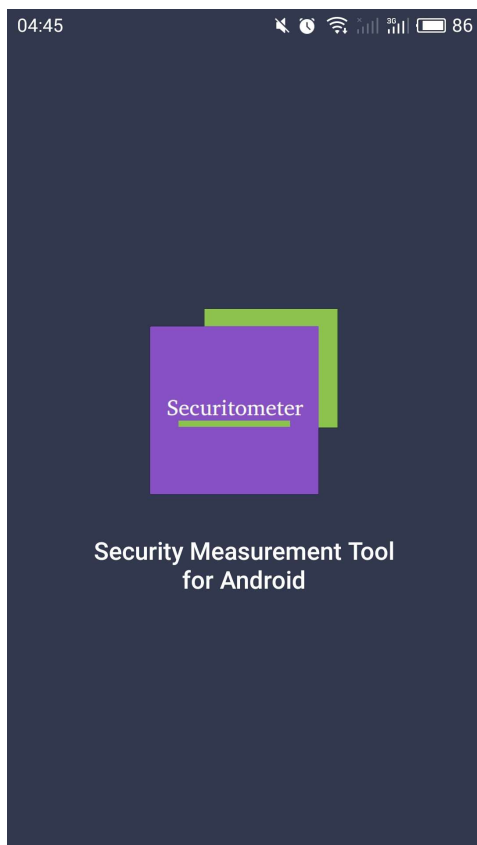
6.3 Future Work

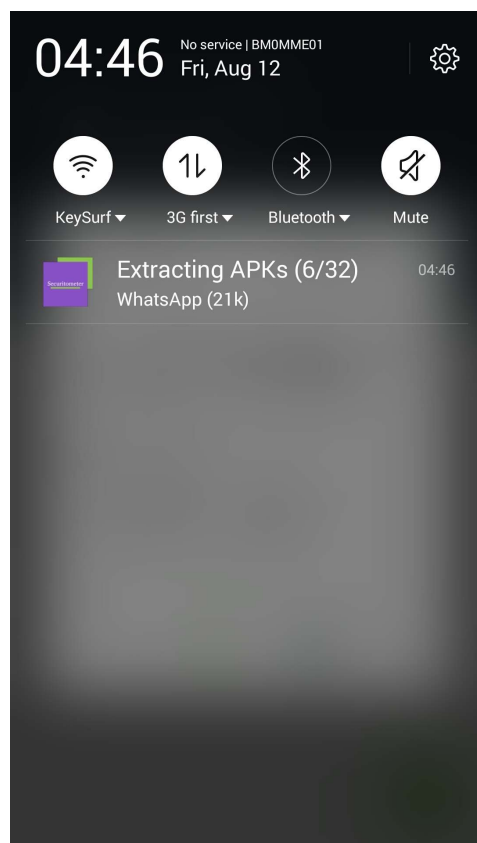
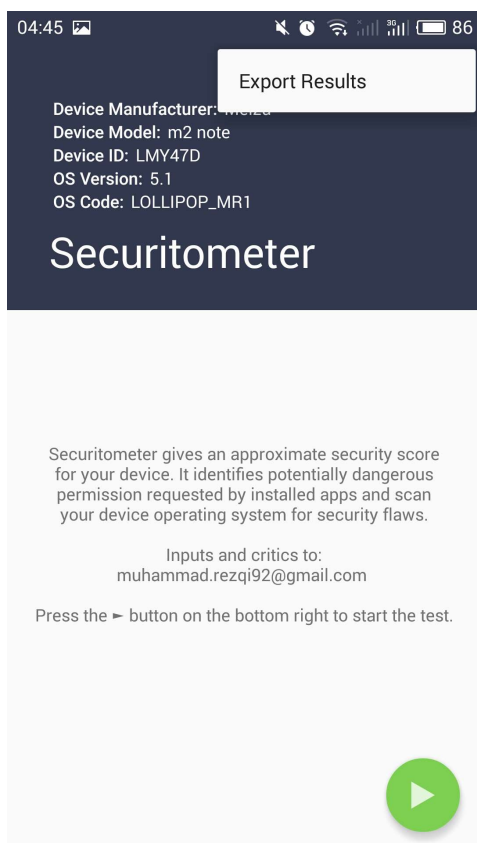
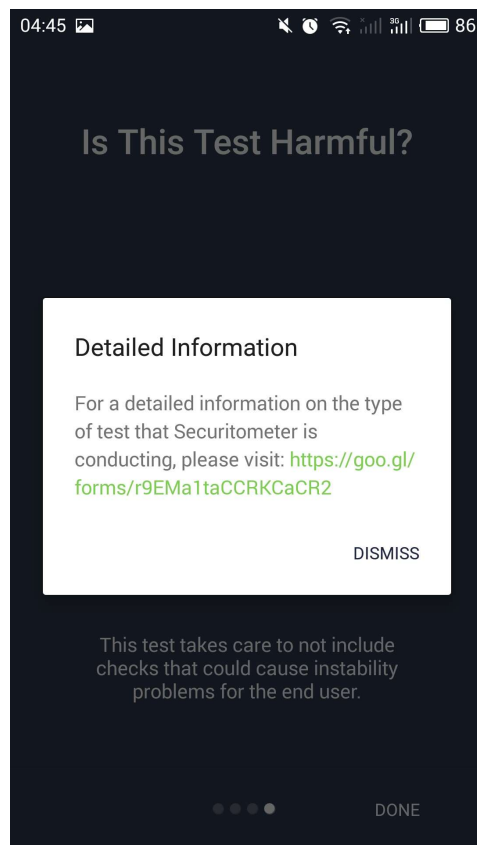
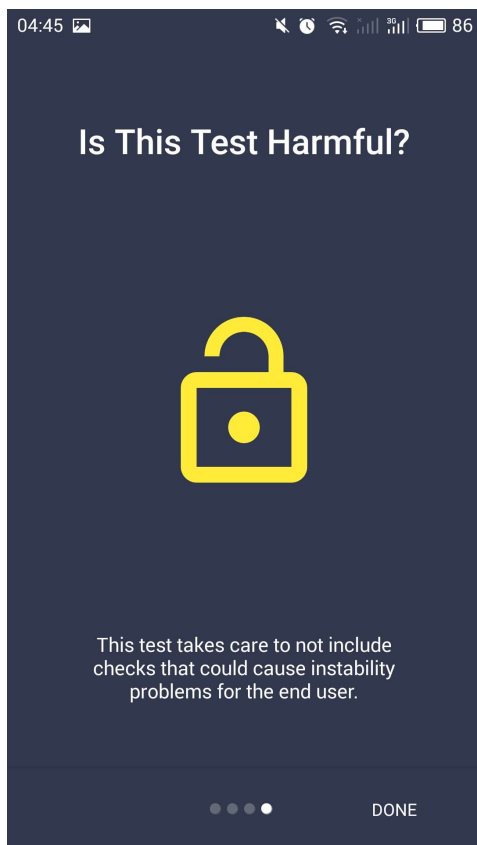
Security metrics and mobile security are both a growing field. I suggest the following areas as the main topic that could be further researched for the future development of Securimeter and the likes:

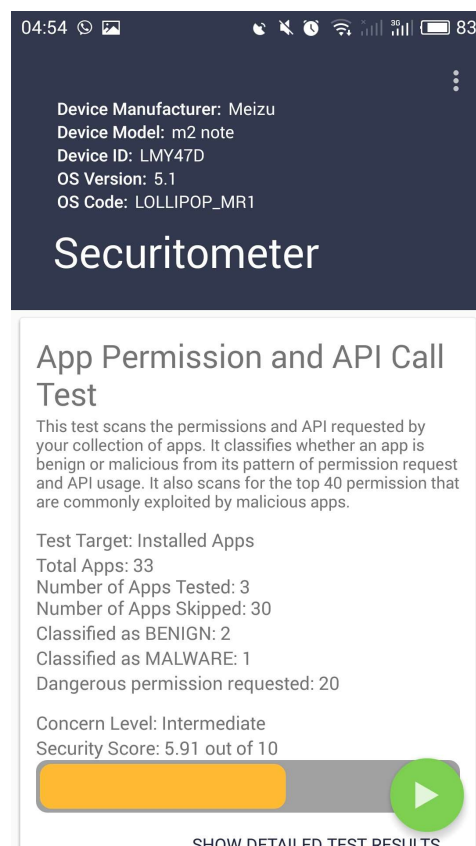
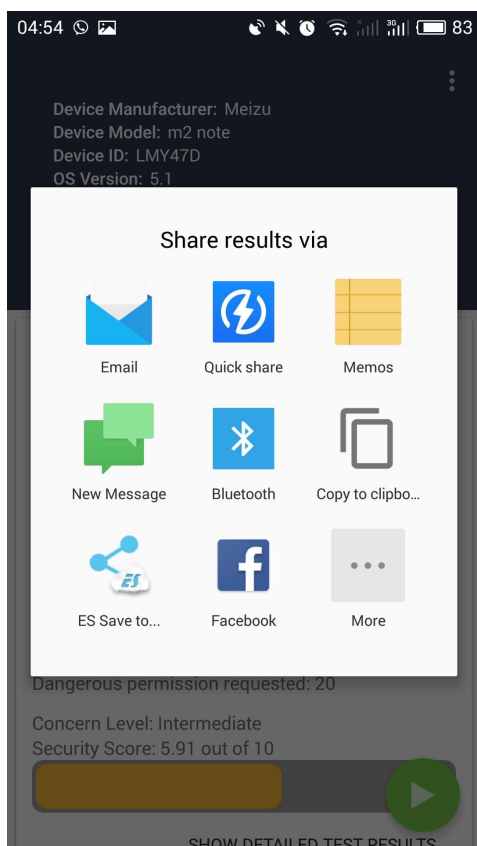
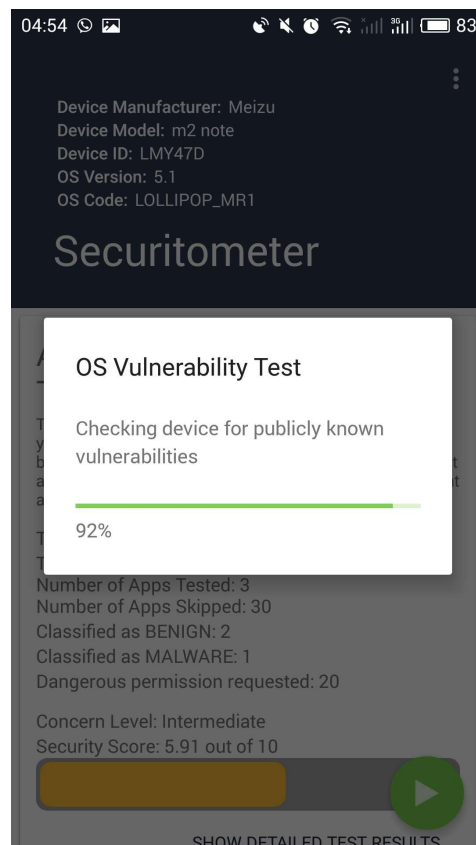
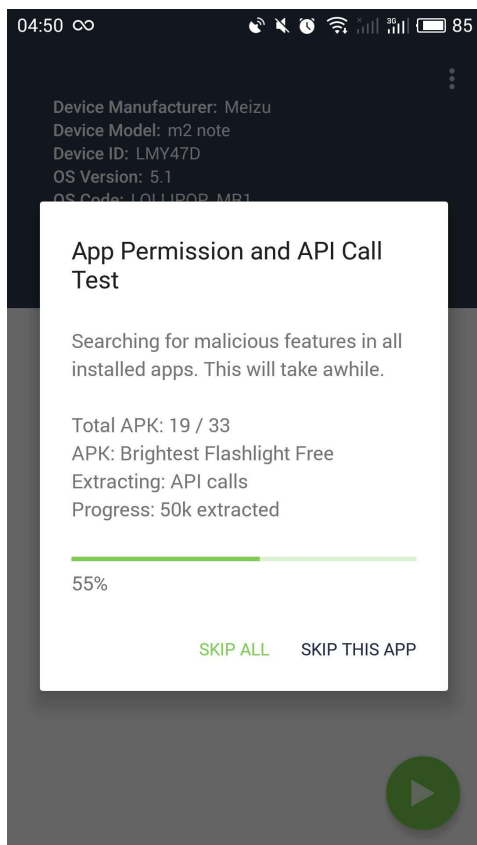
1. Introducing new metrics from device settings, configuration, communication history and connected networks.
2. Improving the accuracy of the incorporated malware classifier by collecting an anonymous list of installed applications. This might need further research on user privacy.
3. Improving the notification feature to let the user know if the device has not been scanned after a period of time. More research on usability might need to be included for this.
4. Automatic update of the permission ranking by feeding the application with updated data from an online malware classifier.
5. Adding more vulnerability test to the fuzion24 test suite. This will need a rigorous test to make sure that the test is stable and is not harmful to the target device.

