TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technology

Department of Software Science

ITC70LT

Iryna Bondar 146085

LUDROID: EVALUATION OF ANDROID MALWARE DETECTION TECHNIQUES AND DEVELOPMENT OF A FIRST LINE DEFENSE SOLUTION

Master thesis

Emin Caliskan

PhD Researcher

Tallinn 2017

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Iryna Bondar

18.05.2017

Abstract

Due to the increasing popularity of mobile platforms based on Android, the number of software distributions increases correspondently. Along with this, the malware industry is also evolving with the time passing. The main Android application dealer (Google Play store) performs intrinsic check and examines the distributed software for malicious components occurrence. But the rest of the distribution sources (alternative Android markets) follow different policies for applications control. Users who choose alternative sources are more likely to expose themselves to the malware effects. To solve this problem researchers and studies started focusing on Android security relying on different aspects. While antivirus systems generally speaking are redundant and reduce battery life and device performance, the proposed academic countermeasures are currently not readily available for research or are not maintained anymore.

The main contribution of this work, besides evaluating and exploring the state of the art of Android malware detection tools and techniques, is a system which allows users to check the applications from third-party repositories without downloading the file and performing malware detection by means of multiple integrated cloud based solutions, called L u d r o i d. Based on the analysis report the user can decide whether to proceed with the download and therefore install the file, or cancel it. Ludroid aims, at least in theory to address in an easy way the threat represented by untrusted applications belonging to unofficial markets.

This thesis is written in English and is 54 pages long, including 5 chapters, 12 figures and 4 tables.

Annotatsioon

LUDROID: Android platvormi pahavara tuvastamistehnikate ülevaade ning pahavara esmase kaitse lahenduse arendamine

Androidi-põhiste mobiiliplatvormide suurenevale populaarsusele vastavalt kasvab ka Androidile mõeldud tarkvara kasutajate hulk. Ajaga areneb järgi ka pahavara. Peamine Androidi-rakenduste levitaja (Google Play pood) teostab rakenduste sisemist kontrolli ja püüab rakendustes tuvastada kuritegelikke komponente. Teised mobiilirakenduste levitamise kanalid (alternatiivsed rakenduste poed) kasutavad rakenduste kontrollimiseks muid strateegiaid. Alternatiivsetest allikatest pärit rakenduste kasutajatel on suurem oht sattuda pahavara ohvriks. Selle probleemi lahendamiseks on teadlased ja uuringud hakanud keskenduma Androidi-rakenduste turvalisuse erinevatele aspektidele. Kuigi viirusetõrje süsteeme on üldiselt palju ning nad pikendavad aku eluiga ja parandavad seadme võimekust, siis pakutud akadeemilised lahendused ei ole teaduslikuks uurimiseks vabalt kättesaadavad või neid enam ei toetata.

Selle töö põhiline panus peale Androidi pahavara tuvastuse tööriistade ja tehnikate ülevaate tutvustamise ja hindamise on süsteem Ludroid, mis lubab kasutajatel kontrollida alternatiivsetest allikatest pärit rakendusi ilma neid alla laadimata ning integreeritud pilvepõhiste lahenduste abil tuvastada pahavara. Selle analüüsi tulemuste põhjal saab kasutaja otsustada, kas rakenduse alla laadimise ja installeerimisega jätkata või mitte. Ludroid'i eesmärgiks on pakkuda vähemasti teoreetiline lahendus mitteametlikest rakenduspoodidest pärit ebausaldusväärsete rakenduste poolt kujutatava ohu lihtsaks adresseerimiseks.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 54 leheküljel, 5 peatükki, 12 joonist, 4 tabelit.

Table of abbreviations and terms

APK	Android Package
OS	Operating System
SQL	Structured Query Language
GL	Graphics Library
libc	Standard library for the C programming language
SGL	Scalable Graphics Library
SSL	Secure Sockets Layer
CPU	Central Processing Unit
API	Application Programming Interface
IMEI	International Mobile Equipment Identity
SMS	Short Message Service
VM	Virtual machine
JVM	Java virtual machine
DEX	Dalvik Executable format
JAR	Java Archive
UID	User identifier
PC	Personal computer
ICCG	Inter-Component Call Graph
GPS	Global Positioning System

URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
AV	Anti-Virus
GHz	Gigahertz
RAM	Random-access memory
DDR3	Double data rate type three
TB	Terabyte
SATA	Serial AT Attachment
IMSI	International mobile subscriber identity
MCC	Mobile country code
MNC	Mobile network code
LAC	Location area code
CID	Cell ID
WiFi	Wireless fidelity
MAC	A media access control address
APN	Access Point Name
PIM	Personal information manager
SD	Secure Digital
SHA1	Secure Hash Algorithm 1
PHP	Personal Home Page, a server-side scripting language

Table of contents

List of figures

List of tables

1. Introduction

Smartphones have become a crucial part of everyday human life and its usage is increasing exponentially. Android has become one of the most commonly used and popular operating systems for mobile devices [53]. Its popularity is partially induced by the immense collection of extensive smartphone applications in various official and third party mobile application markets. One of the significant abilities of Android operating system over other platforms is the ability to support third party applications that are offered through officious untrusted third party repositories and storages. This feature together with the tremendous vogue and user friendliness of Android system has made it highly attractive to malware authors targeting for information and identity theft [55].

1.1. Research problem

The applications in Android environment are mostly downloaded and installed through Google Play, an application market managed by Google, which performs malware checks for every application uploaded [60]. Despite the fact that some malicious applications have passed the tests and have been uploaded on the official market, this represents a corner case out of the scope of this research.

The situation changes drastically when a user enables the possibility of installing applications from other sources, downloading an apk file from the Internet. There are a lot of unofficial websites or markets where applications can be downloaded. For instance, Aptoide[1], one of the most popular third-party repositories, has 115 million unique users with more than 6 thousand applications available on the market [66]. Most of popular and trusted officious markets use the anti-virus systems integrated into their security systems. This does not guarantee safe downloads, since various markets might apply different filters once the anti-virus system flags the suspicious activity. After contacting the Security Department of Aptoide (Appendix 1), they confirmed that in their systems some anti-virus detections are automatically ignored, such as:

[1] http://www.aptoide.com/

“- Detections that have been confirmed to be false positives at all times;

- Potentially unwanted adware that is not extremely invasive […]. However, in such cases, the application immediately gets a Warning or Adware badge due to such detections;
- Detections for selected applications, such as Lucky Patcher and King Root. Most applications that root your device will be detected by a number of anti-virus systems, even though they are completely safe for the end-user.”

Aside from the most popular markets there are plenty of other less known third party repositories where uploaded applications are rarely checked or scanned for malicious activity and therefore they represent a perfect environment for malware hosting [56]. According to Cheetah Mobile security lab [7] in fact, the percentage of malware in unofficial repositories is significantly higher than the one on Google Play store (0.16% against 0.005%), and the percentage is even higher if small and unknown Android application markets are taken in consideration [22].

Fewer people prefer using antivirus applications on their devices [62], (Chapter 4.8). The solution offered by antivirus companies contains a security suite that manually scans every application, monitors traffic, may perform remote tracking or data wipe if the phone has been lost or stolen, backup for all files and data, and so on. These applications are assumed to be redundant and impairing device’s performance and are generally annoying with plentiful notifications and pop ups, so most of the users prefer not to use them and therefore don’t have any protection against malicious application, once they decide to use an application hosted on officious markets [65]. Besides, the studies of Android anti-virus applications effectiveness have shown that only 30% of chosen applications from official Android market were able to detect spyware installed or being installed [61].

Repackaged applications hosted on third party repositories are popular means for cybercriminals to hit new victims. To users, repackaged application may seem normal, but the truth is that malware authors take legitimate applications and add malicious code to them before distribution that leads to a range of unwanted behavior. Another problem related to malwares on Android devices is that most people don’t read the permissions for the applications they install [15]. Repackaged applications are almost always identical in appearance to legitimate versions but they often require more permissions than they need. When a user initiates the process of installing an application downloaded from an unofficial website, he or she has to physically tap on a notification to install the apk downloaded from a source outside the Google Play and is presented with the list of permissions that the application requests. Studies show that users do not pay attention or understand Android permissions while granting them to the application [15], and despite the request of sensitive permissions, such as access to contacts, local storage, microphone, camera, and location tracking, which might not be needed by those specific applications, they proceed with the installation rather than canceling it.

1.2. Research questions

The study will be guided by the following research questions:

- How much and why is Android targeted by malware authors?
- What are the existing solutions to eliminate malwares from Android devices?
- What are the limitations of the existing solutions?
- How to protect users from downloading and installing malware?

1.3. Objectives and scope

The aim of this project is to protect the users from downloading malicious applications, by performing an efficient malware detection at runtime before allowing them to download the application. With the above premises, an easy way to achieve this is to delegate the security check to another application, which will take care of the download just if and only if the tests are passed. Given this purpose, the main problem is that there does not exist a unique way or a standard way to detect malicious applications, and many different techniques exist and keep being developed. To fill this gap the author decided to combine several publicly available cloud-based solutions with different features sets in one single system. This application is mainly made of two parts: a clear and user-friendly front-end, and a back-end, which performs the malware detection and gives back a result of the scanned application to the user.

2. Background

2.1. Overview of Android OS

Android is a comprehensive open source, Linux based platform and application environment designed for mobile devices. The openness of system allows for a much larger number of devices to run the same applications and is beneficial for developers as well as for consumers. Android provides huge variety of tools and frameworks making mobile applications development quick and easy. Android is user friendly and allows users to customize and adapt their phones individually and according to their needs. For manufacturers, it is the complete solution for running their devices. Other than some hardware-specific drivers, Android provides everything else to make their devices work [1]. Figure 1 [1] depicts the main components of the platform.

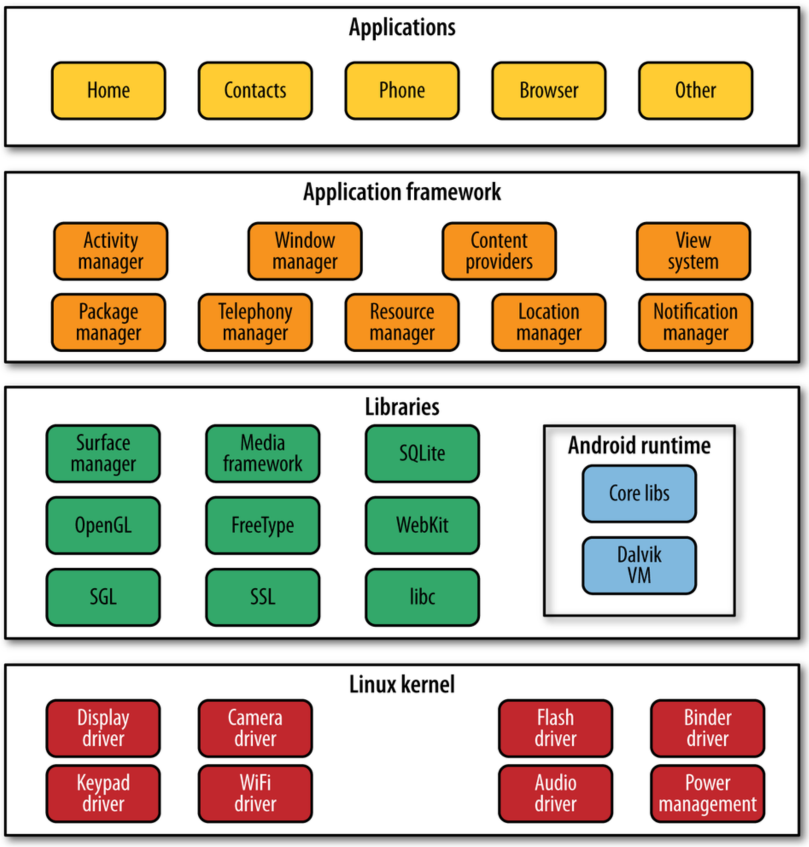


Figure 1. The Android software stack.

Android relies on Linux kernel to take advantage of key security features and allows developers to modify the kernel to fit their needs. The kernel is the first abstraction layer between the hardware and the rest of the software stack. It provides basic architectural model for process scheduling, support for memory management, resource handling, and networking.

Many core Android system components and services are built from native code that requires native libraries written in C and C++. Among others, they include:

“Webkit - a fast web-rendering engine used by Safari, Chrome, and other browsers;

SQLite - a full-featured SQL database;

Apache Harmony - an open source implementation of Java;

OpenGL - 3D graphics libraries;

OpenSSL - The secure sockets layer [1].

