NTNU – Trondheim
Norwegian University of
Science and Technology

Android Apps and Permissions: Security and Privacy Risks

Trond Boksasp
Eivind Utnes

Master of Telematics - Communication Networks and Networked
Submission date: June 2012
Supervisor: Svein Johan Knapskog, ITEM
Co-supervisor: Pern Hui Chia, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Third-party applications drive the attractiveness of web and mobile application platforms. Many platforms (incl. Android, HTML5 web apps, Facebook) rely on granular permissions to avoid granting full privileges to third-party applications. The case of Android OS is particularly interesting. However, the permission system on Android is complex. There are more than 135 official permissions, and it has been a challenge in communicating the actual scope of each permission to both the developers and users. This creates rooms for exploitations; malicious applications (or grayware) disguise themselves amongst the hundreds of thousands of normal ones.

This project will focus on a large scale data collection and analysis to measure and characterise the behaviour of bad applications. The basic ideas (adaptable to student's interests) would be as follows:

1. Build an automated and long term data collection process (e.g., using Bash/Python)
2. Parse and organise the information obtained into structured database (e.g., using MySQL)
3. Analyse the data and visualize interesting patterns (e.g., using R)
4. Characterise the behaviour of bad apps (e.g., detecting anomalous permission requests)

Assignment given: 23.01.2012
Supervisor: Pern Hui Chia, Q2S Professor: Svein Knapskog, Q2S

Preface

This master's thesis completes our 2 year master's program in Telematics at the Norwegian University of Science and Technology.

We would like to thank our supervisor Pern Hui Chia from Q2S at NTNU for all the valuable guidance and help during the course of this project. This project could not have been accomplished without you. Thanks also to Professor Svein Johan Knapskog from Q2S for getting this project up and running, and for guiding us through the finishing stages.

We greatly appreciate the students at Futurum for keeping our spirits up, and the students at Victoria for keeping us sane.

Lastly, we would like to thank our families for believing in us even when we didn't. Your continuous support over the years have been important.

Abstract

This thesis investigates the permissions requested by Android applications, and the possibility of identifying suspicious applications based only on information presented to the user before an application is downloaded. During the course of this project, a large data set consisting of applications published on Google Play and three different third-party Android application markets was collected over a two-month period. These applications are analysed using manual pattern recognition and k-means clustering, focusing on the permissions they request. The pattern analysis is based on a smaller data set consisting of confirmed malicious applications. The method is evaluated based on its ability to recognise malicious potential in the analysed applications. The k-means clustering analysis takes the whole data set into consideration, in the attempt of uncovering suspicious patterns. This method is evaluated based on its ability to uncover distinct suspicious permission patterns and the findings acquired after further analysis of the clustering results.

Sammendrag

Denne masteroppgaven undersøker tillatelsene etterspurt av Android applikasjoner og mulighetene for å identifisere mistenkelige programmer basert kun på informasjon presentert til brukeren før applikasjonen blir lastet ned. Under gjennomføringen av dette prosjektet har vi laget ett datasett bestående av applikasjoner fra Google Play og tre tredjeparts applikasjons-markeder, samlet over en tomåneders periode. Applikasjonene er analysert med manuell mønstergjenkjenning og k-means gruppering med fokus på tillatelsene de ber om. Mønstergjenkjenningen er basert på et mindre datasett bestående av bekreftede ondsinnede applikasjoner, og metoden er evaluert etter dens evne til å gjenkjenne ondsinnet potensiale i de analyserte applikasjonene. Grupperingsanalysen tar hele datasettet i betraktning for å finne mistenkelige mønstre. Denne metoden er evaluert etter dens evne til å avdekke mistenkelige mønstre og funnene ervervet etter nærmere analyse av resultatene fra grupperingen.