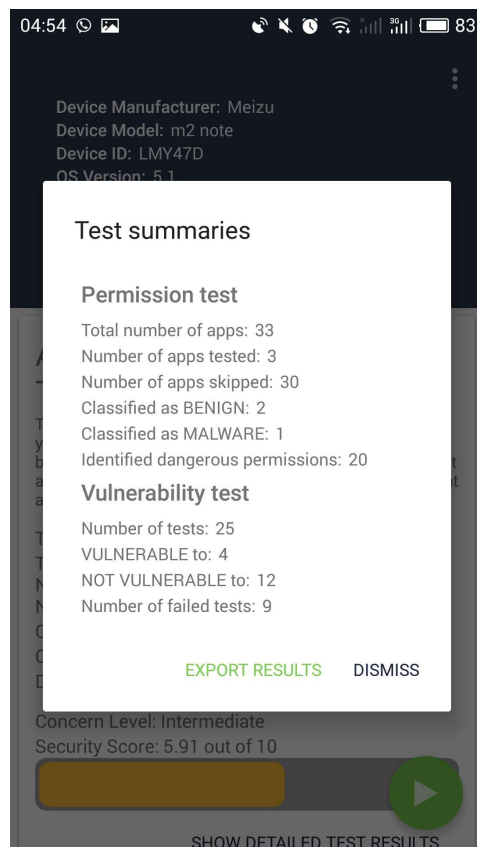
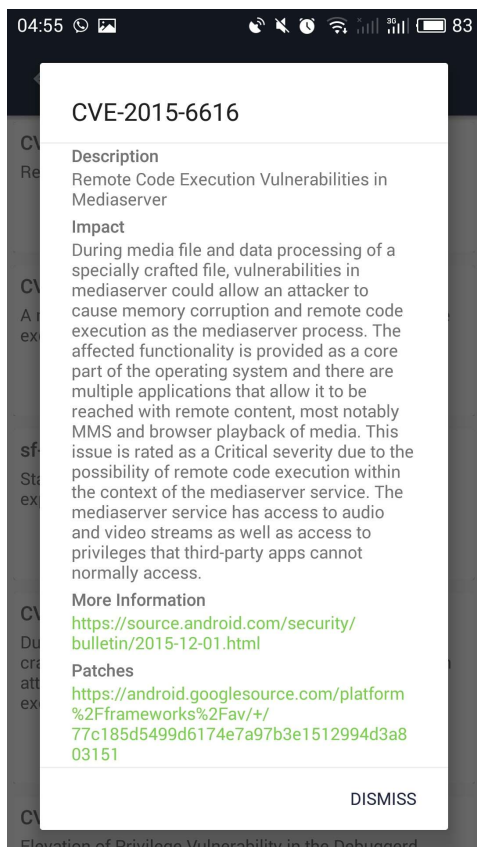
Appendix A

Appendix: Screenshot of Securitometer









Appendix B

Appendix: Initial Survey

```
{
  "buildInfo": {
    "fingerprint": "Meizu/meizu_m2note \
      \m2note:5.1\LMY47D\1464266312 \
      :user\release-keys",
    "kernelVersion": "3.10.65-user",
    "brand": "Meizu",
    "manufacturer": "Meizu",
    "model": "m2 note",
    "release": "5.1",
    "sdk": "22",
    "buildDate": 1464266461000,
    "id": "LMY47D",
    "cpuABI": "armeabi-v7a",
    "cpuABI2": "armeabi",
    "supportedABIs": [
      "arm64-v8a",
      "armeabi-v7a",
      "armeabi"
    ],
    "versionCode": 1,
    "versionName": "v.1"
  },
  "appResults": [
    {
      "name": "Brightest Flashlight Free",
      "label": "MALWARE",
      "score": 0,
      "permissions": "[android.permission.READ_PHONE_STATE, \\"
```

```

        android.permission.READ_EXTERNAL_STORAGE, \
        android.permission.WAKE_LOCK, \
        android.permission.ACCESS_NETWORK_STATE, \
        android.permission.WRITE_EXTERNAL_STORAGE, \
        android.permission.INTERNET, \
        android.permission.ACCESS_WIFI_STATE, \
        android.permission.CAMERA, \
        android.permission.ACCESS_FINE_LOCATION] "
    },
    {
      "name": "Smart Voice Recorder",
      "label": "BENIGN",
      "score": 8.529483579538518,
      "permissions": "[android.permission.READ_PHONE_STATE, \
        android.permission.READ_EXTERNAL_STORAGE, \
        android.permission.WAKE_LOCK, \
        android.permission.ACCESS_NETWORK_STATE, \
        android.permission.WRITE_SETTINGS, \
        android.permission.WRITE_EXTERNAL_STORAGE, \
        android.permission.INTERNET] "
    },
    {
      "name": "Terminal Emulator",
      "label": "BENIGN",
      "score": 9.214381638383593,
      "permissions": "[android.permission.READ_EXTERNAL_STORAGE, \
        android.permission.WAKE_LOCK, \
        android.permission.WRITE_EXTERNAL_STORAGE, \
        android.permission.INTERNET] "
    }
  ],
  "osResults": [
    {
      "name": "ZipBug 9950697",
      "isVulnerable": false
    },
    {
      "name": "CVE-2013-4787",
      "isVulnerable": false
    },
    {
      "name": "ZipBug 9695860",

```

```
    "isVulnerable": false
  },
  {
    "name": "CVE-2013-6282",
    "isVulnerable": false,
    "exception": "java.lang.Exception: Error running test"
  },
  {
    "name": "CVE-2014-3153",
    "isVulnerable": false
  },
  {
    "name": "CVE-2014-4943",
    "isVulnerable": false
  },
  {
    "name": "CVE-2014-3847",
    "isVulnerable": false
  },
  {
    "name": "CVE-2015-1474",
    "isVulnerable": false
  },
  {
    "name": "CVE-2015-1538-1",
    "isVulnerable": false,
    "exception": "java.lang.Exception: Test error"
  },
  {
    "name": "CVE-2015-1538-2",
    "isVulnerable": false,
    "exception": "java.lang.Exception: Test error"
  },
  {
    "name": "CVE-2015-1538-3",
    "isVulnerable": false,
    "exception": "java.lang.Exception: Test error"
  },
  {
    "name": "CVE-2015-1538-4",
    "isVulnerable": false,
    "exception": "java.lang.Exception: Test error"
```

```

    },
    {
      "name": "CVE-2015-1539",
      "isVulnerable": false,
      "exception": "java.lang.Exception: Test error"
    },
    {
      "name": "CVE-2015-3824",
      "isVulnerable": true
    },
    {
      "name": "CVE-2015-3828",
      "isVulnerable": false,
      "exception": "java.lang.Exception: Test error"
    },
    {
      "name": "CVE-2015-3829",
      "isVulnerable": false,
      "exception": "java.lang.Exception: Test error"
    },
    {
      "name": "CVE-2015-3864",
      "isVulnerable": false,
      "exception": "java.lang.Exception: Test error"
    },
    {
      "name": "sf-itunes-poc",
      "isVulnerable": true
    },
  ],
}

```

Listing 9: Sample report by Securitometer

Appendix C

Appendix: Initial Survey

Securitometer Application - Initial Survey

Hi, I am a student from University of Edinburgh, currently pursuing a master degree in cyber security and privacy.

I am developing an Android application for Penetration Tester, Security Tester, and IT Auditor called "Securitometer".

Securitometer is an app that gives an approximate security score for an Android device. A broad explanation on Securitometer is provided in the next section.

Currently I am building the prototype and would like to hear your inputs and critics. This will help me make improvements to this app. The survey should only take 10-20 minutes, and your responses are completely anonymous.

If you have any questions about the survey, please email me: muhammad.rezqi92@email.com

I would really appreciate your input!

*Required

Brief Explanation

What is Securitometer?