The libraries provide necessary services to the Android application level and play a vital role in optimizing the CPU and memory consumption.

The application framework is an environment that contains numerous Java libraries specifically built for Android. The entire feature-set of the Android OS is available through APIs written in the Java language. This layer provides numerous services designed to simplify the reuse of components including the following:

- View System – component allowing to build an app’s user interface, including lists, grids, text boxes, buttons, and even an embeddable web browser;
- Resource Manager – component providing access to non-code resources such as localized strings, graphics, and layout files;
- Notification Manager – component that enables all applications to display custom alerts in the status bar;
- Telephony Manager – provides device’s information like the IMEI number.
- Activity Manager - component that manages the lifecycle of applications and provide interface for the users to interact with the application;
- Content Providers - enable applications to access data from other applications, such as the contacts applications, or to share their own data [3].

Android phones come with a rich set of built-in applications, including email, SMS messaging, a Web browser, calendars, contacts, maps and more. The platform allows users to customize their phone replacing the built-in applications and so a third-party application can become the user’s default web browser, SMS messenger, or even the default keyboard [3]. All the applications reuse the same activity and this is an example of the system’s open design.

Applications are installed from a single application package with the .apk extension. Four main Android application components are:

- Activities: codes for a single, user focused task – an entry point for a user’s interaction with an application;
- Services: application component that can perform time-consuming operations in the background and doesn’t have user interface;
- Content providers: the standard interface for data fusion in the same process with the code that is running in another process. They encapsulate data and provide mechanisms to ensure their security;
- Broadcast receivers: messaging system across applications and outside of the normal user flow.

Additional Android application components:

- AndroidManifest.xml file: a file where all global settings are made controlling such components activities, services, broadcast receiver, content providers and intent filters. It also specifies which permissions are required;
- Intents and intent filters: messaging facility with which actions can be performed at the request of another component of the application;
- Fragments: fragment class is the behavior of the user interface or in operation (Activity class);
- Loaders: simplify asynchronous data loading in operation or fragment;
- Application Widgets: miniature application views that can be embedded in other applications [3].

Android runtime is the managed runtime used by applications and some system services on Android during the installation time. This component includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language and Dalvik virtual machine. “Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. During an application compilation, the Dalvik VM executes files in the Dalvik Executable (.dex) format that has been converted from the Java bytecode by the included dx tool in order to optimize minimal memory footprint and enhance multitasking ability”. [2] The usage of Dalvik VM is beneficial compared to other virtual Java-machines as it uses a special DEX format to store binary codes, not the JAR and Pack200, which are standard for other virtual Java machines, is optimized to run multiple processes simultaneously and uses a register-based architecture compared to the stack architecture in other JVMs, which increases the execution speed and reduces binaries size. Also it uses its own sets of instructions and allows to start several independent Android applications in one process [2].

2.2. Security measures

The Android security model is based mainly on permissions. In the Android model, each application runs as its own user account, meaning that, by default, all applications are separated and may access only their own data, not data from other applications. The system then applies a rigorous permissions system to services that are provided for use of installed applications. Permission is something that is granted to applications and required by APIs in order to run. In order to make use of services provided by other code on an Android device that may be sensitive or dangerous, such as accessing a user’s personal data or opening an Internet connection, an application must first request permission and be granted by the device’s user. Android uses an install-time permission request model, where an application specifies which of these permissions it requires in its manifest. On the installation phase the user can review the list of potentially dangerous things that the application is requesting to be allowed to do and has to approve them before the application is installed. This permission model consequently informs the user of what operations the application would be able to perform if the installation went successfully, and allows the user to make a decision whether to grant such permissions to the application and install it at all. This model has two primary advantages over traditional ones. First of all, before the application is being installed it brings to the attention of user all the dangerous things that application can do. Secondly, this permissions model allows constraining attacks on legitimate applications.

Applications inevitably contain coding problems and in many cases, skilled attackers exploit these errors to take over the running application and cause their own arbitrary code to run in the same context as the compromised application (with the same unique identifier (UID) and the same permissions) [9].

2.3. Classification of Android application malware

Malware is a program or file that is specifically designed to perform a variety of functions, including gaining access, encrypting, stealing and damaging sensitive data without knowledge and permission of a user. Smartphones have become an obvious target for malicious actors on the rise of mobile phones popularity. People started using their pocket devices more than PCs, carrying them all the time and using them for different purposes, starting with playing games and using multimedia programs, ending with personal conversations and confidential matters, like internet banking, financial transactions and storing sensitive personal data. Users are dependent on their mobile devices due to its feature rich applications and user friendliness. This makes smartphones more vulnerable to malware attacks and becomes the target for information and identity theft [10].

Gaining worldwide popularity, Android, being one of the newer operating systems targeting smartphones, has become one of the most popular mobile platforms, obviously attracting the attention of malware authors [53], [55]. Android relies on its security permission system and on the consumers' feedbacks to protect users against suspicious programs uploaded on Google Play Market. Unfortunately, users don't usually read required permissions and have no security consciousness before installing an application, thus provoking malware authors.

The classification of Android malwares based on their behavior and current attacks occurrence is presented in [11]:

a) Information Extraction

Having the right permissions, application can get access to user's personal data, contacts, browsing history, IMEI number, users' credentials and confidential bank details, thus compromising the device security, stealing and providing all the above mentioned data to malicious actors.

b) Premium Rate Calls and SMS

The malwares of this type masquerades as an application of another kind, such as a media player or a game and starts secretly making calls and sending SMS to some premium numbers after installation. The cost of these services is charged then to the sender’s phone bill.

c) Root Exploits

The malware gains system root privileges and takes control over the system being able to access and modify information without the user’s permissions and knowledge.

d) Search Engine Optimization

Artificial search of specific terms and simulated clicks on targeted websites compromised to boost the revenue of a search engine or increase the traffic on a website.

e) Dynamically Downloaded Code

Due to Android lack of security update patches this category of malwares can download malicious payload in form of benign application components, such as plugins extensions or updates and deploy them on the device.

f) Covert Channels

This type of malware is compromised by mobile phones’ vulnerability, allowing transfer of information between processes that are not supposed to communicate, thus originating information leak.

g) Botnets

A collection of several bots connected with each other with the help of Command and Control (C&C) networks, compromised by a botmaster. Botnets gain complete access to the device and its contents and provide the botmaster with root permissions over the compromised mobile device allowing malicious activities performance such as: sending e-mails or text messages, make phone calls, access contacts and photos, etc. Most of botnets act in covert and spread themselves by forwarding their copies to other devices using messages and e-mails [11].

The techniques for detecting and identifying these malware families on Android platform are described in Chapter 2.6.

2.4. Threat model

Each Android store follows different set of policies for policing applications. Android doesn’t provide any guarantees from the harm that third-party applications may cause the user. “The default setting of Android is that it does not allow its users to install applications from any source other than the official market, Google Play. The user has to enable the «Allow installation of apps from unknown sources» option from the Security settings screen to be able to install apps from unofficial sources [57]. This means that by downloading applications from unofficial stores, the user is installing applications completely at his/her own risk.

The easiest way to infect the device is to download the legitimate-looking application containing malicious code from third-party application stores [8]. In [58] the authors categorize three main social engineering-based techniques to install malware onto Android devices based on its internal activity pattern, namely: repackaging, update attack and drive-by download.

Repackaging is one of the most common techniques that malicious authors use to masquerade malicious applications as legitimate ones. To create a repackaged application a malicious developer downloads popular legitimate applications, disassemble them, make malicious changes, repackage and release them to alternative Android stores. By enclosing malicious payloads or simply inserting advertisements to the application, the repackaged applications bring the revenue to malicious authors once their applications are being widely downloaded and installed.

Update attack does not directly inject malicious payloads into benign applications. Instead, the malicious payloads are disguised as the “updated” version of legitimate applications. It is difficult to identify this kind of attack since it is often used by legitimate applications for the benign purposes as well, such as fixing bugs, upgrading installed games etc.

Drive-by download is similar to traditional web-based attack that is launched to redirect users to malicious or compromised websites with exploit codes that target mobile browser vulnerabilities [58]. Once the malicious application is downloaded, it runs transparently so that the user doesn’t see any suspicious activity. And once the malware has the access over the device, it may use its control of permissions to force it to download applications and tap on adverts to generate fraudulent advertising revenue potentially without the user’s knowledge.

2.5. Third party repositories

According to [4] Android is growing increasingly fast and has become the largest installed base of any mobile platform. It has the largest market share and hundreds of millions of mobile devices in more than 190 countries being sold around the world. The openness of platform allows developers around the world creating applications and games for users everywhere as well as distributing them in an open market. Google Play Store is the premier marketplace for selling and distributing Android applications [4].

Statista Inc. Figure 2 [5], has presented the statistic that shows the number of available applications in the Google Play Store from December 2009 to March 2017.

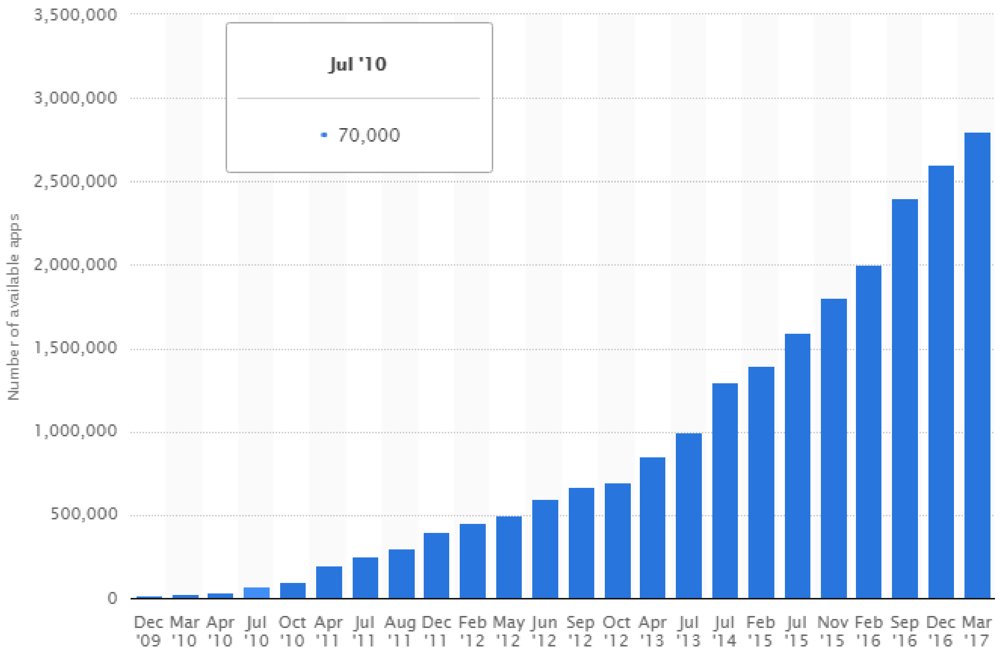


Figure 2. Number of available applications in Google Play Store from December 2009 to March 2017.

The number of available applications in the Google Play Store reached 1 million in July 2013 and was most recently placed at 2.8 million in March 2017. Google Play Store uses a security service called Bouncer [63]. This malicious application detection system automatically scans both new and existing applications and flags them as malicious when any anomalous activity is detected. Although by the time the malicious application is detected, it could have already made enough harm.

Moreover, in addition to the official market, there also exist a number of third-party applications stores which are popular and convenient because of the huge variety of applications which can be downloaded and installed for free, while being fee-based in

Google Play Store, as well as in countries where official applications are not available. A specific case can be Aptoide repository (Figure 3), which surpassed 3 billion downloads in 2016 with 1.5 million daily active users [66].