Contents

List of Figures

List of Tables

Abbreviations

Amazon EC2	Amazon Elastic Compute Cloud
APK	Application Package
C&C	Command and Control Server
DOS	Disk Operating System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
MD5	Message Digest 5
OEM	Original Equipment Manufacturer
SIM	Subscriber Identity Module
SMS	Short Message Service
SQL	Structured Query Language
SSE	Sum of Squared Error
UID	User Identifier
XML	Extensible Markup Language

Chapter 1

Introduction

With the advent of smartphones, users are, knowingly or not, carrying more and more private information around with them on their phones. This information range from the location of the device to the reading habits of the user and even his or her bank details. While attacks on mobile devices have largely focused on earning the attacker quick cash by sending text messages to or calling premium numbers, the focus has shifted towards stealing the private data contained on the devices [82].

As the Android platform has grown to take one of the largest shares of the smartphone market, the platform has become the prime target for criminals seeking the private data the users are carrying around with them. At the same time, the security of the platform has come under scrutiny from security professionals.

This project will focus on identifying suspicious applications, both from the official Google Play market and third-party markets.

1.1 Motivation

Malicious software is a common problem for every software platform, and the Android platform is no exception. Since the first malicious Android application was discovered in 2010, the number of malicious applications has been consistently rising. Looking at the details of the various Android malware applications

recognised by Symantec [82], there is an apparent trend towards information stealing. We see that a large subsection of the malicious applications is dedicated to stealing private information from the users.

While the Google team is quick to remove malicious applications from Google Play when they are made aware of them, this can be after a considerable number of users have already downloaded the applications. The same applies to other third-party markets, and as such a method of identifying malicious applications before they are installed is required. Anti-virus applications and the recently unveiled Google Bouncer do partially fill this gap, but both the Bouncer and the anti-virus applications require that the malicious code has been analysed beforehand. A method of identifying new malicious applications before they are accepted to the markets is required.

1.2 Objective

The objective of this project is to gather and analyse a large data set consisting of publicly available application information from various Android markets. Additionally, a sample of malicious applications will be acquired for comparison.

These data sets will be used to examine the permissions requested by applications, through machine learning and permission request analysis. This examination will focus on the possibility of distinguishing whether or not an application is "suspicious" based on information available before the application is downloaded to the device, with particular focus on the requested permissions, either one permission alone or in combination with other permissions. In a real-world scenario, an application deemed suspicious by this method would be flagged for review.

1.3 Limitations

Because this project only examines the data available to the user before an application is downloaded, the source code of the applications will not be taken into consideration. As such, there are limitations to what can be analysed.

First, this means that implicit vulnerabilities, like permission sharing and capability leaks, as presented in Chapter 3, will not be detected. Secondly, because the markets used are not consistently listing third-party permissions among the regular Android permissions, third-party permissions will not be studied beyond

a cursory examination where appropriate during the analysis of the malicious applications in Chapter 5.

1.4 Thesis Structure

Chapter 2 presents the methodology used in this project.

Chapter 3 explains the background for this project. It will present general information on malicious applications, threats and the security model used on the Android platform. Additionally, it will cover work that relates to this project.

Chapter 4 relates to the data collection done for this project. It will explain how the third-party markets were chosen, how the data was collected and the challenges faced in the data collection part of the project.

Chapter 5 presents the analysis of the gathered data. This chapter will include both pattern analysis and analysis by k-means clustering.

Chapter 6 presents a discussion of the results from Chapter 5.

Chapter 7 present our ideas for future work.

Chapter 8 presents the conclusion of this thesis.

Chapter 2

Methodology

The progress of this project was spaced out in four phases.

2.1 Phase 1 - Surveying Android Markets & Android Security

First, decisions were made as to what information would be required and which markets were suitable. From markets dedicated to open source (F-Droid), markets aimed at handsets sold by particular vendors (LG, Samsung) to markets publishing adult content (Mikandi), the list of Android dedicated markets is long and growing. See Appendix A for a list of all the surveyed markets.

Looking at the list of markets, we were originally afraid that we would have to discard perfectly valid markets due to time constraints, but in the end only four markets fulfilled our criteria. The process of selecting suitable markets is further described in Section 4.1.1.

An application with a template of what data we should focus on was provided to us by our supervisor. This blueprint was based on his research [17] on the available data in the Google Play market, and the application was used as a starting point in the next phase of the project.

After looking at the information present in the selected Android markets, it was

decided that we would use the template with some modifications. Due to how some of the selected markets are constructed, the developer information (except the developer identifier) and the comments made about the application were ignored. An exception to this was made for the Google Play market, for which the aforementioned application developed by our supervisor was used.