Securitometer is an Android app that gives an approximate security score for an Android device. Securitometer performs 2 type of tests:

1) App Permission Test

Scans your collection of apps and identify the requested permissions that are categorized as dangerous by Google.

2) System Test

Scans your device operating system for security flaws (vulnerability scanning).

Target Audience

Securitometer is designed for Penetration Tester, Security Tester, and IT Auditor so that they could incorporate this test as part of their professional working routine.

Bottom Line Question

How likely are you as a (penetration tester / IT auditor / computer science student / general android user), to use Securitometer?

Tasks & Usage Scenario

XYZ Corporation hired you as their penetration tester.

XYZ Corporation loaned a couple of company-issued smartphones for their employee.

XYZ Corporation has no clear Standard Operating Procedure about the usage of one's personal smartphones for work-related task and communication.

Objective

As a penetration tester, your objective is to test XYZ's compliance to security policy and its employees' security awareness especially on mobile phone usage. You need to submit a report on how vulnerable are these phones to attack and how several apps that are installed on these phones are potentially problematic.

Demographic Questions

1. What is your age? *

.....

2. What is your gender? *

Mark only one oval.

- ☐ Male
☐ Female

3. What is your job/occupation? (Check all that apply) *

Tick all that apply.

- ☐ Security Researcher
☐ Penetration Tester / IT Auditor / Security Tester
☐ Company Owner / Board Member
☐ Student (Computer Science)
☐ Student (Non Computer Science)
☐ Employee in an Information Technology Company
☐ None of the above, but is an Information Security Enthusiast
☐ None of the above

4. What level of decision-making authority do you have on purchasing IT related hardware, software or services for your organization? *

Mark only one oval.

- ☐ Final decision-making authority (individually or as part of a group)
☐ Significant decision-making or influence (individually or as part of a group)
☐ Minimal decision-making or influence
☐ No input

5. Do you use Android smartphone? *

Mark only one oval.

- ☐ Yes
☐ No

6. Do you know how to use Android smartphone? **Mark only one oval.*

- ☐ Yes
- ☐ No

7. How long have you been using Android smartphone? **Mark only one oval.*

- ☐ < 1 month
- ☐ 1 – 6 months
- ☐ 6 – 12 months
- ☐ 1 – 2 years
- ☐ 2 – 5 years
- ☐ > 5 years

8. Which brand of Android smartphone are you familiar with? (Check all that apply) **Tick all that apply.*

- ☐ Samsung
- ☐ LG
- ☐ Motorola
- ☐ HTC
- ☐ Sony
- ☐ Other:

9. How familiar are you with **Mark only one oval per row.*

	Not at all familiar	Slightly familiar	Moderately familiar	Very familiar	Extremely familiar
Google Play	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
App Installation in Android	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Android Vulnerabilities	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
App Permissions in Android	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Penetration Testing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Root Access in Android	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Starting Up

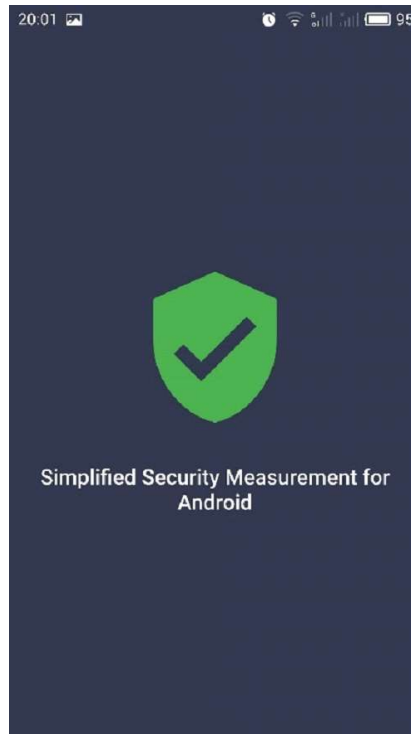
You acted as the penetration tester for XYZ Corporation.

You collected several company-issued Android smartphone.

You installed the Securitometer app, and start to evaluate the security level of each device.

You opened the app for the first time after installing it.

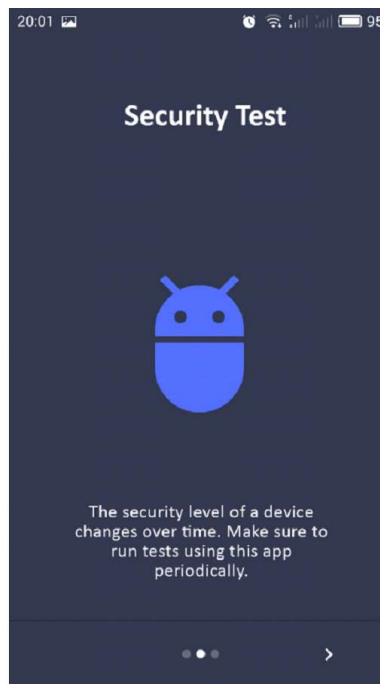
The first page that loads is the following (splash screen):



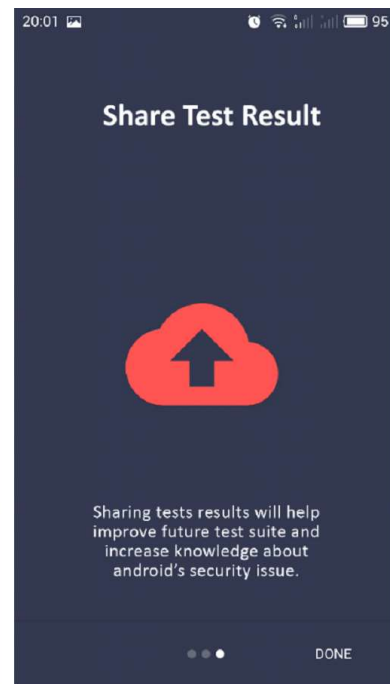
Following the splash screen, three introduction pages are automatically displayed with the following order. Swipe gesture is used to navigate between pages. You click "DONE" to continue.



(1)
First introductory slide



(2)
Second introductory slide



(3)
Third introductory slide

10. Do the introductory pages and splash screen explain the purpose of the Securitomer app? *

Mark only one oval.

☐ Yes

☐ No

11. How well do you think you understand what the idea of the Securitometer app is? **Mark only one oval.*

- ☐ Do not know
- ☐ Do not understand at all
- ☐ Understand poorly
- ☐ Understand relatively well
- ☐ Understand very well

12. What would you like to change? (if any)

.....

.....

.....

.....

Before entering the main page, a consent form is displayed to let user decide whether they want to share their test results or not. Users are also reminded that no sensitive, personal, and private information will be collected.

20:01

Share Test Result

Sharing scan results

Do you want automatically share the scan results for research purpose? The device information collected does not contains unique identifiers or private information. Nothing related to your Google (or any other) account is collected.

NOT SHARING SHARING

Sharing tests results will help improve future test suite and increase knowledge about android's security issue.

...

DONE

Result sharing consent form

13. How well does the consent form address users' concern about their privacy and safety? *

Mark only one oval.

- ☐ Very poorly
- ☐ Poorly
- ☐ Adequately
- ☐ Well
- ☐ Very well

14. What would you like to change? (if any)

.....

.....

.....

.....

Initiating the Test

After the splash screen and introductory pages, the first page that loads up is the following. The information about your device including its model and manufacturer are displayed on top of the app. You press the "START" button on the bottom-right to initiate the test.



Start the test to get the results



15. Is the device information helpful? *

Mark only one oval.

- ☐ Not helpful at all
- ☐ Slightly helpful
- ☐ Moderately helpful
- ☐ Very helpful
- ☐ Extremely helpful

16. Is the device information adequate for reporting purposes? *

Mark only one oval.

- ☐ Yes
- ☐ No

17. Is it clear what to do next? **Mark only one oval.*

- ☐ Yes
- ☐ No

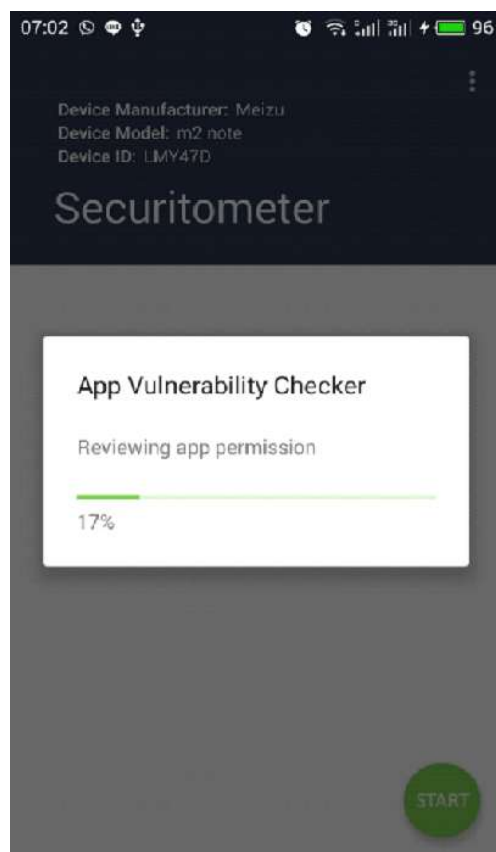
18. What would you like to change? (if any)

.....

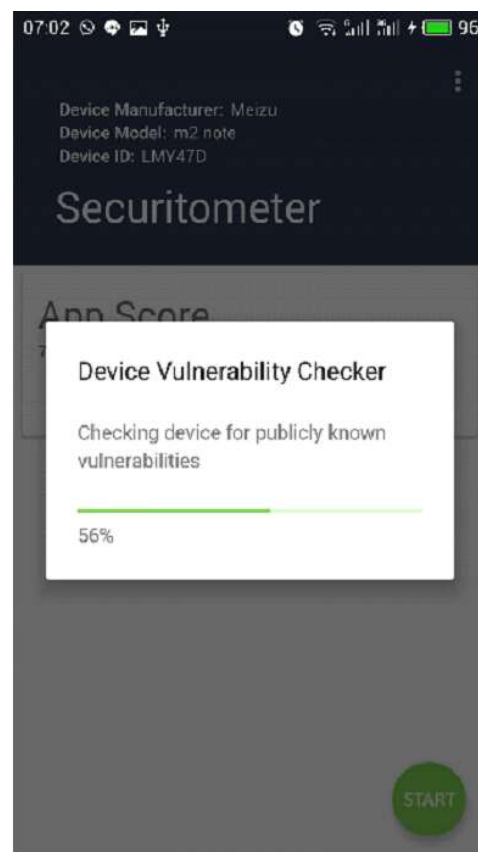
.....