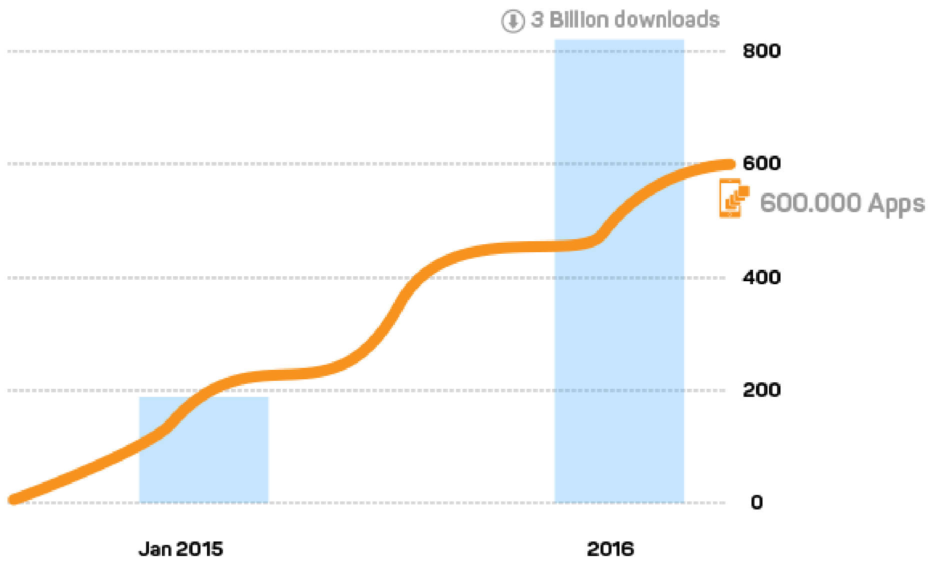
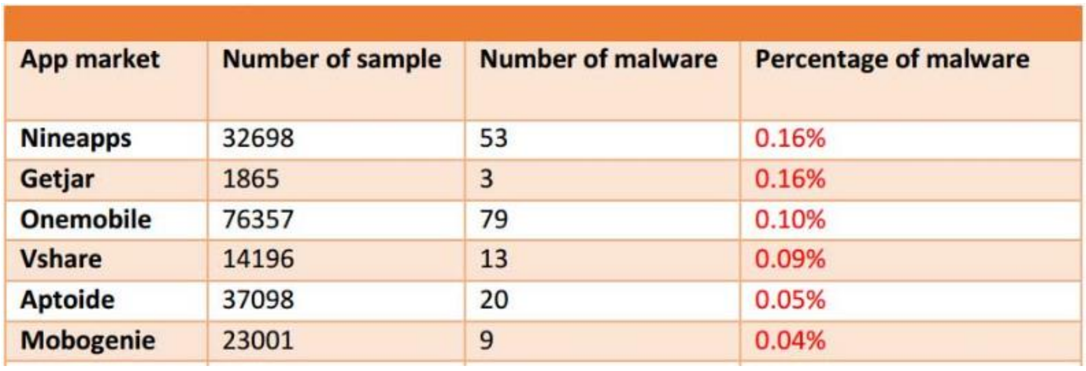


Figure 3. Number of available applications and their downloads on Aptoide in 1 year period.

Due to weak security monitoring, many third-party applications stores have been infected by malicious applications, usually pretending to be legitimate applications from top companies, being a great threat to users. The cybersecurity company Opswat has presented a research where they claim that almost a third of Android applications in third-party stores contain some form of malicious software [54].

Cheetah Mobile Security Lab [7] took samples from several well-known third-party Android applications stores and found numbers of malware families and their samples, starting from adware and ending with remote control which have been downloaded tens of thousands of times [7], affecting millions of users. Number of malwares and their samples in third-party applications stores are shown on Figure 4 [7].



App market	Number of sample	Number of malware	Percentage of malware
Nineapps	32698	53	0.16%
Getjar	1865	3	0.16%
Onemobile	76357	79	0.10%
Vshare	14196	13	0.09%
Aptoide	37098	20	0.05%
Mobogenie	23001	9	0.04%

Figure 4. Number of malwares and their samples in third-party applications stores.

In the report researchers have presented the top three markets containing malware-infected Android devices, namely: China, India and Indonesia. There are 1.5 million infected devices in China, 1.1 million in India and 800.000 in Indonesia. Russia has over 4.5 million infected devices and in fifth place, Malaysia. The report also includes the statistics of a malware massive increase up from 2.8 million in 2014 to more than 9.5 million in 2015 [7].

The authors in "Android Malware and Analysis" underline the threat of such repositories:

“Such sites or domains are dedicated to knockoff typosquatting-type domains and names related to popular games and software are very common in such markets.” [8]

Most of third-party repositories host pirated repackaged applications. Repackaged applications usually request more permissions than the original ones. When a user starts the process of installing an application, he or she is displayed the list of permissions that the application requests and all of the phone resources that the application will have access to if it is installed. And while granting those permissions users are most often not even aware of what an application will actually do with their data [15].

2.6. Classification of Android malware detection techniques

Based on the features used to classify an application, three different Android malware detection techniques exist: Static, Dynamic and Hybrid. Static analysis is done by extracting static features, such as permissions and API calls from the AndroidManifest.xml file and inspecting the downloaded application and its source code without running the application. Static detection techniques are classified as:

signature based ([36], [37], [38]), permission based ([24], [27], [28]) and Dalvik byte code analysis ([18], [32], [33]). The static analysis is usually very efficient in terms of performances, but not much in terms of detection, especially in case of obfuscation techniques the malware authors employ to evade from static detection techniques [12], and suits well as a first layer of detection, saving the server machine of the main application from running all other tests against a known malware. Several tools for static analysis exist, with focus on different aspect of Android applications as discussed in Chapter 3.

In contrast to static analysis, in dynamic analysis, the mobile application is executed in an isolated controlled environment such as virtual machine and emulator (Sandbox), to monitor the dynamic behavior of the extracted dynamic features of the application, such as network traffic, battery usage, IP address. By monitoring and logging every relevant operation of the execution, a report is automatically generated for each analysis. Dynamic detection techniques are classified as: anomaly based ([20], [41], [42]), taint analysis ([19]) and emulator based ([44], [45], [46]). Dynamic analysis can combat obfuscation techniques but can be circumvented by runtime detection methods and cannot respond to new malware families quickly [13].

To overcome the drawbacks in both approaches, hybrid analysis was introduced ([41], [44], [47]). The hybrid methodology involves combining static and dynamic features collected from analyzing the application and extracting information while the application is running, respectively [12].

Alongside the above mentioned techniques a large number of cloud-based solutions for identification of malicious content detected by antivirus engines and website scanners exists, and it is freely available for public use. The working principle of online scanners is the following: a user uploads any type of file, after scanning it the report about malware found is presented. All these systems thanks to reduced management effort greatly increase mobile protection.

At first, data inspection is performed by means of powerful cloud processors. Suspicious applications are evaluated using processors with far more compute power than ones on mobile devices. When a new malicious application is discovered it is being executed in the protected environment in order to determine what privileges would be requested from a mobile device and whether they would perform any other actions capable to compromise a device [64].

Secondly, cloud-based solutions support a worldwide community of users. As soon as the malicious application has been uploaded and analyzed by one user, its report will remain in the database available for public view.

Thirdly, information about scanned files is stored in the virtually unlimited storage with a maintained dataset far larger than any mobile device can support [64].

3. Related work

The analysis and detection of Android malware has been a broad area of research in the last years. Researchers have proposed several concepts and techniques to detect and analyze Android malware. Described below static, dynamic and hybrid malware detection techniques are taken in consideration during the development of this project.

3.1. Static analysis

Feng, Yu and Anand, S. designed Apposcopy [14], a static analysis technique based on the combination of a program representation called Inter-Component Call Graph (ICCG) during the first phase, that depicts main Android application components like activities, services, broadcast receivers and content providers (Chapter 2.1), and a static taint analysis as a second phase, that is capable of exposing applications that leak private user information.

Daniel Arp et al. proposed Drebin [16], a method capable for identifying malware on smart phones directly by performing a broad static analysis of all the application features gathered and applying machine learning techniques to classify the applications as benign or malware. Another example for static analysis is Androguard by Desnos et al. [18], which decompiles the application and applies signature based malware detection to identify cloned applications.

Adroit [34] is a combination of text mining and machine learning approaches is applied over the meta-information extracted from the manifest file, which includes developer data, the permissions or the description of the application to detect malicious applications. Similar approach to Adroit is used in the CHABADA [35] framework with the difference that here the static analysis is performed by applying clustering techniques over the extracted metadata.

Several signature based technique were developed. Among them AndroSimilar [36], a static analysis framework that is capable to detect unknown variants of existing malware samples that are usually generated by using repackaging and code obfuscation techniques, DroidAnalytics [37], a system which can automatically collect malware, generate signatures for applications and identify malicious code, RobotDroid [38], a lightweight mobile framework that is capable of distinguishing between benign and malicious applications of the same name and version, detecting anomalous behavior of known applications.

Although signature based techniques are very efficient for known malware, they are not able to detect unknown malware types and due to limited signature database most of the malware families are not detected [39].

A number of permission based detection techniques are available. For example, Kirin [24], a static analysis tool that defines security rules and checks the permissions of applications for indications of malicious activity. If an application fails to pass the security rules at the install time it’s being reported as malicious**.** Similarly, Stowaway [25] determines the set of API calls that an application uses to detect overprivileged applications and RiskRanker [26] estimates the potential security risks from untrusted applications by analyzing whether these applications carry any dangerous behaviors. Another set of permission based mechanism PUMA [27] and MAMA [59] were developed to detect malicious applications through machine-learning techniques by analyzing the extracted permissions and Vetdroid [28] that reconstructs sensitive behavior in the application from the permission use behavior.

Despite permission based tools are quick in performing application scanning and identifying whether the application is benign or malware, it only analyzes the manifest file leaving other files which may contain malicious code untouched [39].

Among Dalvik Bytecode Analysis techniques, where Dalvik executable files of the application are being statically analyzed, the following solutions are worth mentioning: Androwarn[68], SCANDAL [29], that determines the data flow from information source and detects the privacy leakage in applications, DroidMOSS [30] extracts the Dalvik Byte code sequence and developer information of application to detect the repackaged applications, built upon Androgaurd [18] DroidAPIMiner [31], identifies the malware by tracking the sensitive API calls, dangerous parameters invoked and package level, ComDroid [32] that is able to detect the communication based vulnerabilities among Android applications by dynamically observing interactions between the Android components and Flowdroid [33], a static taint analysis system for

Android applications that analyzes both application byte code and configuration files with very high recall and precision.

Dalvik Bytecode Analysis is performed on a higher level that will obviously consume more power and storage place. Since the android devices are resource-intensive, this approach remains flawed [39].

3.2. Dynamic analysis

Dynamic detection techniques are classified as: anomaly based, taint related and emulator based. In anomaly based detection in order to classify the application it is being continuously monitored at the kernel and user level. In pBMDS [40], MADAM [41], Andromaly [42] and Maline [43] after monitoring various features and activities obtained from the device, machine learning techniques are applied to detect anomalous application behaviors and classify them as harmless or hostile. CrowDroid [20] is a lightweight client that analyzes smartphone application activity by monitoring system calls of running applications and sending them to a centralized server. The server then performs behavioral analysis to classify the applications as malware or benign.

Enck et al**.** proposed TaintDroid [19] that enables dynamically monitoring applications in a protected environment focusing on taint analysis. It marks data simultaneously tracked from multiple sensitive sources such as GPS, camera, microphone and other phone identifiers and monitors all network interfaces for sensitive data leaks.

On the other hand, emulation detection is based on the execution of the application in an isolated sandbox environment to analyze low level interactions with the system. Among such techniques the most popular are: Droidbox [44], an extension of TaintDroid [19] that records application behavioral in file operations, SMS and phone operations, cryptography operations and network traffic monitoring to identify sensitive information; Droidscope [45], a dynamic analysis tool that can be used to reconstruct both OS-level and Java-level components of Android Applications. On top of DroidScope, the researchers have developed several analysis tools to collect detailed native and Dalvik instruction traces, profile API-level activity, and track information leakage through both the Java and native components using taint analysis [45]. Another proposed emulation based technique is CopperDroid [46], a system call based analysis framework that monitors inter-process communication and reconstructs the behavior of the Android applications.

3.3. Hybrid analysis

Since both, static and dynamic techniques have limitations, both academic and industry research has focused on hybrid analysis techniques. Application Sandbox (AASandbox) by Bläsing et al.[21] was the first framework combining static and dynamic analysis. It first extracts the .dex file and then performs static analysis scanning the software for malicious patterns without installing it. In the dynamic analysis the application is executed in the Sandbox with the traced system calls and corresponding reports for the further analysis. Another system combining static and dynamic analysis is DroidRanger [22] where two techniques based on application permission analysis are applied both for unknown and known Android malware; the first proposed scheme is permission based behavioral footprinting for detecting malware running against known malware samples, the second is heuristics-based filtering for identifying inherent behavior of unknown malicious families. In Mobile Sandbox [17] firstly the application's manifest file is parsed and the application is being decompiled to better identify suspicious code; after that the check on suspicious looking permissions or intents is performed, and in the dynamic analysis the application is being executed in order to log all the performed operations. The developers have declared that the system is currently unavailable and it's not certain that there will future development.