2.2 Phase 2 - Developing Data Collection Programs and Infrastructure

Several applications were required to gather the required data, and decisions had to be made on which language(s) to use. This decision was based on scalability, platform independence and the existence of any libraries required. The application mentioned in the previous phase was used as the baseline for the data collection programs. One data collection program was required for each market, as detailed in Section 4.1.

During this phase, one member of the team wrote the data collection program while the other developed automation and warning scripts for the server. As explained in Section 4.1.1, the data collection programs were implemented on an Amazon Elastic Compute Cloud (Amazon EC2) instance, which meant that server stability was guaranteed by Amazon. This meant that we did not have to worry about the stability of the servers, but we would still need warnings if the applications ran into issues.

2.3 Phase 3 - Data Collection, Initial Analysis Planning

While the programs were collecting data, plans were laid on how to best analyse the collected data. It is important to note that this was done in this phase in order to base it on the actual data collected from the markets, as before the data collection started we were unable to verify how consistent the markets were in their data presentation.

After the planning was completed, one member of the team started studying the machine learning algorithms while the other started collecting malicious applications to use as a comparison for the applications retrieved from the markets.

During this phase, one of the data collection programs underwent a minor adjustment as the F-Droid market changed the layout and coding of the front page. The issue was quickly rectified, and this combined with the slow growth of the F-Droid market meant that there was not a significant loss of data.

The data collection phase was stopped after sixty-five days, after having run from the third of March to the seventh of May. This provided us with a total of 26,438 applications in total, both legitimate and malicious.

2.4 Phase 4 - Analysis, Findings & Write-up

Before the data could be analysed the permission names needed to be standardised, and misspelled and undocumented permissions had to be identified. Analysis based on categories was largely cut due to time constraints, although briefly examined during the analysis of the clustering algorithm results.

The analysis was split up between the two members of the team, with one member analysing patterns in the requested permissions and the second analysing them using clustering algorithms. The results of these analysis methods were then compared to each other, and conclusions on the validity of each method were reached.

Chapter 3

Background

3.1 Short History of Malware

Since the first recognised malicious application, the "Brain" virus [26] which first attacked the DOS platform back in 1986, viruses have evolved drastically, both in complexity and targeting. While the original Brain virus simply renamed the C: drive on the infected computers, more recent Trojans like "Zeus" [23] which attacks the Windows operating system, and "Flashback" [22] which attacks the OSX platform, attempt to steal personal and financial information.

This trend has been going on since Brain's inception, with malicious applications moving from proof-of-concept and bragging rights towards financially motivated attacks. The same trend is evident on the mobile platform, when seen as a whole. The first malware aimed at mobile devices, Cabir [70], infected devices running the Symbian operating system through their Bluetooth connection with the sole purpose of propagating itself. It did not appear to have any payload beyond what was necessary to continue spreading, and as such was more annoying than dangerous. By contrast, recent mobile malware attacking the Android platform, like DroidKungFu [24] and GinMaster [25], attempt to steal private information much like their desktop counterparts. Other malicious applications attempt to turn the devices into bots or simply incur costs on behalf of the malware developers.

3.2 Android Malware

Android malware appears to have moved beyond the proof-of-concept and destructive phase almost completely. The first malware recorded by Symantec, Ewalls [76], attempts to steal personal information from the device it is installed on, including the devices IMEI (International Mobile Equipment Identity) number and details from the SIM (Subscriber Identity Module) card including operator name and serial number.

We would argue that this is because the various mobile platforms should be seen as a single platform as far as malware motivation is concerned. More established mobile platforms, including Symbian and iOS, have already created a market for malicious applications which translates to the Android platform.

It is worth noting that despite this, the FakePlayer Trojan [78] is often considered the first malware aimed at Android [67]. This could be explained by the spread of the malware, as Ewalls appears to have managed to compromise a far smaller number of Android devices than Fakeplayer.