.....

Then you click on the “START” button, which will display the following progress dialogs indicating that the test suite is initiated.

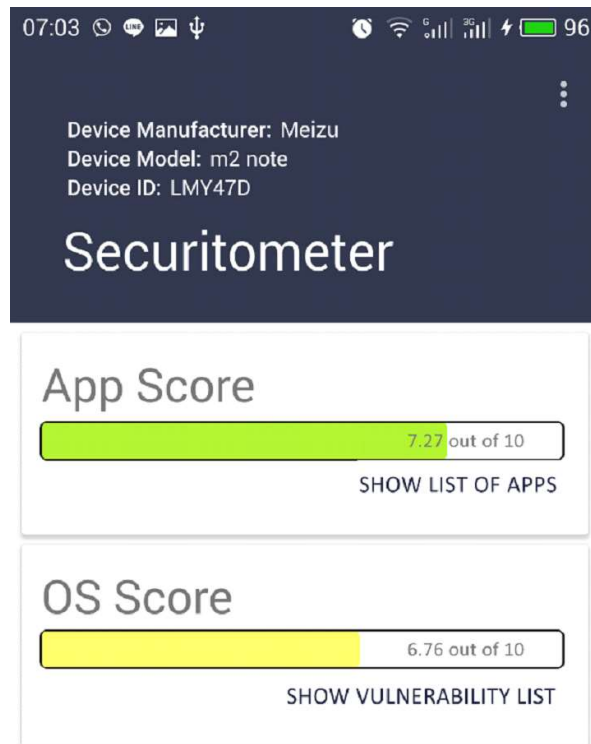


(1)



(2)

After a couple of seconds, the overall security score of the device will be displayed as follow:

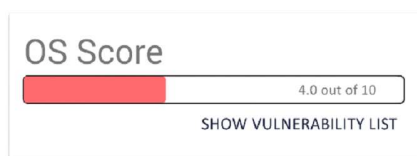


The coloured test result bar will change colour according to the test score. It follows a traffic light colour scheme.

For a score of below 5.0, the colour would be red.

For a score of 5.0 – 7.0, the colour would be yellow.

And for a score of 7.0 – 10.0 the colour would be green.



19. If you see a RED test result bar, how concerned are you about the security of the device? *

Mark only one oval.

- ☐ Not at all concerned
- ☐ Slightly concerned
- ☐ Somewhat concerned
- ☐ Moderately concerned
- ☐ Extremely concerned

20. What would you like to change? (if any)

.....

.....

.....

.....

App Permission Test, Scoring Mechanism

The app permission test, works by identifying dangerous permissions from your collection of apps.

Permissions are the mechanism by which app developers disclose how their apps will interact with users' devices and personal information on devices running Google's Android operating system.

Dangerous permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps. For example, the ability to read the user's contacts is a dangerous permission. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app.

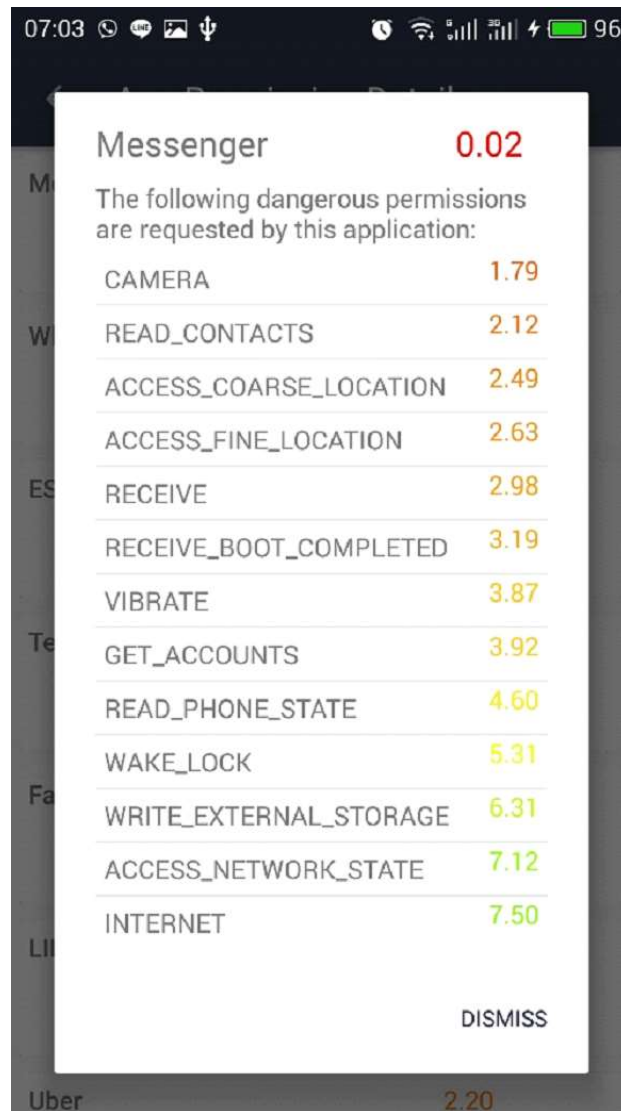


Now, you clicked the button "SHOW LIST OF APPS" on the Application score card, and the following page loads up:

07:03		20:03	
App Permission Details		App Permission Details	
Messenger	0.02	PitchLab Pro	9.98
	SHOW DETAILS		SHOW DETAILS
WhatsApp	0.02	GoPro	9.98
	SHOW DETAILS		SHOW DETAILS
ES File Explorer	0.03	Google PDF Viewer	9.98
	SHOW DETAILS		SHOW DETAILS
Telegram	0.06	Quora	9.99
	SHOW DETAILS		SHOW DETAILS
Facebook	0.12	Drive	9.99
	SHOW DETAILS		SHOW DETAILS
LINE	1.47	Securitometer	10.00
	SHOW DETAILS		SHOW DETAILS
Uber	2.20		

As explained above, the “Messenger” app in the sample above is scored based on its request for the following dangerous permissions:
SEND_SMS, READ_SMS, RECEIVE_SMS, CAMERA, READ_CONTACTS, etc.

Now, if you clicked the “SHOW DETAILS” button, the following details will show up, elaborating the list of permissions that each app requested and granted and the severity rating of each permissions.



The colour of the apps score will change according to its value. It follows a traffic light colour scheme.

If the score gets closer to 10.0, the colour would also get closer to green.

If the score gets closer to 0.0, the colour would also get closer to red.

The score for each requested dangerous permissions is based on an experiment that analyse a huge numbers of malicious and benign apps.

Do you agree with the following statement:

21. **"An app that requested more dangerous permissions have more potential to be problematic in the future, compared to the one that requested less dangerous permissions" ***

Mark only one oval.

- ☐ Strongly disagree
- ☐ Disagree
- ☐ Neither agree or disagree
- ☐ Agree
- ☐ Strongly agree

22. **"In ordering the list of apps, apps that are more likely to be problematic in the future should be displayed above the other apps (i.e: Messenger is put on top of Google PDF)" ***

Mark only one oval.

- ☐ Strongly disagree
- ☐ Disagree
- ☐ Neither agree or disagree
- ☐ Agree
- ☐ Strongly agree

23. **"The use of traffic colour scheme to display the score helps user understand the impact that each app has on the security level of the device" ***

Mark only one oval.

- ☐ Strongly disagree
- ☐ Disagree
- ☐ Neither agree or disagree
- ☐ Agree
- ☐ Strongly agree

24. **What would you like to change? (if any)**

.....

.....

.....

.....

25. **Would you consider the app permission score results, a helpful parameter in the evaluation of your devices' security? ***

Mark only one oval.

- ☐ Not helpful at all
- ☐ Slightly helpful
- ☐ Moderately helpful
- ☐ Very helpful
- ☐ Extremely helpful

26. If the Securitometer app could classify whether or not each of your app is benign or malicious, would you consider that to be a useful feature? *

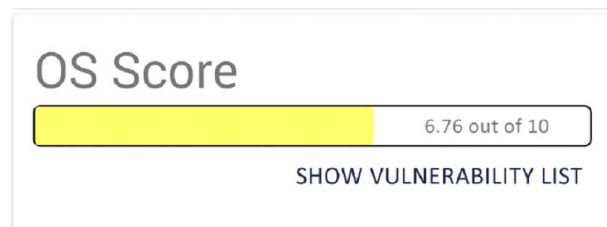
Mark only one oval.

- ☐ Not useful at all
- ☐ Barely useful
- ☐ Slightly useful
- ☐ Neutral
- ☐ Moderately useful
- ☐ Very useful
- ☐ Extremely useful

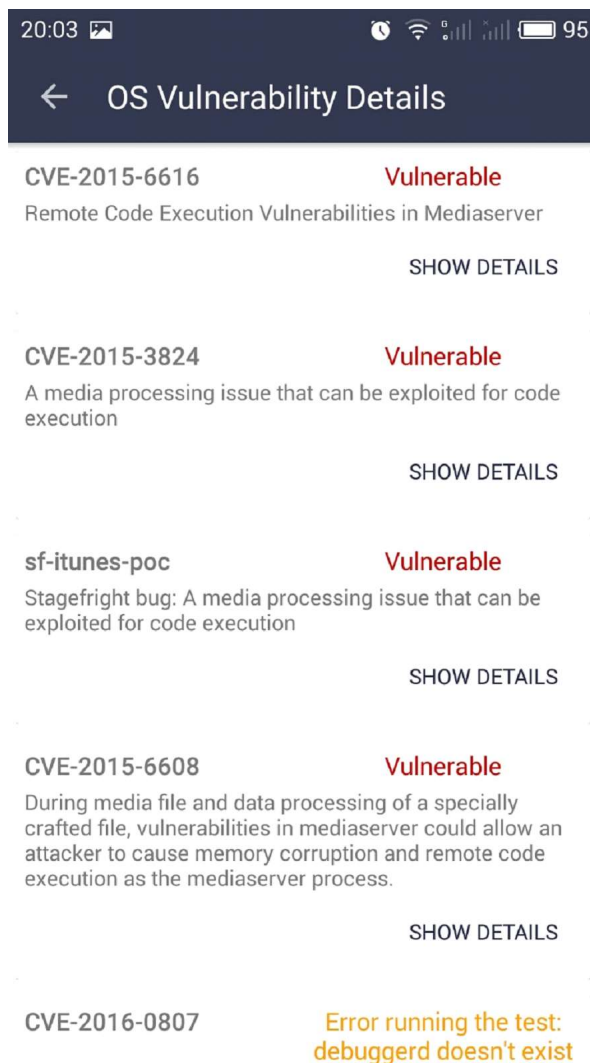
System Test, Scoring Mechanism

Securitometer assigns a device security level based on the CVSS score of each vulnerability identified within the device.