Andrubis [23] is a web-based interface framework similar to Mobile Sandbox [17]. In their approach researchers also used Droidbox [44], TaintDroid [19] and Androguard [18] for fully automated analysis. Although the Andrubis approach is limited to applications beneath API level 9 (Android 2.3), at the time when Mobile Sandbox supports up to API level 17 (Android 4.2). Another tool with the similar performance and background is DroidAnalyst [47] that has been released recently and therefore supports devices with higher API versions for which the APKs are developed by third party developers and is more sophisticated in detection analysis environment-reactive malware [48]. A couple of other hybrid analysis techniques has been introduced, such as: Androinspector [49], DRACO [50] and on device hybrid analysis techniques MONET [51] against transformation attack and Marvin [52].

3.4. Limitations

The problem of usability of the above mentioned solutions remains urgent. Many proposed techniques require installing and maintaining components by command line and many of them are hard to install even for advanced users. Another limitation of the proposed high level solutions is that some of them just perform an analysis of the application and not the malware detection. Although they provide information about the application, such as permissions, API calls, network traffic, etc. they don’t define a criteria to conclude whether the application can be considered as malicious or not. For this research thesis author tried to get/use most of the techniques described above and the obtained results are presented in Table 1 below:

Table 1. Availability of existing malware detection techniques.

Approach	**Name**	**Availability**	**Last update**
Static	Apposcopy	No source code	-
	VetDroid	Source code available	September 7, 2015
	Adroit	Source code available	April 4, 2017
	Drebin	No source code	-
	Androwarn	Source code available	March 23, 2013
Dynamic	CopperDroid	No source code	-
	Droidbox	Source code available	September 25, 2015
	Maline	Source code available	May 6, 2015
Hybrid	DroidRanger	Not maintained anymore	2012
	Mobile Sandbox		2012
	AASandbox		2010
	DRACO	Source code released, API not available	June 16, 2016
	Monet	No source code	-

Although there are plenty of tools, very few are effectively ready and can be used, while most present the technique and a proof of concept but are still far from being industry grade.

4. Project

In essence, the Ludroid system is made of two parts: a front-end (Figure 5), which is aimed to be straightforward and easy, where the user can paste the link of the application he/she wants to download:

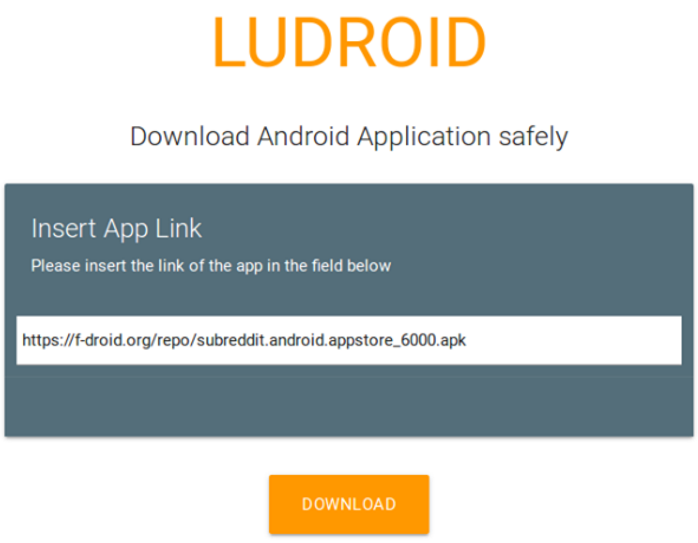

LUDROID
Download Android Application safely
Insert App Link
Please insert the link of the app in the field below
https://f-droid.org/repo/subreddit.android.appstore_6000.apk
DOWNLOAD

Figure 5. Front-end of Ludroid.

and a back-end which performs the following sequentally: receives the URL for the specific APK from the front-end and proceeds to download it to the temporary local storage of the server. First, the hash of the .apk file is computed (SHA1) to identify univocally the application, then a set of scripts are started and these implement the client side of the public API for the cloud based tools described in Chapter 4.3. The same scripts then take care of collecting the reports, or some specially interesting parts of them, from the above mentioned tools again using the API and saving them.

OS specification

Each Android device supports exactly one unique Android platform version API level. The API level identifies the version of the libraries that the application can call, the combination of manifest elements, permissions, etc. This system of API levels helps Android to determine whether an application is compatible with an Android system image prior to installing the application on a device. When an application is built, it contains the target API level of Android that the application is built to run on and the minimum API level of Android that is required to run the application.These settings are used to ensure that the functionality needed to run the applicaion correctly is available on the Android device at installation time [4].

The Ludroid system is currently built employing cloud based detection techniques which don't require application's sandboxing or in-depth analysis while scanning. The detection is performed by comparing the application's signatures with the ones existing in the signature database of the antiviruses used by the cloud tool (i.e. VirusTotal).

Because of the above mentioned premises there is no need to establish a certain API level as a target for Ludroid at the present moment.

The rest of the Chapter is organized as follows: Chapter 4.1 introduces the approach used in the project; the use case is described in Chapter 4.2; in Chapter 4.3 the selection criteria for the project techniques is explained; the author performs evaluation test of the applications collected from unofficial repositories in Chapter 4.4; description of the Dataset is outlined in Chapter 4.5; Chapter 4.6 features the software components description; Chapter 4.7 shows the techniques performance; decision making is described in Chapter 4.8, the results of a survey on users behavior of mobile applications installation are outlined in Chapter 4.9 and finally, the results are presented in Chapter 4.10.

4.1. Approach used in the project

For this project a quantitative approach was used by collecting Android malware samples of different families. Once the data was collected to the database, it was analyzed with different malware detection techniques in order to check the accuracy. The malwares which form the dataset were downloaded in bulk without being specifically selected or categorized by families, types or risk levels.

This approach is also applied to other processes in the project, such as: measuring the malware occurrence in third-party stores and evaluating the malware detection techniques.

4.2. Use case

The server hosting Ludroid has been rented from Online.net[1].

Server specifications:

- Ubuntu 14.04 64bit;
- Intel C2750 2.4GHz;
- 16GB RAM DDR3;
- 1TB hard SATA drive;
- 2.5Gb/s Connectivity.

Most of the tools have been previously tested on a local machine.

Local machine specifications:

- Lenovo Y50-70;
- Intel i7 4710HQ @2.5GHz;
- 8GB RAM DD3;
- 1TB hard SATA drive;
- Linux Mint 17.1.

4.3. Selection of techniques

The Ludroid users should be able to exclusively rely on it. Therefore the assurance of the most accurate detection performance is one of the biggest priorities.

[1] https://www.online.net/en

Ankita Kapratwar from San Jose State University in her research [67] has analyzed the effectiveness of combining static and dynamic techniques for detecting Android malware using machine learning techniques. After performing the evaluation experiment the thesis author concluded that combining analysis techniques is more efficient than using them independently.

For this reason it was decided to integrate both static and dynamic malware detection techniques simultaneously with cloud-based detection solutions in Ludroid. The tools were chosen on the basis of their efficiency and availability.

Firstly, the thesis author tried to use Androwarn [68], a static code analyzer of the bytecode targeting different malicious behaviors categories. After running the tool it performed structural and data flow analysis of application behaviors, collecting information such as: telephony identifiers leakage (IMEI, IMSI, MCC, MNC, LAC, CID, operator’s name), device settings (software version, usage statistics, system settings, logs), geolocation information leakage (GPS/WiFi geolocation), connection interfaces information (WiFi credentials, Bluetooth MAC address), telephony services abuse (premium SMS sending, phone call composition), audio/video flow interception (call recording, video capture), remote connection establishment (socket open call, Bluetooth pairing, APN settings), PIM data leakage (contacts, calendar, SMS, mails), external memory operations (file access on SD card), PIM data modification (add/delete contacts, calendar events), denial of service (event notification deactivation, file deletion, process killing, virtual keyboard disable, terminal shutdown/reboot). Although the tool performs deep static analysis of the application’s Dalvik bytecode, it doesn’t provide any response about the nature of application, but rather analyzes its behavior providing an information background to perform detection with a custom implemented technique, which was out of the scope of this project.

Another static technique that was tested is Vetdroid [28], based on Androguard [18]. The tool targets mainly the permissions used by an application and reports sensitive permissions usage. The report contains simple API misuse static audit, manifest configuration misuse and custom API call analysis. Unfortunately many benign applications require sensitive permissions (writing access to storage, GPS etc.) and therefore the usage of this tool individually would create many false positives. And so, similarly to Androwarn, a custom implemented detection technique is necessary to decide whether the application in analysis makes legitimate use of all the required permissions or represents a threat.

Among dynamic analysis techniques Droidbox [44] was tested being one of the most used and popular. It uses a modified version of the Android emulator that enables tracking Android applications' activity, such as tainted data leaked out, SMS sent and network communications. After getting Android source code and applying Droidbox patches, the new emulator was launched. Although the emulator process was launched successfully, Droidbox code couldn't start the activity of the main application analyzed. From the documentation it is clear that the tool performs in-depth dynamic analysis of the application but doesn't conclude if the application is malicious or benign. An example of Droidbox report analysis retrieved from [69] can be found in Appendix 2.

Given these results the only compromise between availability and functionality is performing the malware check by means of cloud-based solutions. Among broad variety of publicly available tools the choice was made to use a Payload Security service VxStream Sandbox [70]. The technique uses hybrid analysis technology, possessing tremendous amount of capabilities to detect unknown threats independent of anti-virus signatures. Unfortunately the thesis author didn't get any reply on the premium free-trial request; and since the free service doesn't allow processing more than 30 unique files per month, this tool was discarded.

Further investigation has brought to the analysis technology Joe Sandbox, the service of Joe Security[1].The analysis technology includes static, dynamic, hybrid, simulation and virtualization techniques able to detect any malicious behavior, including obfuscated, non-executed or hidden code segments. Joe Sandbox executes the files in a controlled environment and monitors the applications behavior for suspicious activities. The report of the scanning includes structures and domains of the file, data about strings. Malicious behavior is determined by matching generic signatures. Integration of Joe Sanbox into Ludroid came out unsuccessfully due to current company's policy not allowing participation in academic projects.

[1] https://www.joesecurity.org/

Another feature rich malware analysis system was examined as an option, namely Cuckoo Sandbox[1]. Cuckoo provides detailed results about file nature, execution time, signatures, network activity etc. by executing the file in an isolated environment. After recent update the support for Android has not been added yet to the service, so this tool will be considered in the future work.

Among other publicly available cloud-based detection systems the ones worth mentioning are: VirusTotal[2], Andrototal[3], Jotti Malware scan[4] and NVISO ApkScan[5].

The most simple and popular among such resources is VirusTotal, which has an open API and requires nothing more than a registration on the site. VirusTotal supports scanning of more than 50 antivirus scanners. This list of antivirus software vendors is constantly supplemented by new scanners. The maximum size of the uploaded file is limited to 128 MB for a maximum of four files per minute. The execution time of scanning operation depends on the file size, connection speed and network load.

As part of the experiment, AndroTotal, a free service that scans suspicious APKs against multiple mobile antivirus applications was tested. Through the scanning process it was discovered that for the detection of malicious activity AndroTotal uses 4 antivirus engines (Dr. Web, Kaspersky Mobile Security, Norton Security Antivirus and Avast) already present in the VirusTotal engines set. This system was categorized as a subset of VirusTotal and therefore excluded from the project.

NVISO ApkScan, a comprehensive tool capable of performing both static and dynamic analysis on Android programs to automatically detect suspicious applications was also taken into consideration. Malware analysis report contains general information about the scanned application, depicts extracted static features, such as permissions, API calls, services etc. from the AndroidManifest.xml, as well as dynamic features of the

[1] https://cuckoosandbox.org/

[2] https://www.virustotal.com/

[3] https://andrototal.org/

[4] https://virusscan.Jotti.org/

[5] https://apkscan.nviso.be/

application, such as disk activity, network activity, cryptographic activity, information leakage and miscellaneous. The security department of the company agreed to cooperate with the thesis author and provide an API key with extended permission to satisfy the need to scan a large amount of files per day, but hasn't granted it yet. Once the API account is ready for use the system will be integrated to the Ludroid.

And finally, a malware scan Jotti, a free service that detects suspicious files with several anti-virus programs. While the developers maintained their own scanning system and API, they didn't design any novel detection mechanism. Jotti is fully supported by third party anti-virus products and aggregate their results.