Perceived rise of Android malware There are many actors involved in the Android security scene, including Trend Micro, Symantec, Lookout and many more. Even as such, it is surprisingly hard to get reliable numbers about malicious applications in the wild, or even the growth of the malicious applications. Headlines like «Android Malware Surges Nearly Five-Fold Since July» [86] and «Android malware has jumped up 472% since July» [65] have been touted by many sites, but finding the original numbers have been surprisingly hard. The Divide by Zero blog managed to find the original source [72], and noted that the numbers were misrepresented.

Even industry giants like Trend Micro struggle with providing levelheaded data, as reading up on their Bouncer-like application (see Section 3.3.6) we are presented with a collection of impressive numbers. Quoting from the article [58]; «Trend Micro threat research experts identified more than 1,000 malicious Android applications in 2011» without stating whether or not this was unique infections or multiple samples of the same infections, and «that number growing at an astounding rate of 60% month over month», culminating in 120,000 malicious applications during 2012. For comparison, there are currently 440,000 [7]

applications in the Google Play market.

Despite the sensationalist scaremongering, the consensus is that the number of malicious applications targeting the Android platform is increasing, and the malware is becoming more and more sophisticated. As such, even if the numbers are grossly overstated, identifying and neutralising malicious applications should be a top priority for both Google and other interested parties.

3.3 Android Security

The Android system uses several methods to secure the devices of the users. Below we will describe the security features that affect applications directly, which are the features that are relevant for malicious applications to attempt to defeat or circumvent.

3.3.1 Permissions

Android restricts the capabilities of applications installed on the device by explicitly requesting the user to allow the application to access various parts of the operating system or features of the device. In order for an application to be able to use one of these capabilities, it is required to have the related permission been granted by the user (see Appendix B) during installation, as demonstrated in Figure 3.1. The permission system [36] is comprehensive and provides a good framework for determining what resources an application will have access to once it has been installed on a device.

These permissions are stored in a file called *manifest.xml* found inside the APK (Application Package) file of the application, and cannot be changed after the application is installed. An exception from this rule is made when updating applications, but the user is still required to approve any new permissions, in a similar process as when first installing the application.

Permissions are all or nothing. This means that when a user is installing an application, the user must either grant the application all the permissions it requests or refrain from installing the application. This solution prevents developers from worrying about whether or not a refused permission will cause the application to crash or behave incorrectly when trying to access the denied feature. It also prevents users from denying a suspicious application certain permissions even if