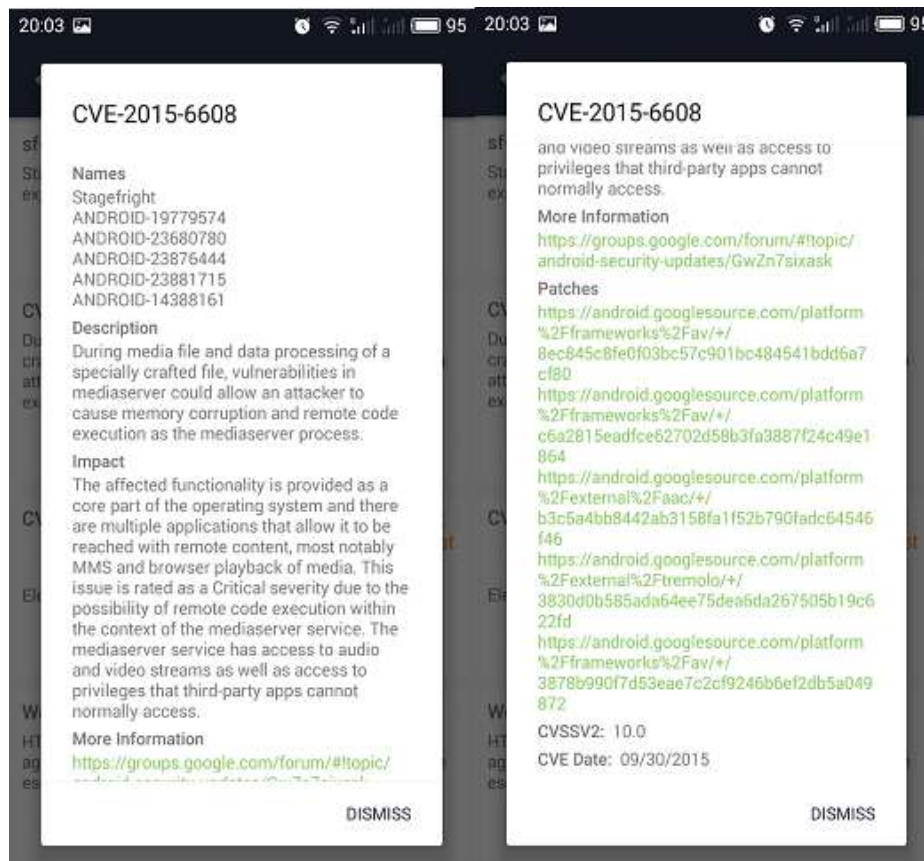
CVSS Score (Common Vulnerability Scoring System) is a common language of scoring IT vulnerabilities for IT managers, vulnerability bulletin providers, security vendors, application vendors and researchers.



You clicked the button “SHOW VULNERABILITY LIST” on the OS score card, and the following page loads up. Securitometer tries out several vulnerability checks on the device. The test results indicate whether or not the vulnerability is present within the device.



Clicking on the "SHOW DETAILS" button on each vulnerability, will display a vulnerability descriptor dialog. The dialog will contain information such as: vulnerability aliases, description, link for more information, and most importantly the CVSS V2 score.



27. Would you consider the OS score results, a helpful parameter in the evaluation of your devices' security? *

Mark only one oval.

- ☐ Not helpful at all
- ☐ Slightly helpful
- ☐ Moderately helpful
- ☐ Very helpful
- ☐ Extremely helpful

28. What would you like to change? (if any)

.....

.....

.....

.....

.....

Security Score System

29. Which of the following metrics do you believe is important to determine Android security? **Mark only one oval per row.*

	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree	I Don't Know
App Permission Check	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Malicious App Identification	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Storage Encryption Check	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
OS Vulnerability Scanner Result	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Use of Passcode	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

30. Would a device that passed these checks qualify as secure to you? **Mark only one oval.*

- ☐ Yes
- ☐ No

31. What metric do you believe is important in your opinion but is missing?

.....

.....

.....

32. In your opinion could the Securitometer app possibly harm your devices? **Mark only one oval.*

- ☐ Yes
- ☐ No

Type of Vulnerability Scanning App

There are 3 type of Vulnerability Scanning App:

1) Lookup Based

App that uses the device version/build information to lookup common vulnerabilities found in that Android version. No actual vulnerability scan takes place.

2) Exploit attempt

App that attempts exploits which could cause instability to the device being scanned.

3) Stable Vulnerability Check

App that attempts to detect the presence of vulnerabilities without actually exploiting it, taking care to not include checks that could cause instability problems for the end user and therefore may omit checks that could cause these types of issues.

In this section, please rank the type of vulnerability scanning app that you prefer for conducting a penetration test on other people's Android device (for example: your client's employee) !

33. Rank the type of vulnerability scanning app that you prefer, (left being the most preferred, and right being the least preferred)

Mark only one oval.

- ☐ Stable Vulnerability Check > Exploit Attempt > Lookup Based
- ☐ Stable Vulnerability Check > Lookup Based > Exploit Attempt
- ☐ Exploit Attempt > Lookup Based > Stable Vulnerability Check
- ☐ Exploit Attempt > Stable Vulnerability Check > Lookup Based
- ☐ Lookup Based > Stable Vulnerability Check > Exploit Attempt
- ☐ Lookup Based > Exploit Attempt > Stable Vulnerability Check

ACCEPTANCE TEST

34. How well do you think you understand what the idea of the Securitometer app is? *

Mark only one oval.

- ☐ Do not know
- ☐ Do not understand at all
- ☐ Understand poorly
- ☐ Understand relatively well
- ☐ Understand very well

35. How helpful/useful do you think the application would be for you? *

Mark only one oval.

- ☐ Not useful at all
- ☐ Barely useful
- ☐ Slightly useful
- ☐ Neutral
- ☐ Moderately useful
- ☐ Very useful
- ☐ Extremely useful

36. Who do you think will benefit the most from using this application? (Check all that apply) *

Tick all that apply.

- ☐ Security Researcher
- ☐ Penetration Tester / IT Auditor / Security Tester
- ☐ Company Owner / Board Member
- ☐ Students
- ☐ General Android Owners
- ☐ Other:



Appendix D

Appendix: Final Survey

Securitometer Application - Final Survey

Hi, my name is Rezqi, a student from University of Edinburgh, currently pursuing a master degree in cyber security and privacy. I developed an Android application for Penetration Tester, Security Tester, and IT Auditor called "Securitometer".

If you have filled my initial survey before (link: <https://goo.gl/forms/HZ9rKpmc0yfaYB4I3>), this is a survey to evaluate the changes that I have incorporated to the App based on your inputs.

If you have any questions about the survey, please email me: muhammad.rezqi92@gmail.com

I would really appreciate your input!

***Required**

Demographic Questions

1. What is your age? *

.....

2. What is your gender? *

Mark only one oval.

- ☐ Male
☐ Female

3. What is your job/occupation? (Check all that apply) *

Tick all that apply.

- ☐ Security Researcher
☐ Penetration Tester / IT Auditor / Security Tester
☐ Company Owner / Board Member
☐ Student (Computer Science)
☐ Student (Non Computer Science)
☐ Employee in an Information Technology Company
☐ None of the above, but is an Information Security Enthusiast
☐ None of the above

Scenario

You acted as the penetration tester for XYZ Corporation.

You collected several company-issued Android smartphone.

You installed the Securitometer app, and start to evaluate the security level of each device.

Instructions

Try to use the device to evaluate the security level of your own device.

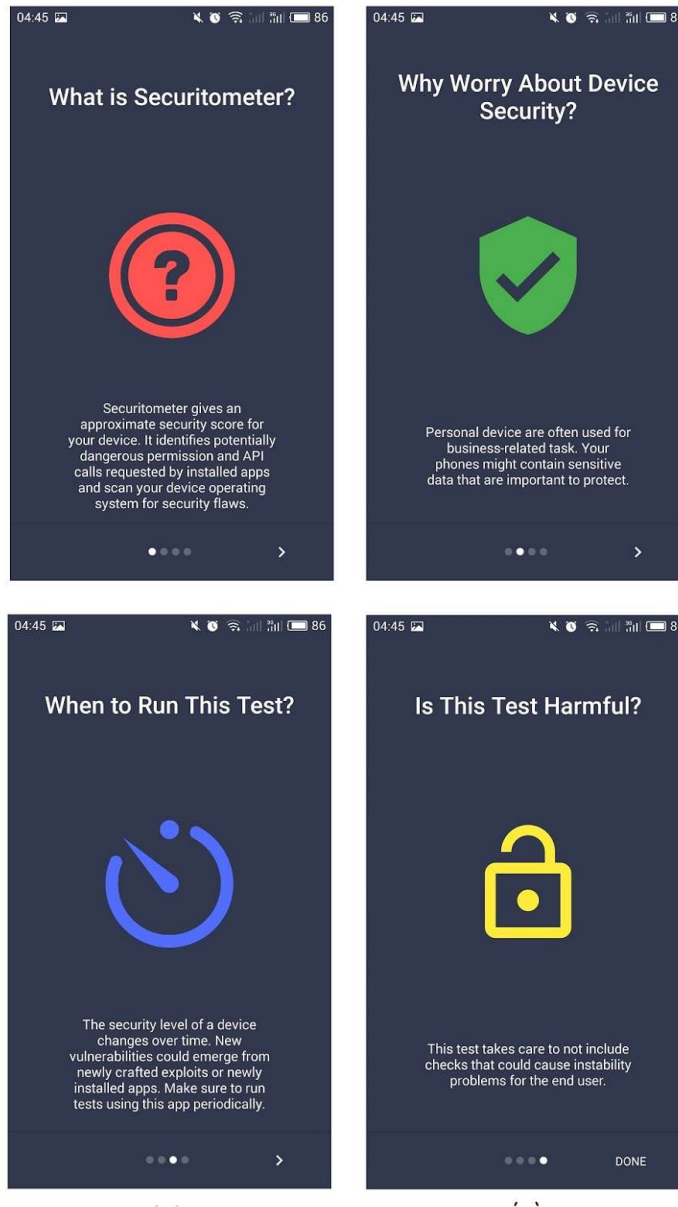
Before we start, you need to download the APK file and installed it on your device first.