4.4. Measuring the malware occurrence in third-party stores

Studies have shown that almost a third of Android applications hosted on third-party applications repositories contain some form of malicious software [7], [54]. To check the accuracy and relevance of the data the thesis author decided to perform an independent experimental evaluation of applications gathered from two different third-party stores. The collection of applications was made in the period of one month, March 2017, from two unofficial marketplaces, namely: F-Droid[1] and Apkrepo[2]. To select the candidate sites the author used the first pages of Google results for "free android applications market".

With the help of HTTrack tool[3] the author has downloaded a subset of the applications from each repository to the server. The number of applications from each marketplace is shown in Table 2.

Table 2. Number of applications hosted on third-party repositories.

Market	**Number of applications**
F-Droid	4029
Apkrepo	941

[1] https://f-droid.org/

[2] http://www.apkrepo.com/

[3] https://www.httrack.com/

All the applications were scanned with VirusTotal. The result of F-Droid samples scan is shown on Figure 6.

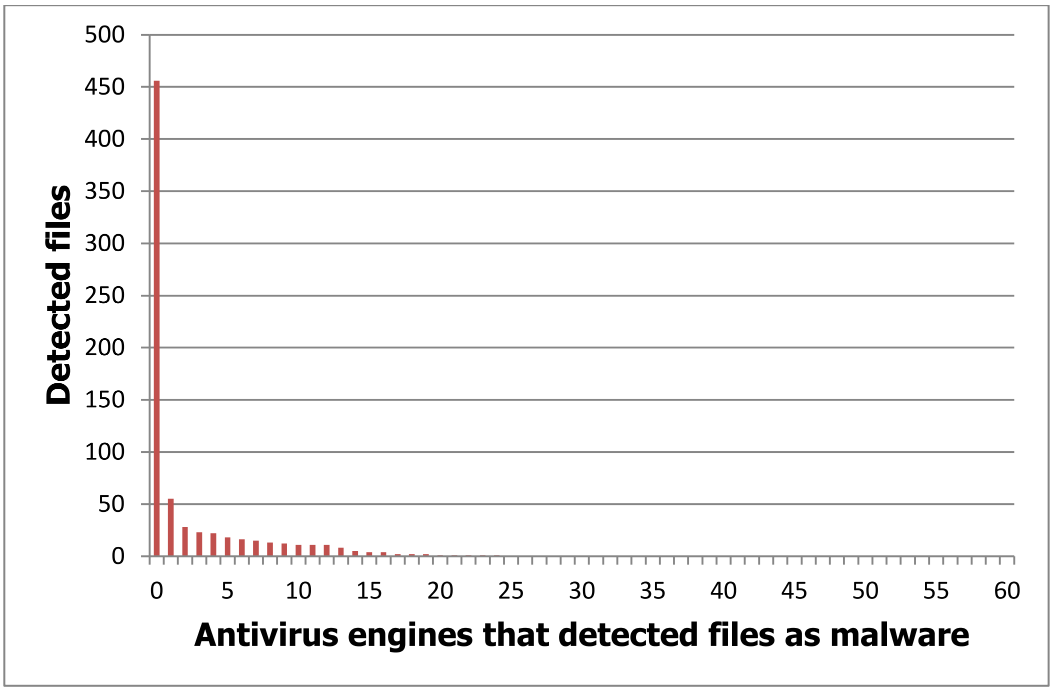


Figure 6. Result of F-Droid samples scan.

Based on the results of scanned files represented on Figure 6 it was decided to report applications as malicious on condition that they trigger five or more different antivirus engines, establishing an arbitrary despite reasonable threshold between false positives and real malware. The overall results of the scanning are summarized in Table 3.

Table 3. Results of scanned applications in third-party repositories.

Market	**Applications which triggered less than 5 AV**	**Applications which triggered 5 or more AV**	**Total detections (more than 0)**
F-Droid	434 (10.7%)	22 (0.54%)	456 (11.3%)
Apkrepo	111 (11.8%)	26 (2.7%)	137 (14.5%)

Considering that the selected applications were a random set, there is no reason to suspect that the obtained percentage is higher or lower than if testing the whole repository. Despite the fact that the criteria is not an accurate value, at least it seems to support the results stated by Opswat cybersecurity company and Cheetah Mobile security lab [7], [54] and affirm the need of the applications' scan at runtime before allowing users to download them.

4.5. Dataset

For the evaluation process the thesis author gathered a total collection of 434 unique malicious Android applications. Malware samples were collected from private repositories, namely “Collection of android malware samples”[1] by Ashish Bhatia and “Mobile malware mini dump”[2] by Mila Parkour. The collection is publicly available[3] and is currently being extended dependent on the availability of other private repositories which need special rights to be accessed.

4.6. Techniques performance

The general workflow of the techniques is described below.

4.6.1. VirusTotal Performance

To scan files with VirusTotal a slightly variated version of publicly available code[4] is used, after having obtained a personal API key.

The workflow is linear: first, an API call to the 'get' endpoint is performed, passing as parameter the SHA1 hash of the file in analysis. This is made mainly to save computation time in case the file has been scanned before.

If the file is present, the report is directly retrieved from the 'report' endpoint, if it is not, then the application is uploaded to the 'scan' endpoint.

After the application has been submitted, the report will be retrieved.

The VirusTotal report is quite detailed, an example can be found in Appendix 3.

From this report, some essential information is extracted; specifically the permanent links are kept and the total number of positive detections together with the total number of antiviruses used.

[1] https://github.com/ashishb/android-malware

[2] http://contagiominidump.blogspot.com.ee/

[3] https://github.com/tootsy42/Ludroid-dataset

[4] https://github.com/Gawen/virustotal

4.6.2. Jotti performance

For Jotti there was not any ready client written, so the author had to implement some basic functions on her own.

To scan files from Jotti, some steps are necessary after having obtained a personal API key.

First, with the same motivations valid for VirusTotal, the hash of the file is computed and a GET request to 'getfileinfo' endpoint is made to check whether the file has already been submitted and in positive case, the report for it is directly queried with a GET request to the 'getjobstatus' endpoint.

In case the file was not present, a ScanToken must be obtained, which can be used later to create a job for the specific file. After the job is created, the report is again obtained with a GET request to the 'getjobstatus' endpoint.

An example of report response from Jotti is reported in Appendix 4.

4.7. Decision making

The decision making process is implemented following the principle established in Chapter 4.10. Once the reports are generated, the last necessary step is to parse them and make a decision about the nature of the application and whether to warn the user or not before downloading it. Despite this process of collaborative decision would require a very in-depth research, the author decided to simplify it and establish that the user will be warned about the danger of the application if at least one of the tools used reports the file as malicious with a confidence (Chapter 4.10) of more than 20%. This arbitrary value is established on one hand to avoid many false positives, and on the other hand to keep it lower than the minimum confidence level ecountered in the evaluation made in Chapter 4.10. Eventually it would be possible also to let the user choose the level of strictness among a set of multiple choices (High, Medium, Low, Paranoid).

After the above mentioned process, the decision about the application is made and the back-end will input the result back to the front-end, which will display it accordingly and will eventually allow the user to download the application.

4.8. System description

Ludroid is running Apache2 web server with most of the website logic written in PHP.

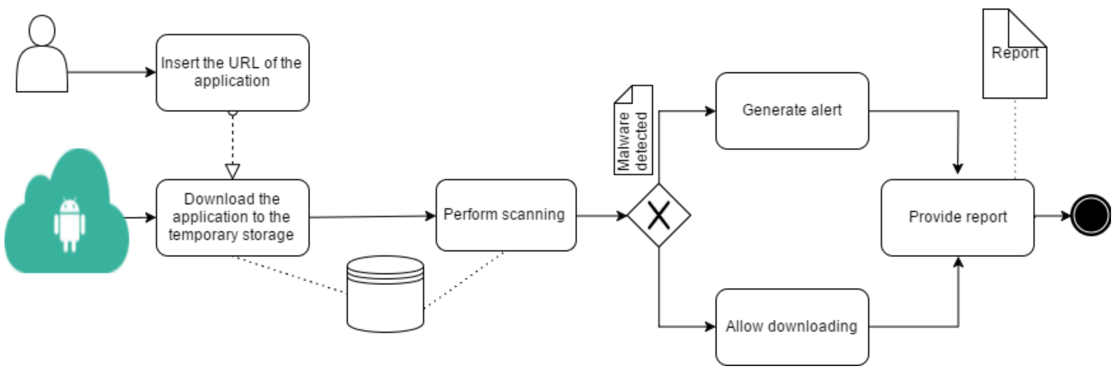
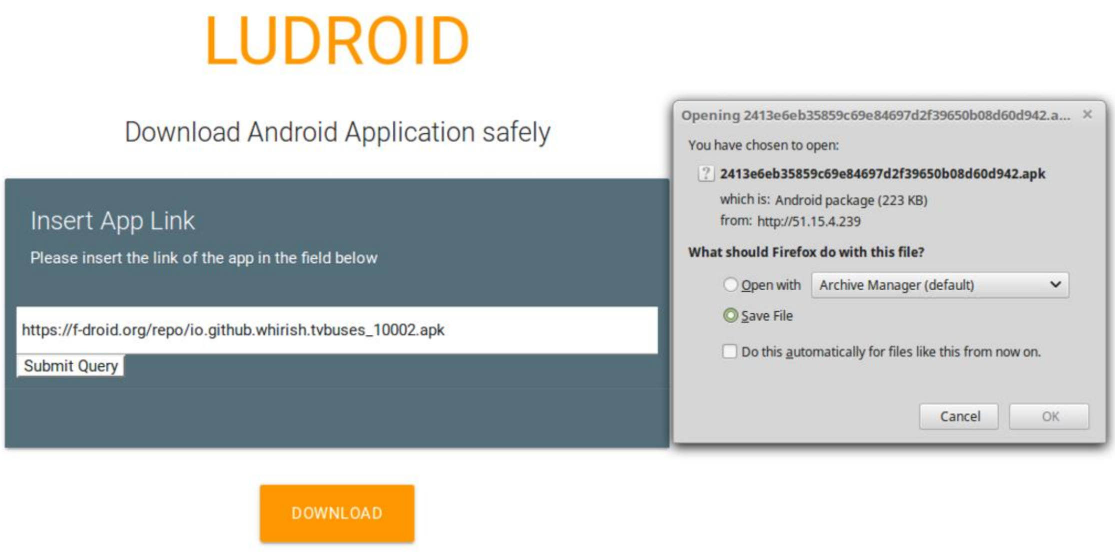


Figure 7. Application workflow.

The application workflow is represented on Figure 7 and is the following: the user opens the Homepage and inputs a link for direct download of the application; a POST request with the link as a parameter is sent to the server which, on its reception, calls a bash script passing it as a parameter. The bash script downloads the file, scans it with VirusTotal, saves it and its corresponding scan result in a directory named as the SHA1 hash of the file and renames the app as SHA1.apk, where SHA1 is the corresponding application hash and after returns the hash as a result to the php page. Thereafter the server checks the directory called as the hash and verifies the malware confidence. If the value of malware confidence is less than 20% it outputs the file directly to the user (Figure 8).



LUDROID

Download Android Application safely

Insert App Link

Please insert the link of the app in the field below

https://f-droid.org/repo/io.github.whirish.tvbuses_10002.apk

Submit Query

DOWNLOAD

Opening 2413e6eb35859c69e84697d2f39650b08d60d942.a...

You have chosen to open:

2413e6eb35859c69e84697d2f39650b08d60d942.apk

which is: Android package (223 KB)

from: http://51.15.4.239

What should Firefox do with this file?

Open with Archive Manager (default)

[✓ Save File]

Do this automatically for files like this from now on.

Cancel OK

Figure 8. Result of the scanned file with malware confidence less than 20%.

If the value of malware confidence is 20% or more, the server redirects the user to the Alert.php, that shows a message indicating that the scanned file is malicious (Figure 9).

THE FILE YOU TRIED TO DOWNLOAD WAS MALICIOUS

Figure 9. Result of the scanned file with malware confidence 20% or more.

4.9. Internet survey

The survey was conducted between Internet participants from author’s digital connections in the period of April 23 – May 18. At the present time 152 users took part in the questionnaire. The aim of the survey was to find out the reputation of anti-virus systems among consumers, to find out the statistical information about third-party repositories usage among respondents and to determine the need for a system such as Ludroid.

The web-based questionnaire consisted of 5 questions which asked respondents of three different age groups to evaluate the need of security mechanism when downloading applications from third-party repositories.