Download here: <http://bit.ly/2bmeRUQ>

You opened the app for the first time after installing it. The first page that loads is the following (splash screen):



Following the splash screen, four introduction pages are automatically displayed with the following order. Swipe gesture is used to navigate between pages. You click "DONE" to continue.



4. Do the introductory pages and splash screen explain the purpose of the Securitomer app? *

Mark only one oval.

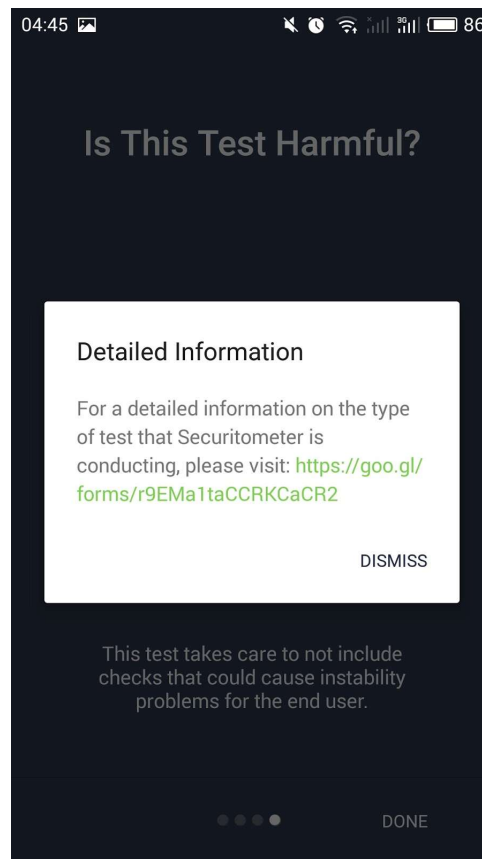
- ☐ Yes
- ☐ No

5. How well do you think you understand what the idea of the Securitometer app is? *

Mark only one oval.

- ☐ Do not know
- ☐ Do not understand at all
- ☐ Understand poorly
- ☐ Understand relatively well
- ☐ Understand very well

Before entering the main page, the application offers user a detailed read of what Securitometer is doing to their device including a link that refer to Securitometer documentation.



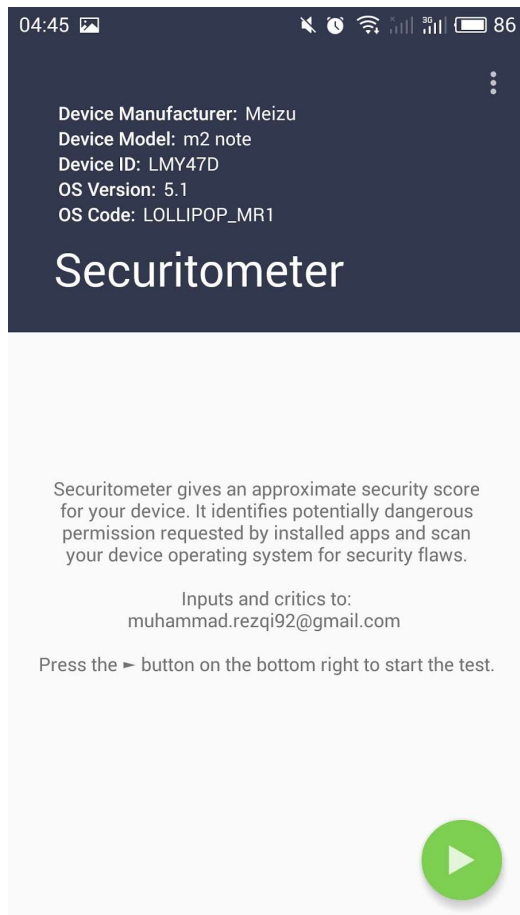
6. How well do you think Securitometer ensure that users are convinced that using Securitometer will not harm their device? *

Mark only one oval.

- ☐ Very poorly
- ☐ Poorly
- ☐ Adequately
- ☐ Well
- ☐ Very well

Initiating the Test

After the splash screen and introductory pages, the first page that loads up is the following. The information about your device including its model and manufacturer are displayed on top of the app. You press the start button on the bottom-right to initiate the test.

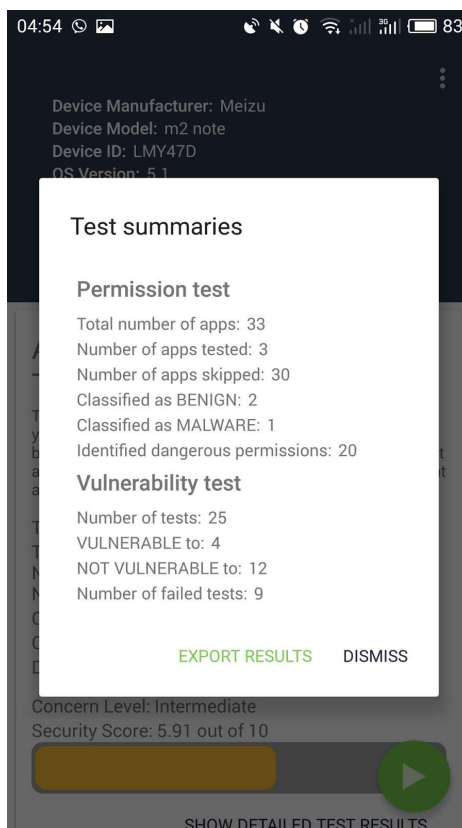
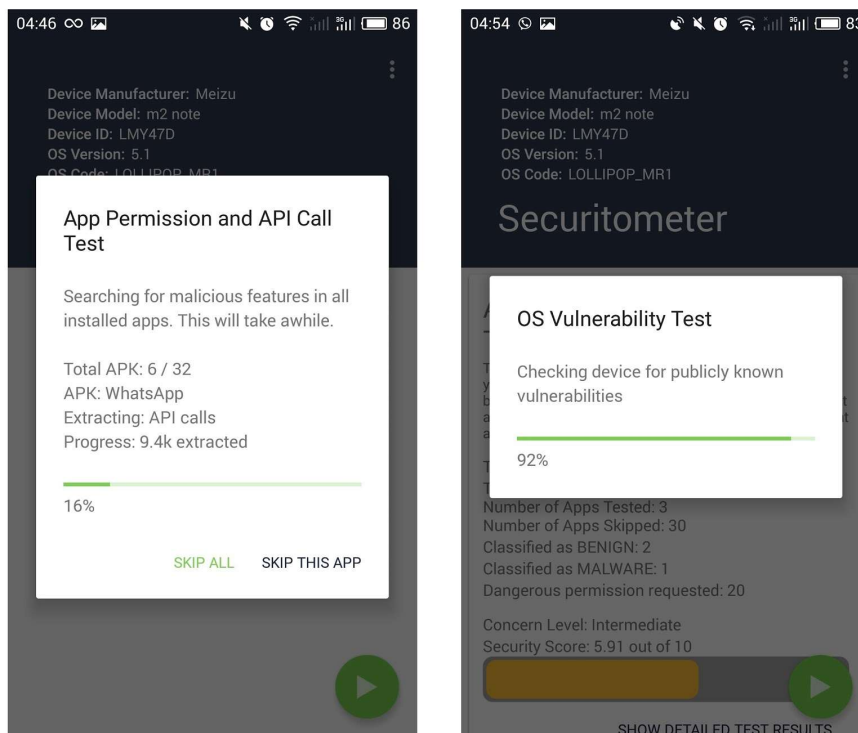


7. Is the device information helpful? *

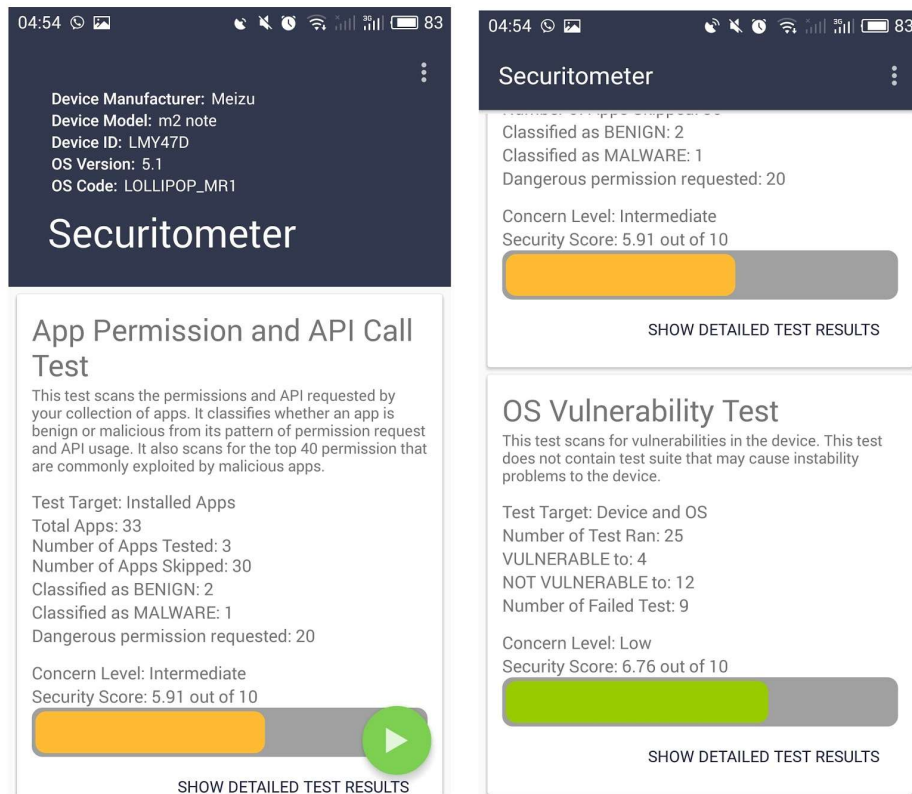
Mark only one oval.