Among the 152 respondents, 47.4% users have downloaded applications at least once from unofficial sources. From Figure 10 it can be seen that 38.8% of participants answered they would use an application that would let them downloading applications from unofficial markets only after the application passes strict security checks; 27.6% respondents answered they wouldn’t use this system and 33.6% stated that they don’t know if they would use it.

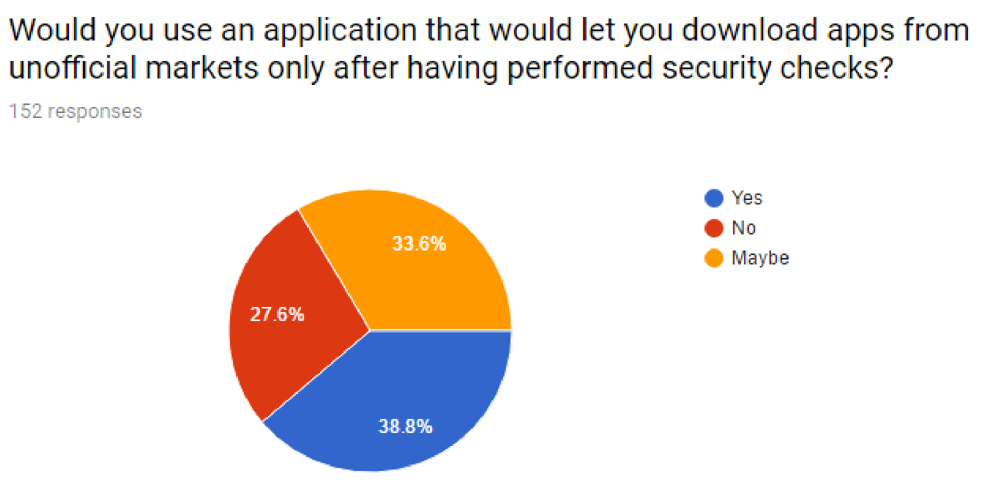


Figure 10. Evaluation of the need for a system such as Ludroid.

Analyzing the current trend in the survey, and integrating this with the information already gathered [62], it looks legitimate to conclude that many users are unaware of the threats that unofficial applications may cause. For this reason it makes sense to expand the project and embed it to the security system of third-party markets, ensuring that only benign applications will reach the users. Survey results for each question can be found in Appendix 5.

4.10. Results

The thesis author has established the value ***Malware confidence*** to have some term of comparison between used techniques and to ease the process of the decision making. Malware confidence (1) is defined as a ratio between systems reported the file as malicious and the total number of systems used.

$$Malware\ confidence = \frac{Positive}{Total} \times 100 \qquad (1)$$

Malware confidence distribution for VirusTotal and Jotti are depicted on Figure 11 and Figure 12 correspondently.

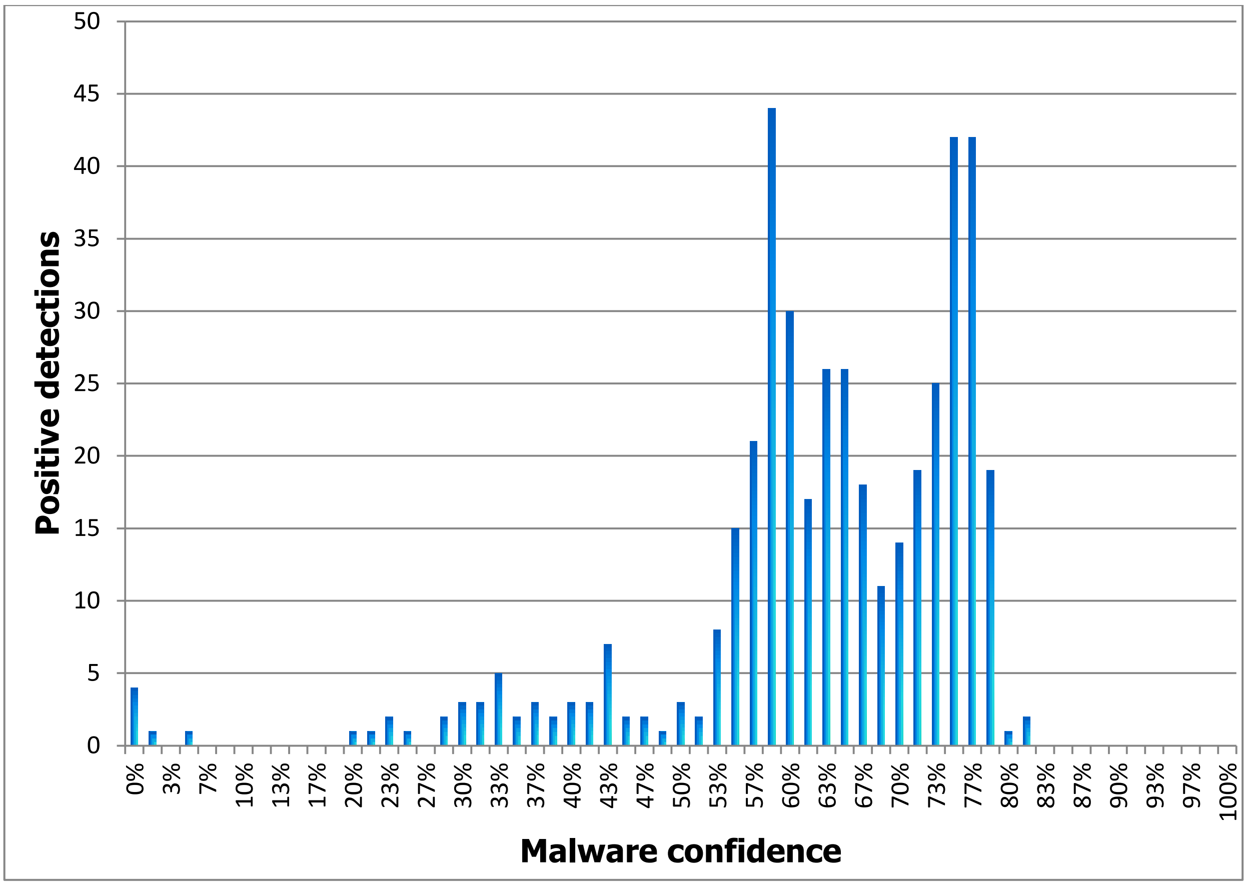


Figure 11. VirusTotal malware confidence distribution.

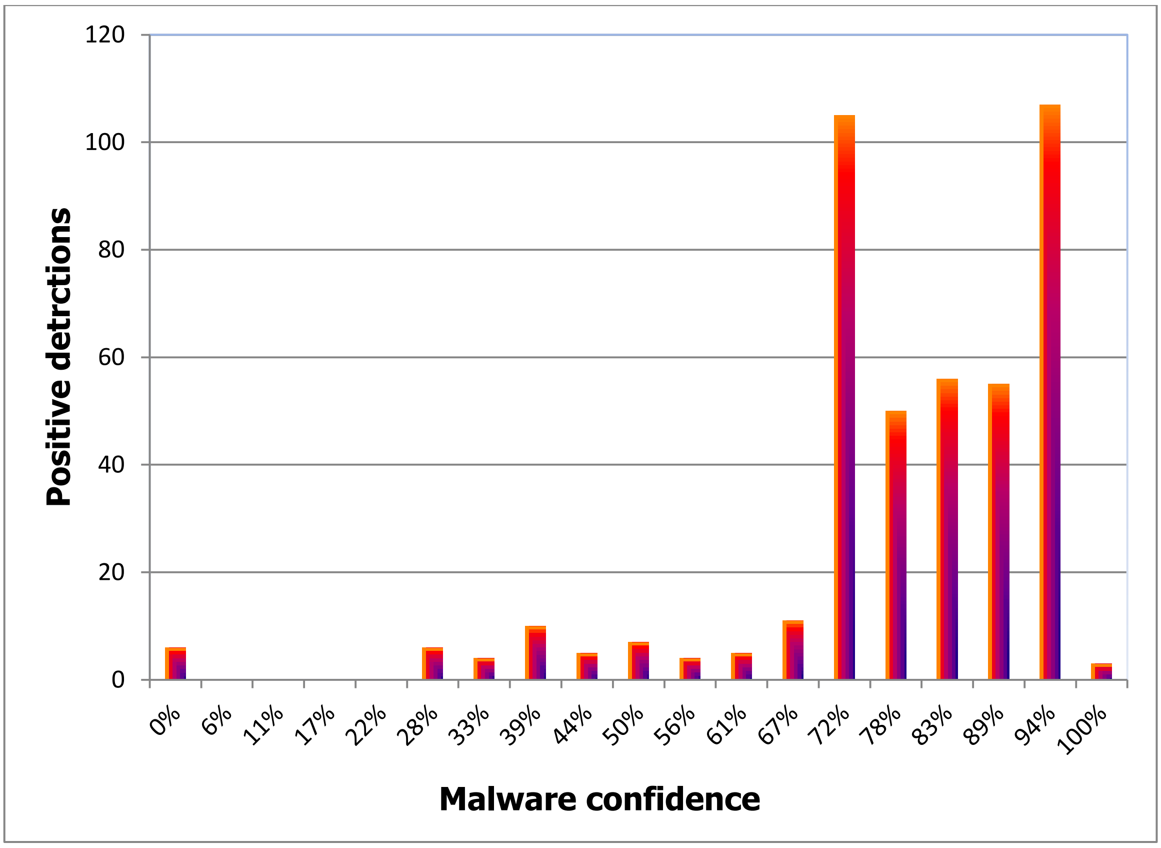


Figure 12. Jotti malware confidence distribution.

To estimate the average malware confidence (2) for every technique the total malware confidence is divided by the number of malwares:

$$Malware\ confidence_{aver.} = \frac{\sum malw.conf.}{TotalMalware} \tag{2}$$

The average value of malware confidence for both techniques is summed up in Table 4.

Table 4. Overview of techniques performance.

Technique	**Average malware confidence, %**
Jotti	77.6
VirusTotal	62.5

The result of the benchmark is overall similar, despite some minor differences that might put in evidence some characteristics. The average value of confidence, which represents the number of antiviruses that detected the malware compared to the total number of antivirus engines is generally higher for Jotti. This means that in general the selection of antiviruses used in Jotti looks quite accurate. From Figure 11 and Figure 12 it can be observed that the number of malwares with confidence level of 0% is higher in Jotti than in VirusTotal. VirusTotal didn’t detect four malwares while this number grows to six in Jotti and therefore Jotti seems to produce a slightly bigger number of false negatives (x% vs y%). Despite the fact that Jotti’s malware confidence is particularly higher for most of the detected malware and the execution time is lower due to the reduced number of systems used, VirusTotal still performs an overall better detection and it is preferred for the use in Ludroid.

The cloud-based detection techniques used in the project are signature based, where malware is detected by scanning and finding certain patterns that have been encountered before. Therefore, the malware has to be known by the scanning engines through a signature database, otherwise it won’t be detected. This limitation of the technique explains the false negatives for both systems used. However, the value of malware confidence estimated by the thesis author doesn’t aim to be definite, it is used in particular for the known malware leaving the unknown samples out of the reach of the project and for a further development when sandboxing or more advanced techniques, capable of unknown malware detection, will be used, guaranteeing a lower amount of false negatives and a better performance overall.

5. Conclusions and future work

With the rapid distribution of Android devices equipped with a lot of features the number of mobile malware is increasing. The thesis author has analyzed how much is Android targeted by malicious authors and provided an overview of the existing countermeasures together with their limitations.

Taking into account the limited computation power and energy source of smarthones Ludroid, a new security mechanism was presented. This system allows users to analyze Android files prior of downloading them; all the user has to do is to paste the link of the application he/she wants to test and after the malware analysis is finished the user will receive security recommendations or alerts about installing the desired file. This feature distinguishes this project from other cloud based detection techniques. And most importantly, this complex is a flexible and scalable system that might be integrated directly into unoffical markets and third-party distributions. While selecting the system which will be integrated into Ludroid, a public API became the main criterion. Since this work is a non-commercial project (with the exception of renting a server), it was decided to use only non-commercial cloud-based detection solutions for demonstration purposes. Due to the fact that almost all available online scanners use mechanisms similar to each other, it was decided to discard alternative scanners and to confine to VirusTotal, currently the most comprehensive and publicly available.

After a new malware is launched, it takes some time for the anti-virus engines to detect this new threat; only after one of the systems detected it, the file will be flagged as malicious by the system which integrates it, VirusTotal in our case. Therefore, with simultaneous verification by a large number of anti-virus scanners, the probability of finding malicious code increases but to the time it's detected it might already cause a lot of harm.

Because of the above mentioned limitations the thesis author decided that in the future, in order to enhance the tool performance novel detection techniques, such as custom sandboxing or static analysis tools will be integrated into Ludroid, based on their efficiency, availability and technological advancement.

References

[1] M. Gargenta, Learning android, O’Reilly Media, Inc., 2011, p. 2.

[2] Developers, "What is android," Android Studio, 2011. [Online]. Available: https://developer.android.com/guide/components/fundamentals.html. [Accessed 15 November 2016].

[3] Developers, "Application Manifest," Android Studio, 2011. [Online]. Available: http://developer.android.com/guide/topics/manifest/manifest-intro.html. [Accessed 19 November 2016].

[4] Developers, "About Android," Android Studio, 2011. [Online]. Available: https://developer.android.com/about/android.html. [Accessed 26 November 2016].

[5] Statista Inc., "Google Play: number of available apps 2009-2017," [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. [Accessed 13 April 2017].

[6] Canalys, "Over 1 billion Android-based smart phones to ship in 2017," 4 June 2013. [Online]. Available: https://www.canalys.com/static/press_release/2013/canalys-press-release-040613-shipments-android-based-smart-phones-exceed-1-billion-2017.pdf. [Accessed October 2016].

[7] Cheetah Mobile Security Lab, "Android App Stores Become Significant Sources for Malware," Cheetah Mobile, 20 January 2016. [Online]. Available: http://www.cmcm.com/blog/en/security/2016-01-20/925.html. [Accessed 4 February 2017].

[8] K. Dunham, S. Hartman, . M. Quintans and J. A. Morales, Android Malware and Analysis, CRC Press, 2014, p. 36.

[9] J. Six, Application Security for the Android Platform: Processes, Permissions, and Other Safeguards, O'Reilly Media, Inc, 2011, pp. 25-26.

[10] M. Chandramohan and B. K. T. Hee, "Detection of mobile malware in the wild," *Computer 45,* no. 9, pp. 65-71.

[11] R. Raveendranath, . V. Rajamani, A. Babu and . S. Datta, "Android malware attacks and countermeasures: Current and future directions," in *Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on*, 2014.

[12] B. Baskaran and A. Ralescu, A Study of Android Malware Detection Techniques and Machine Learning, 2016, p. 16.

[13] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck and J. Hoffmann, "Mobile-sandbox: having a deeper look into android applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, March.

[14] Y. Feng, S. Anand, I. Dillig and A. Aiken, "Apposcopy: Semantics-based detection of android malware through static analysis.," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 576-587. ACM, November 2014.

[15] A. Felt, E. Ha, S. Egelman, A. Haney, E. Chin and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the eighth symposium on usable privacy and security*, p.3. ACM, July 2012.

[16] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck and Siemens, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *NDSS*, February 2014.

[17] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck and J. Hoffmann, "Mobile-sandbox: having a deeper look into android applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 1808-1815. ACM, March 2013.

[18] A. Desnos, "Androguard-reverse engineering, malware and goodware analysis of android applications... and more (ninja!) Googld Project Hosting," 2012.

[19] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. Chun, L. Cox, J. Jung, P. McDaniel and A. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *ACM Transactions on Computer Systems (TOCS)*, January 2014.

[20] I. Burguera, U. Zurutuza and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 15-26. ACM, October 2011.

[21] T. Bläsing, L. Batyuk, A. Schmidt, S. Camtepe and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Malicious and unwanted software (MALWARE), 2010 5th international conference on*, pp. 55-62. IEEE , 2010.

[22] Y. Zhou, Z. Wang, W. Zhou and X. Jiang, "Hey, you, get off of my market: detecting malicious apps in official and alternative android markets," *NDSS*, vol. 25, no. 4, pp. 50-52, February 2012.

[23] L. Weichselbaum, M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. van der Veen and C. Platzer, "Andrubis: Android malware under the magnifying glass," in *Vienna University of Technology, Tech. Rep. TR-ISECLAB-0414-001*, 2014.

[24] W. Enck, M. Ongtang and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 235-245. ACM, 2009.

[25] A. Felt, E. Chin, S. Hanna, D. Song and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 627-638. ACM, October 2011.

[26] M. Grace, Y. Zhou, Q. Zhang, S. Zou and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 281-294. ACM, June 2012.

[27] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. Bringas and G. Álvarez, "Puma: Permission usage to detect malware in android," in *International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pp. 289-298. Springer Berlin Heidelberg, 2013.

[28] M. N. P. Pravin, "VetDroid: Analysis Using Permission for Vetting Undesirable," *International Journal of Innovative and Emerging*, vol. 2, no. 3, p. 6, 2015.

[29] J. Kim, Y. Yoon, K. Yi, J. Shin and S. Center, "SCANDAL: Static Analyzer for Detecting Privacy Leaks in Android Applications," MoST, May 2012. [Online]. Available: http://www.mostconf.org/2012/papers/26.pdf. [Accessed January 2016].

[30] W. Zhou, Y. Zhou, X. Jiang and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pp. 317-326. ACM, February 2012.

[31] Y. Aafer, W. Du and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *International Conference on Security and Privacy in*

Communication Systems, pp. 86-103. Springer International Publishing, September 2013.

[32] E. Chin, A. Felt, K. Greenwood and W. D., "Analyzing inter-application communication in Android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services,* pp. 239-252. ACM, June 2011.

[33] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *Acm Sigplan Notices,* vol. 49, no. 6, pp. 259-269, June 2014.

[34] A. Martín, A. Calleja, H. Menéndez, J. Tapiador and D. Camacho, "ADROIT: Android malware detection using meta-information," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on,* pp. 1-8. IEEE, December 2016.

[35] A. Gorla, I. Tavecchia, F. Gross and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering,* pp. 1025-1035. ACM, May 2014.

[36] P. Faruki, V. Laxmi, A. Bharmal, M. Gaur and V. Ganmoor, "AndroSimilar: Robust signature for detecting variants of Android malware," *Journal of Information Security and Applications,* vol. 22, pp. 66-80, June 2015.

[37] M. Zheng, M. Sun and J. Lui, "Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pp. 163-171. IEEE, July 2013.

[38] M. Zhao, T. Zhang, F. Ge and Z. Yuan, "RobotDroid: A Lightweight Malware Detection Framework On Smartphones," *JNW,* vol. 7, no. 4, pp. 715-722, April 2012.

[39] S. Arshad, M. Shah, A. Khan and M. Ahmed, "Android malware detection & protection: a survey," *Int. J. Adv. Comput. Sci. Appl,* vol. 7, no. 2, pp. 463-475, February 2016.

[40] L. Xie, X. Zhang, J. Seifert and S. Zhu, "pBMDS: a behavior-based malware detection system for cellphone devices," in *Proceedings of the third ACM conference on Wireless network security*, pp. 37-48. ACM, March 2010.

[41] G. Dini, F. Martinelli, A. Saracino and D. Sgandurra, "MADAM: a multi-level anomaly detector for android malware," in *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pp. 240-253. Springer Berlin Heidelberg, October 2012.

[42] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer and Y. Weiss, "“Andromaly”: a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems,* vol. 38, no. 1, pp. 161-190, February 2012.

[43] M. Dimjaševic, S. Atzeni, I. Ugrina and Z. Rakamaric, "Android malware detection based on system calls," University of Utah, Tech. Rep, May 2015.

[44] A. Desnos and P. Lantz, "An android application sandbox for dynamic analysis," Electrical and Information Technology, Lund university, Lund, Sweden, 2011.

[45] L. Yan and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," in *USENIX security symposium*, pp. 569-584, August 2012.

[46] K. Tam, S. Khan, A. Fattori and L. Cavallaro, "CopperDroid: Automatic Reconstruction of Android Malware Behaviors," in *NDSS,* February 2015.

[47] P. Faruki, S. Bhandari, V. Laxmi, M. Gaur and M. Conti, "DroidAnalyst: Synergic App framework for static and dynamic app analysis," in *Recent Advances in Computational Intelligence in Defense and Security,* pp. 519-552, 2016.

[48] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis and S. Ioannidis, "Rage against the virtual machine: hindering dynamic analysis of android malware," in *Proceedings of the Seventh European Workshop on System Security*, (p. 5). ACM., April 2014.

[49] V. Babu Rajesh, P. Reddy, P. Himanshu and M. Patil, "ANDROINSPECTOR: ASystem FOR COMPREHENSIVE ANALYSIS OF ANDROID APPLICATIONS," *International Journal of Network Security & Its Applications (IJNSA),* vol. 7, no. 5, p. 21, September 2015.

[50] S. Bhandari, R. Gupta, V. Laxmi, M. Gaur, A. Zemmari and M. Anikeev, "DRACO: DRoid analyst combo an android malware analysis framework," in *Proceedings of the 8th International Conference on Security of Information and Networks*, pp. 283-289. ACM, September 2015.

[51] M. Sun, X. Li, J. Lui, R. Ma and Z. Liang, "Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android," *IEEE Transactions on Information Forensics and Security,* vol. 12, no. 5, pp. 1103-1112, 2017.

[52] M. Lindorfer, M. Neugschwandtner and C. Platzer, "Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis," *Computer Software and*

Applications Conference (COMPSAC), 2015 IEEE 39th Annual, vol. 2, no. IEEE, pp. 422-433, July 2015.

[53] Gartner, "Global Sales of Smartphones Grew 4.3 Percent Year on Year," 19 August 2016. [Online]. Available: http://www.gartner.com/newsroom/id/3415117. [Accessed 2 March 2017].

[54] L. D., "Scanning for Malware in Android Applications," OPSWAT, 14 April 2014. [Online]. Available: https://www.opswat.com/blog/scanning-malware-android-applications. [Accessed 13 February 2017].

[55] R. A., "PC-Grade Malware Going Mobile," Security Intelligence, 25 June 2014. [Online]. Available: https://securityintelligence.com/pc-grade-malware-going-mobile/. [Accessed March 2017].

[56] T. Vidas and N. Christin, "Sweetening android lemon markets: measuring and combating malware in application marketplaces," in *Proceedings of the third ACM conference on Data and application security and privacy*, pp. 197-208. ACM, 2013.

[57] S. Rastogi, K. Bhushan and B. Gupta, "Measuring Android App Repackaging Prevalence based on the Permissions of App," *Procedia Technology,* vol. 24, pp. 1436-1444, 2016.

[58] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 95-109. IEEE, 2012.

[59] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. Bringas and G. Álvarez Marañón, "MAMA: manifest analysis for malware detection in android," *Cybernetics and Systems,* vol. 44, no. 6-7, pp. 469-488, 2013.

[60] Google Security Services for Android, "Android Security 2015 Year in Review," April 2016. [Online]. Available: https://source.android.com/security/reports/Google_Android_Security_2015_Report_Final.pdf. [Accessed March 2017].

[61] R. Ramachandran, T. Oh and W. Stackpole, "Android anti-virus analysis," in *Annual symposium on information assurance & secure knowledge management*, pp.35-40, 2012.

[62] Statista Inc., "Share of people who used anti-virus software on mobile phones in Denmark in 2016," [Online]. Available: https://www.statista.com/statistics/649682/usage-of-mobile-anti-virus-software-in-denmark/. [Accessed 14 March 2017].

[63] . H. Lockheimer, "Android and Security," VP of Engineering, Android , 2 February 2012. [Online]. Available: http://googlemobile.blogspot.com.ee/2012/02/android-and-security.html. [Accessed April 2017].

[64] Webroot, "Cloud-Based Mobile Device Security," 2013. [Online]. Available: https://www.webroot.com/shared/pdf/WebrootMobileSecurity.pdf. [Accessed April 2017].

[65] Check Point Research Team, "More Than 1 Million Google Accounts Breached by Gooligan," 30 November 2016. [Online]. Available: http://blog.checkpoint.com/2016/11/30/1-million-google-accounts-breached-gooligan/. [Accessed April 2017].

[66] L. Pinto, "Aptoide Numbers Reached New Heights in 2016!," Aptoide, 29 December 2016. [Online]. Available: http://blog.aptoide.com/aptoide-by-numbers/. [Accessed 13 April 2017].

[67] A. Kapratwar, "Static and Dynamic Analysis for Android Malware," 2016. [Online]. Available: http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1488&context=etd_projects. [Accessed January 2017].