- ☐ Not helpful at all
- ☐ Slightly helpful
- ☐ Moderately helpful
- ☐ Very helpful
- ☐ Extremely helpful

Then you click on the "START" button, which will display the following progress dialogs indicating that the test suite is initiated.



After a couple of seconds, the overall security score of the device will be displayed as follow:



The coloured test result bar will change colour according to the test score. It follows a traffic light colour scheme.

For a score of below 5.0, the colour would be red.
 For a score of 5.0 – 7.0, the colour would be yellow.
 And for a score of 7.0 – 10.0 the colour would be green.

8. How well do you think Securitometer overall result give an overview to the Security level of your device? *

Mark only one oval.

- ☐ Very poorly
- ☐ Poorly
- ☐ Adequately
- ☐ Well
- ☐ Very Well

9. Is the overall result adequate for the overview part of your technical report on the device's security level? *

Mark only one oval.

- ☐ Yes
- ☐ No

App Permission Test, Scoring Mechanism

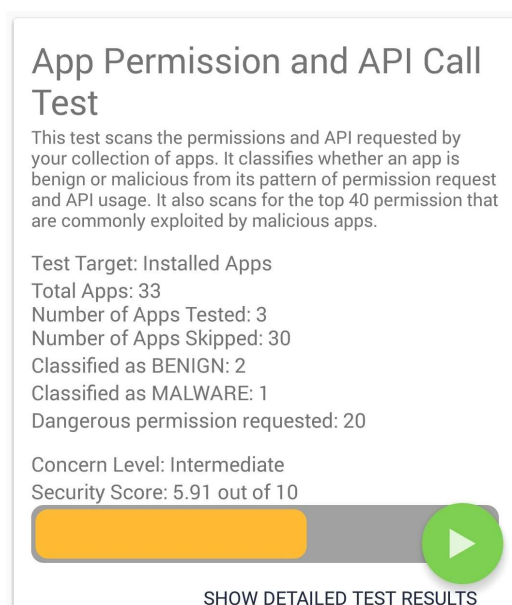
The app permission test first started with a malware classifier. If Securitometer classified an app as a potential malware, it immediately assigned the lowest individual score to it, which is 0. If the app is

classified as benign, Securitometer ran the app permission test to get the app's risk score.

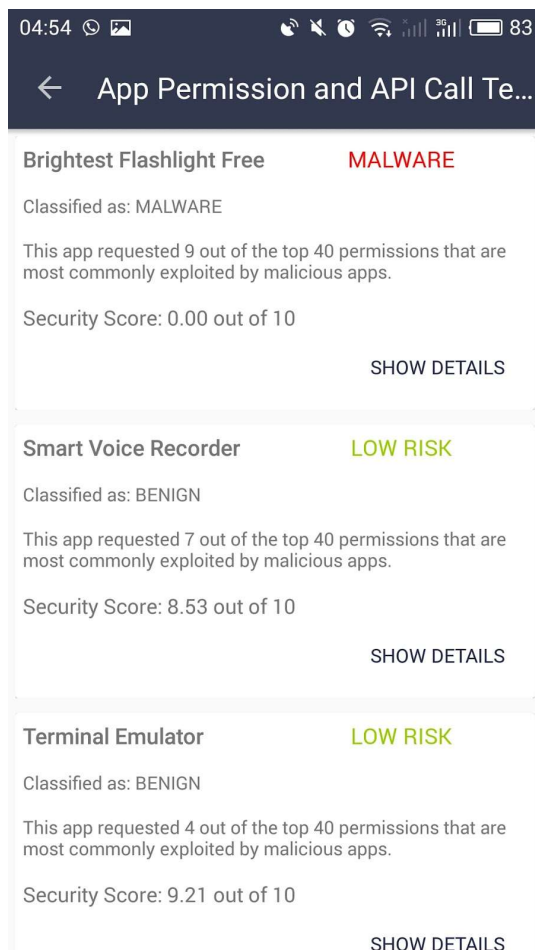
Risk scores are assigned by scanning through the app's list of requested permissions. We used a ranking of Android permission based on a study that compares its percentage of occurrence in malicious and benign Android apps. The top 40 riskiest permissions are listed in the ranking and given a score called risk level.

Permissions are the mechanism by which app developers disclose how their apps will interact with users' devices and personal information on devices running Google's Android operating system.

Dangerous permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps. For example, the ability to read the user's contacts is a dangerous permission. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app.



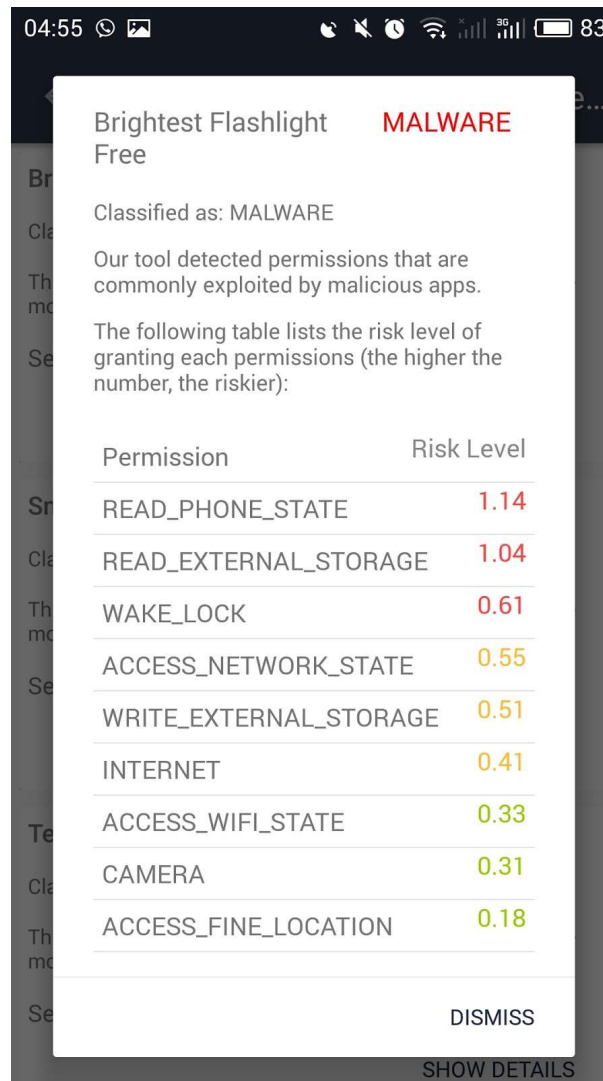
Now, you clicked the button "SHOW DETAILED TEST RESULTS" on the Application score card, and the following page loads up:



As explained above, the “Brightest Flashlight Free” app in the sample above is classified as Malware, thus getting a 0 score.

The "Smart Voice Recorder" app is considered benign thus is assigned score based on its request for permissions.

Now, if you clicked the “SHOW DETAILS” button, the following details will show up, elaborating the list of permissions that each app requested and granted and the risk level of each permissions.



The colour of the risk level will change according to its value.
It follows a traffic light colour scheme.

The higher the risk level, the colour will get closer to red.
The lower the risk level, the colour will get closer to green.

10. **Would you consider the information provided on the app permission score results, a helpful parameter in the evaluation of your devices' security? ***

Mark only one oval.

- ☐ Not helpful at all
- ☐ Slightly helpful
- ☐ Moderately helpful
- ☐ Very helpful
- ☐ Extremely helpful

11. Is the app permission test result adequate for the explanation part of your technical report on the device security level? *

Mark only one oval.

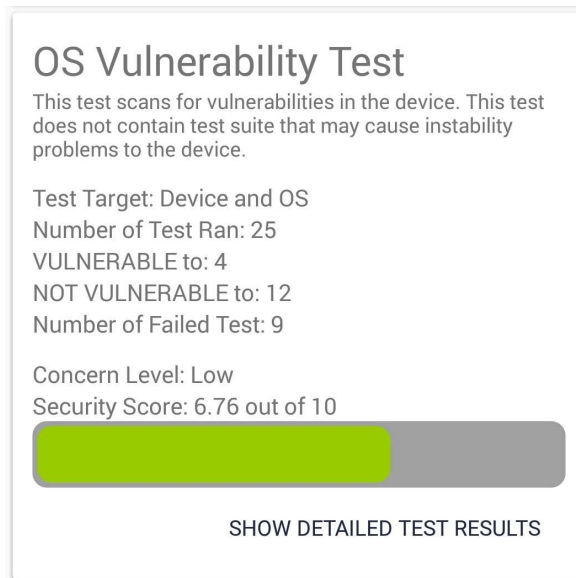
☐ Yes

☐ No

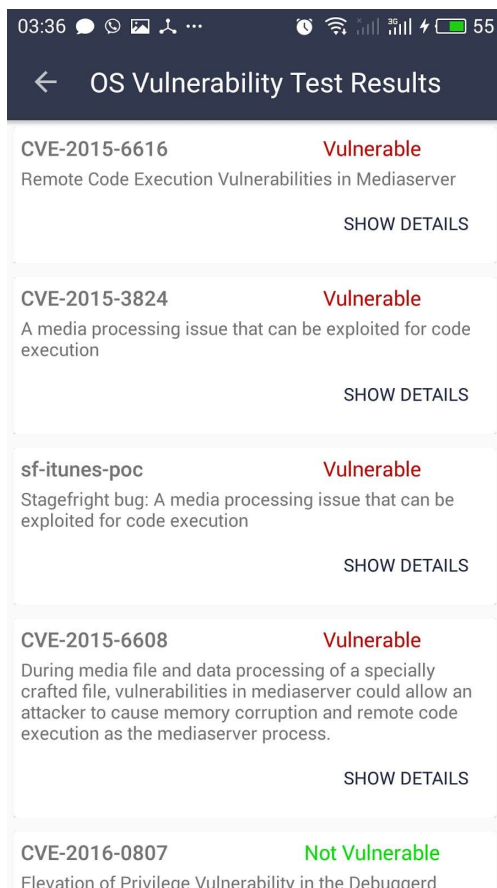
System Test, Scoring Mechanism

Securitometer assigns a device security level based on the CVSS score of each vulnerability identified within the device.