[68] T. D., "Yet another static code analyzer for malicious Android applications," March 2013. [Online]. Available: https://github.com/maaaaz/androwarn. [Accessed January 2017].

[69] koodous.com, April 2017 . [Online]. Available: https://koodous.com/apks/b68afb459d1d01d252409522706aed1d1a23e62782f5582c019cb1006321bf0e/analysis. [Accessed April 2017].

[70] Payload Security, "VxStream Sandbox: Understand and Fight Unknown Threats," [Online]. Available: https://www.payload-security.com/download/VxStream%20Sandbox%20Datasheet.pdf. [Accessed 2017].

Appendix 1 – Interview with Aptoide security personnel

“Our security system scans every uploaded application using multiple anti-virus systems. Since all of the anti-virus systems we use are integrated into our security system, we do not have much interest in using the exact same tools to externally analyze a sample of our applications.

They would not detect anything, as any detection in our security system (with some exceptions, more on that in the next paragraph) immediately removes all instances of the application from Aptoide.

Some anti-virus detections are automatically ignored by our system, such as:

- Detections that have been confirmed to be false positives at all times;

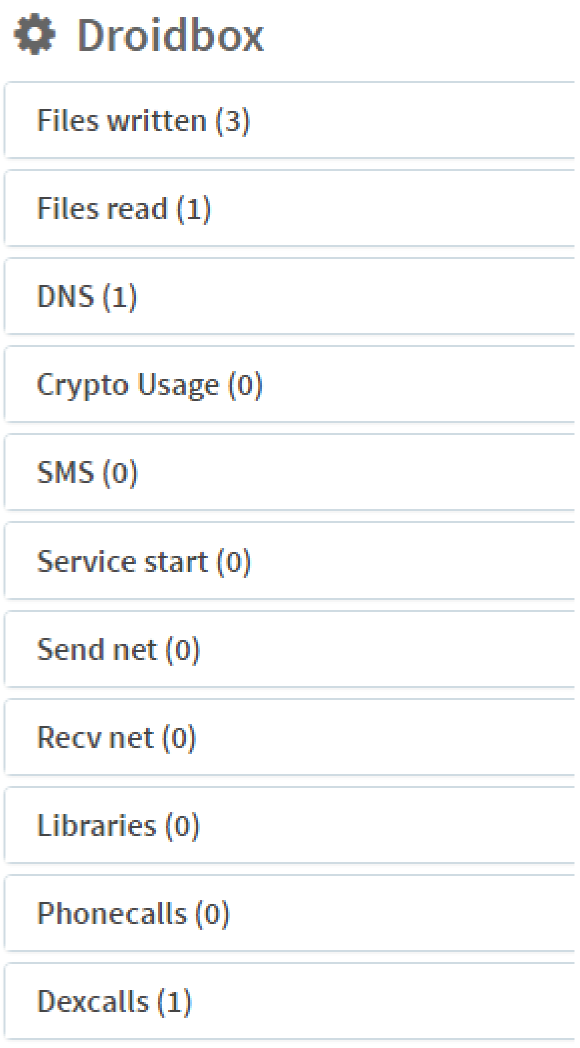
- Potentially unwanted adware that is not extremely invasive (such as Airpush and Startapp). However, in such cases, the application immediately gets a Warning or Adware badge due to such detections;

- Detections for selected applications, such as Lucky Patcher and King Root. Most applications that root your device will be detected by a number of anti-virus systems, even though they are completely safe for the end-user. For instance, for King Root: https://virustotal.com/en/file/ddfc0a7bb5146524b2c2a65e5051b263fe415528b8bdacaf620750e2e11d02ba/analysis/

While we're not exactly sure of what Cheetah Mobile considers as malware or not, as this differs for every anti-virus system, we believe that most, if not all, of their 20 detections are instances of the above cases, which are not very frequent in Aptoide.

To answer your question, yes, 20 occurrences of "malware" out of 37098 applications, accounting for 0.05% of the sample, seems like an accurate value. We do not have any official data about this available, for the reasons explained above.”

Appendix 2 – Example of report generated by Droidbox



Droidbox

Files written (3)

Files read (1)

DNS (1)

Crypto Usage (0)

SMS (0)

Service start (0)

Send net (0)

Recv net (0)

Libraries (0)

Phonecalls (0)

Dexcalls (1)

Appendix 3 – Report response from VirusTotal

```
{
    'response_code': 1,
    'verbose_msg': 'Scan finished, scan information embedded in this object',
    'resource': '99017f6eebbac24f351415dd410d522d',
    'scan_id':          '52d3df0ed60c46f336c131bf2ca454f73bafdc4b04dfa2aea80746f5ba9e6d1c-
1273894724',
    'md5': '99017f6eebbac24f351415dd410d522d',
    'sha1': '4d1740485713a2ab3a4f5822a01f645fe8387f92',
    'sha256': '52d3df0ed60c46f336c131bf2ca454f73bafdc4b04dfa2aea80746f5ba9e6d1c',
    'scan_date': '2010-05-15 03:38:44',
    'positives': 40,
    'total': 40,
    'scans': {
      'nProtect': {'detected': true, 'version': '2010-05-14.01', 'result': 'Trojan.Generic.3611249',
'update': '20100514'},
      'CAT-QuickHeal': {'detected': true, 'version': '10.00', 'result': 'Trojan.VB.acgy', 'update':
'20100514'},
      'McAfee': {'detected': true, 'version': '5.400.0.1158', 'result': 'Generic.dx!rkx', 'update':
'20100515'},
      'TheHacker': {'detected': true, 'version': '6.5.2.0.280', 'result': 'Trojan/VB.gen', 'update':
'20100514'},
      .
      .
      .
      'VirusBuster': {'detected': true, 'version': '5.0.27.0', 'result': 'Trojan.VB.JFDE', 'update':
'20100514'},
      'NOD32': {'detected': true, 'version': '5115', 'result': 'a variant of Win32/Qhost.NTY',
'update': '20100514'},
      'F-Prot': {'detected': false, 'version': '4.5.1.85', 'result': null, 'update': '20100514'},
      'Symantec': {'detected': true, 'version': '20101.1.0.89', 'result': 'Trojan.KillAV', 'update':
'20100515'},
      'Norman': {'detected': true, 'version': '6.04.12', 'result': 'W32/Smalltroj.YFHZ', 'update':
'20100514'},
      'TrendMicro-HouseCall':    {'detected':    true,    'version':    '9.120.0.1004',    'result':
'TROJ_VB.JVJ', 'update': '20100515'},
      'Avast': {'detected': true, 'version': '4.8.1351.0', 'result': 'Win32:Malware-gen', 'update':
'20100514'},
      'eSafe': {'detected': true, 'version': '7.0.17.0', 'result': 'Win32.TRVB.Acgy', 'update':
'20100513'}
    },
    'permalink':
'https://www.virustotal.com/file/52d3df0ed60c46f336c131bf2ca454f73bafdc4b04dfa2aea80746f5ba9e6d1c/analysis/1273894724/'
   }
```

Appendix 4 – Report response from Jotti

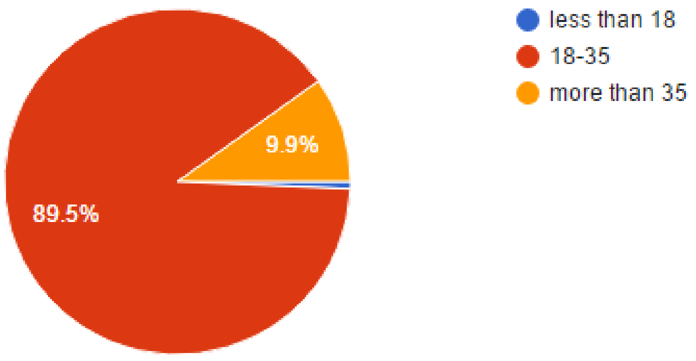
```
{
  'file':{
    'hashes':{
      'sha256':'4ed7b65890b51b0879866fd7a257536061214906e5d778add519437c6f3a9cd7',
      'sha1':'a25368862386892bb7c50c8d88a63d123d17f150',

'sha512':'59e6b2257b874e3e409c70edf2c613ff0a3e108725fb8af5bca499010ed8041566a12204c
b0cdbc2d1232d461d52cbcd80c75b159ebaa17442cfab85f88c1957',
      'md5':'8b484a016745def60f292a127d1ea22b'
    },
    'type':'Zip archive',
    'name':'fb.apk',
    'firstSeen':'2017-04-11 20:09:30+02:00',
    'size':1524590
  },
  'scanJob':{
    'startedOn':'2017-04-11 20:38:16+02:00',
    'scannersDetected':0,
    'finishedOn':'2017-04-11 20:38:48+02:00',
    'webUrl':'https://virusscan.jotti.org/filescanjob/ab26sixjsq',
    'scannerResults':[
      {
        'scannerId':'adaware',
        'scannerLogoUrl':'https://virusscan.jotti.org/img/logo/filescanner/adaware-logo.png',
        'malwareName':'',
        'finished':True,
        'signatureFileDate':'2017-04-11',
        'finishedWithoutResultReason':None,
        'scannerName':'Lavasoft Ad-Aware'
}
```

Appendix 5 – Internet survey results

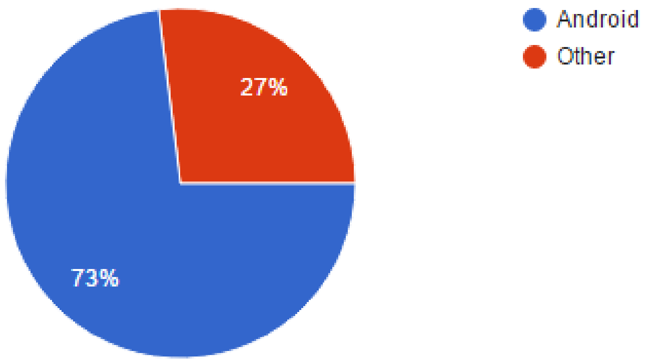
Behavior of mobile applications installation

152 responses

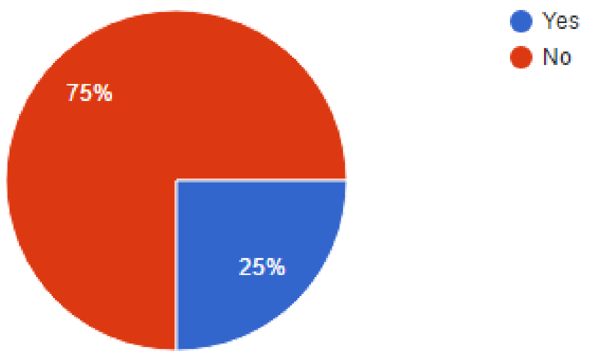
What's your age? (152 responses)



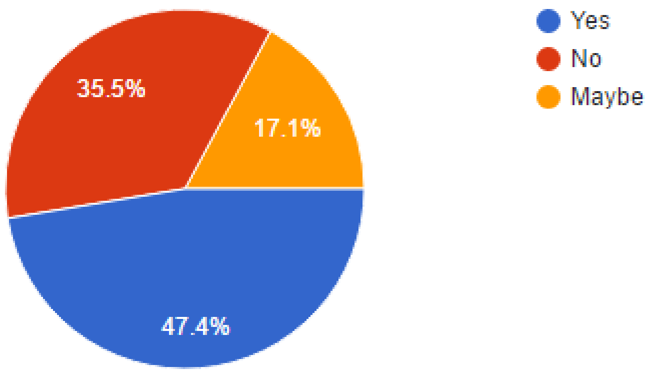
Which mobile platform do you use? (152 responses)



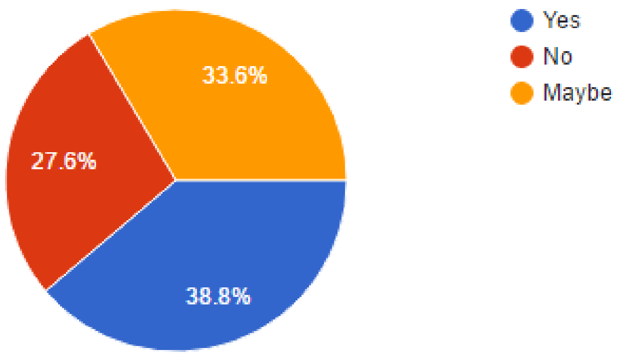
Do you use any antivirus system on your mobile device?
(152 responses)



Have you ever installed applications from third-party repositories?
(152 responses)



Would you use an application that would let you download apps from unofficial markets only after having performed security checks?
(152 responses)


● Yes
● No
● Maybe