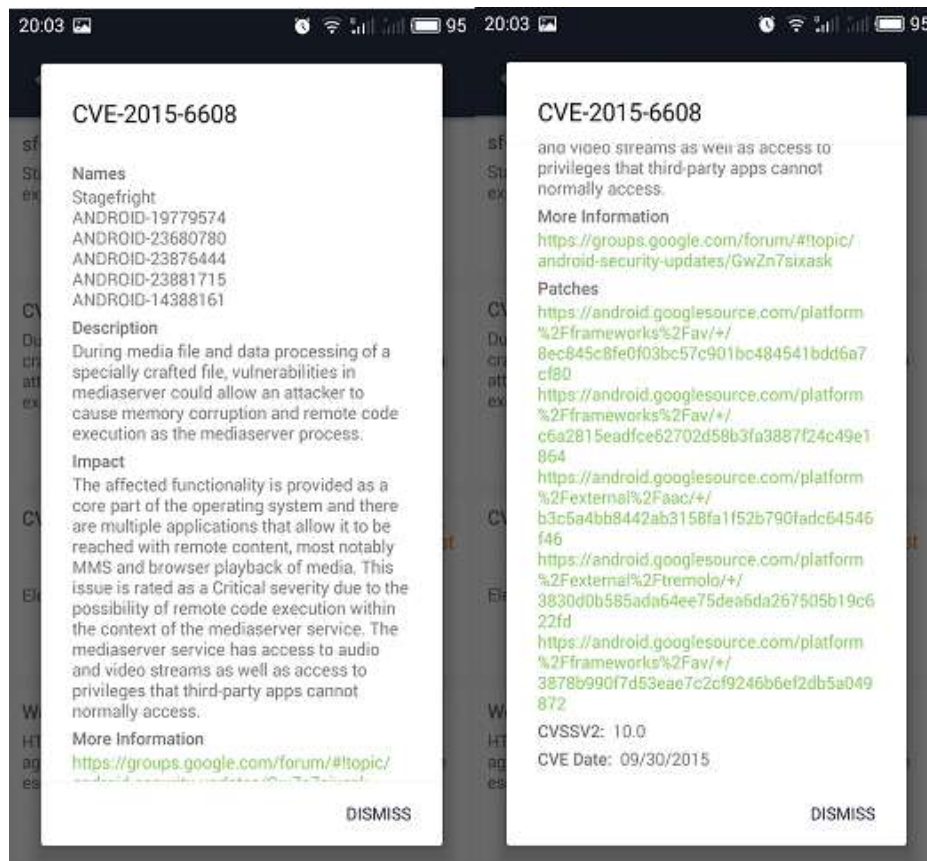
CVSS Score (Common Vulnerability Scoring System) is a common language of scoring IT vulnerabilities for IT managers, vulnerability bulletin providers, security vendors, application vendors and researchers.



You clicked the button “SHOW DETAILED TEST RESULTS” on the OS score card, and the following page loads up. Securitometer tries out several vulnerability checks on the device. The test results indicate whether or not the vulnerability is present within the device.



Clicking on the “SHOW DETAILS” button on each vulnerability, will display a vulnerability descriptor dialog. The dialog will contain information such as: vulnerability aliases, description, link for more information, and most importantly the CVSS V2 score.



12. **Would you consider the information provided on the OS Vulnerability test results, a helpful parameter in the evaluation of your devices' security? ***

Mark only one oval.

- ☐ Not helpful at all
- ☐ Slightly helpful
- ☐ Moderately helpful
- ☐ Very helpful
- ☐ Extremely helpful

13. **Is the OS Vulnerability test result adequate for the explanation part of your technical report on the device security level? ***

Mark only one oval.

- ☐ Yes
- ☐ No

POST TEST

14. Indicate whether you agree or disagree with the following statements? **Mark only one oval per row.*

	Strongly Agree	Agree	Slightly in agreement	Neutral	Slightly in disagreement	Disagree	Strongly Disagree
1) Overall, I am satisfied with how easy it is to use this app.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
2) It was simple to use this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
3) I could effectively complete the tasks and scenarios using this app.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
4) I was able to complete the tasks and scenarios quickly using this app.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
5) I was able to efficiently complete the tasks and scenarios using this app.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
6) I felt comfortable using this app.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
7) It was easy to learn to use this app.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
8) I believe I could become productive quickly using this app.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
9) The app gave error messages that clearly told me how to fix problems.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
10) Whenever I made a mistake using the app, I could recover easily and quickly.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
11) The information (such as on-line help, on-screen messages and other documentation) provided with this app was clear.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
12) It was easy to find the information I needed.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
13) The information	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

information provided for the app was easy to understand.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
14) The information was effective in helping me complete the tasks and scenarios.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
15) The organization of information on the system screens was clear.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
16) The interface of this system was pleasant.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
17) I liked using the interface of this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
18) This system has all the functions and capabilities I expect it to have.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
19) Overall, I am satisfied with this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Powered by



Bibliography

- [1] Android. Abi management. Available online: <https://developer.android.com/ndk/guides/abis.html>.
- [2] Android. Supporting different devices. Available online: <https://developer.android.com/training/basics/supporting-devices/index.html>.
- [3] Sam Bakken. Appealing google play's suspension of the vts for android app, Dec 2015.
- [4] Mario Ballano. Mobile attacks: Cybercriminals' new cash cow. Available online: <http://www.symantec.com/connect/blogs/mobile-attacks-cybercriminals-new-cash-cow>, Aug 2014.
- [5] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowddroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM, 2011.
- [6] Ngai Chee Ban. Is penetration testing recommended for industrial control systems?, 2013.
- [7] Wei Chen, David Aspinall, Andrew D. Gordon, Charles A. Sutton, and Igor Muttik. More semantics more robust: Improving android malware classifiers. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WISEC 2016, Darmstadt, Germany, July 18-22, 2016*, pages 147–158, 2016.
- [8] Steve Easterbrook. Lecture 4: Requirements elicitation, Aug 2016.
- [9] William Enck, Damien Ocateau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, page 2, 2011.

- [10] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245. ACM, 2009.
- [11] Patrick Engebretson. *The basics of hacking and penetration testing: ethical hacking and penetration testing made easy*. Elsevier, 2013.
- [12] Grant H Fenner and Robert W Renn. Technology-assisted supplemental work and work-to-family conflict: The role of instrumentality beliefs, organizational expectations and time management. *Human Relations*, 2009.
- [13] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale. In *International Conference on Trust and Trustworthy Computing*, pages 291–307. Springer, 2012.
- [14] M. Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26, Oct 2007.
- [15] Cedric Halbronn and Jean Sigwald. iphone security model & vulnerabilities. In *Proceedings of Hack in the box sec-conference. Kuala Lumpur, Malaysia*, 2010.
- [16] IDC Research Inc. Idc: Smartphone vendor market share, 2015.
- [17] Willy Jimenez, Amel Mammar, and Ana Cavalli. Software vulnerabilities, prevention and detection methods: A review¹. *Security in Model-Driven Architecture*, page 6, 2009.
- [18] Kaspersky Lab. System vulnerability and exploits, 2016.
- [19] James R Lewis. Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, 1995.
- [20] Steven Millward. 9 alternative android app stores in china, Mar 2016.
- [21] Keith Mokris. Android vts now on google play, Oct 2015.
- [22] Norton. Internet security threat report 2016. 2016.
- [23] NowSecure. Android vts. Available online: <https://github.com/AndroidVTS/android-vts>, 2016.

- [24] NIST NVD. Cvss v3 released. Available online: <https://nvd.nist.gov/CVSS/CVSS-v3-information.aspx>.
- [25] Samantha Rush. Problematic use of smart phones in the workplace: an introductory study. *BArts (Honours) thesis. Central Queensland University, Rockhampton* Available: <http://hdl.cqu.edu.au/10018/914191>, 2011.
- [26] Giovanni Russello, Arturo Blas Jimenez, Habib Naderi, and Wannes van der Mark. Firedroid: Hardening security in almost-stock android. In *Proceedings of the 29th Annual Computer Security Applications Conference, ACSAC '13*, pages 319–328, New York, NY, USA, 2013. ACM.
- [27] Matthew Schwartz. Android app ids smartphone, tablet vulnerabilities, Jul 2012.
- [28] Duo Security. X-ray for android. Available online: <https://github.com/duo-labs/xray>, 2016.
- [29] Mohamed Nassim Seghir and David Aspinall. *EviCheck: Digital Evidence for Android*, pages 221–227. Springer International Publishing, Cham, 2015.
- [30] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.
- [31] Sufatrio, Darell J. J. Tan, Tong-Wei Chua, and Vrizlynn L. L. Thing. Securing android: A survey, taxonomy, and challenges. *ACM Comput. Surv.*, 47(4):58:1–58:45, May 2015.
- [32] Daniel R. Thomas, Alastair R. Beresford, and Andrew Rice. Security metrics for the android ecosystem. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '15*, pages 87–98, New York, NY, USA, 2015. ACM.
- [33] Kevin Townsend. Virustotal policy change rocks anti-malware industry, May 2016.
- [34] James Turland, Lynne Coventry, Debora Jeske, Pam Briggs, and Aad van Moorsel. Nudging towards security: Developing an application for wireless network selection for android phones. In *Proceedings of the 2015 British HCI Conference, British HCI '15*, pages 193–201, New York, NY, USA, 2015. ACM.

- [35] Wade M Vagias. Likert-type scale response anchors. *Clemson International Institute for Tourism & Research Development, Department of Parks, Recreation and Tourism Management. Clemson University*, 2006.
- [36] Christian Vanek. What is a likert scale survey question & how to use it, Apr 2012.
- [37] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9(11):1869–1882, Nov 2014.
- [38] Ryan Welton and Marco Grassi. Current state of android privilege escalation, Oct 2015.
- [39] Lei Wu, Michael Grace, Yajin Zhou, Chiachih Wu, and Xuxian Jiang. The impact of vendor customizations on android security. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 623–634. ACM, 2013.
- [40] Sara Yin. Android malware reports overtake symbian, Aug 2011.
- [41] Yonghai Yu and Yun Bi. A study on x201c;5w1h x201d; user analysis on interaction design of interface. In *Computer-Aided Industrial Design Conceptual Design (CAIDCD), 2010 IEEE 11th International Conference on*, volume 1, pages 329–332, Nov 2010.
- [42] Z. Zhao and F. C. Colon Osono. x201c;trustdroid x2122; x201d;: Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking. In *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*, pages 135–143, Oct 2012.
- [43] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy*, page 95 to 109, May 2012.