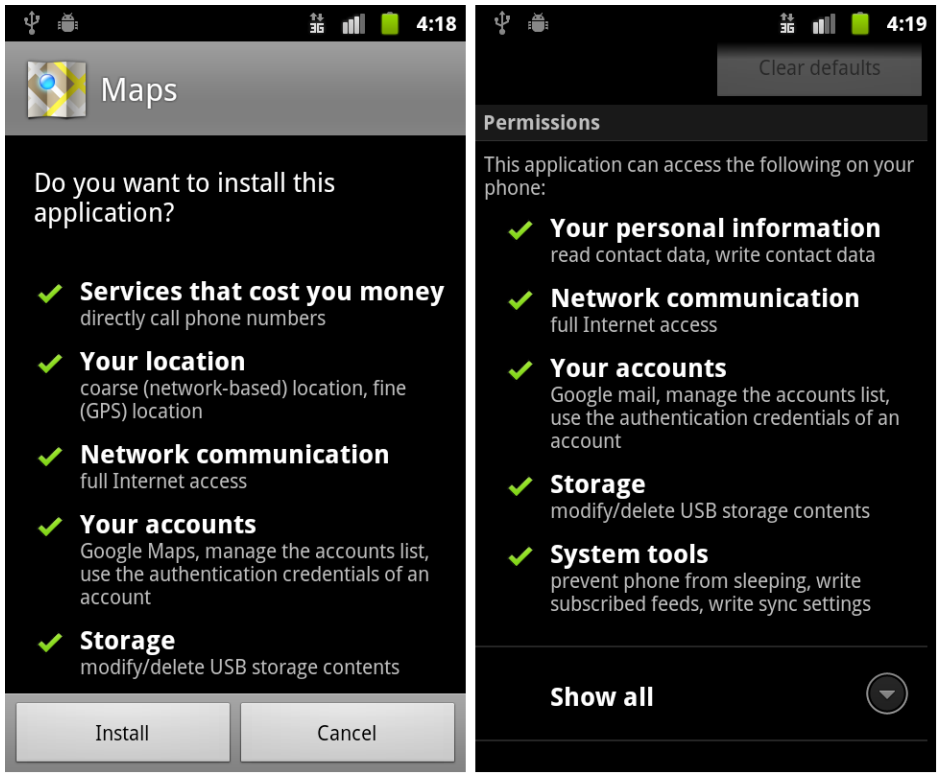

Figure 3.1: Android permission request (left) and the permissions of an installed application (right). Retrieved from [36]

the application itself provides an interesting service. While the general consensus is that if an application is suspicious, you should not install it, this also affects genuine applications which request a suspicious combination of permissions.

Developers can also create their own permissions [43], which can be used to give other applications access to features in the application. This does however counteract parts of the security imposed on the application by the sandbox feature covered in Section 3.3.2. These permissions are not explicitly stated to the users during installation, but can be determined by examining the Android manifest file.

Permissions are divided into four protection levels named, in order of protection; *normal*, *dangerous*, *signature* and *signatureOrSystem*. These levels are based on their capacity for damage or potential cost to the user, with most of the permissions being labeled as *dangerous* . These protection levels can also be given to third-party permissions.

Permissions with the *normal* (also called safe) protection level, like VIBRATE and SET_WALLPAPER, are permissions that are not considered to have any danger associated with them. The Android package installer will not ask the user for approval for these permissions [41]. The *dangerous* protection level however, will cause warnings to be displayed to the user before installation, and requires the user's approval to be granted.

The *signature* and *signatureOrSystem* protection levels protect the most dangerous permissions. *Signature* permissions are only available to an application that is signed with the same certificate, see Section 3.3.3, as the certificate that was used to sign the application declaring the permission [41]. Similarly, the *signatureOrSystem* level requires the application to either be a system application, i.e. a part of the system image, or that the application is signed by the same certificate as the one used to sign the version of the Android system installed on the device.

There are however some issues with the permission system, but these are problems with implementation of the permissions rather than the permissions themselves. Effectively, the permission system can be circumvented as demonstrated by Linberry et al. in their Blackhat talk [51], where they revealed that the RECEIVE_BOOT_COMPLETED permission is not actually checked. This means

that any application could register to start when the phone is turned on, and the system would not actually verify whether or not the application had requested this permission. It is currently unknown if this affects any other permissions. Additionally, in some cases an unrelated permission can give access equivalent to that of another permission, as covered in Section 3.4.9.

3.3.2 Sandbox

In the Android system each application is run as a separate user and provided with its own UID to separate each application into its own *sandbox* [36], preventing direct communication between applications.

Each application is run in its own private environment and is unable to access other applications or be accessed by other applications directly. This is a time-tried security feature that is based on capabilities native to the UNIX environment that Android is built on.

As mentioned in Section 3.3.1, the developer can effectively open the gate to the sandbox by allowing other applications to access features of the application by declaring their own permissions. This makes it possible for other applications to interact with the application despite the sandbox.

Additionally, the developer of an application can ask for a shared UID. With a shared UID multiple applications signed by the same developer (see Section 3.3.3) will share the same sandbox. Applications employing this method would have access to each others permissions, meaning that if one application requested INTERNET and another application requested SEND_SMS both applications would have access to INTERNET and SEND_SMS. They would also have access to each others files and information, and effectively be a single application as far as the operating system is concerned.

3.3.3 Application signing

The Android platform requires developers to sign their applications [44] before they can be installed on a device, using a self-signed certificate. This certificate ensures that a malicious developer is unable to impersonate another developer. The certificate also provides a level of trust between the developer and the operating system, in that the signing process alerts the system if the application has

been modified after the developer signed it.

3.3.4 Remote kill switch

The Google Play application has the ability to remotely remove applications from users' handsets [13] when the application is violating the Developer Distribution Agreement [37] or the Developer Program Policies [35]. In most cases, applications that violate these agreements are malicious in one way or another, and this capability has been utilized to remove malicious applications on more than one occasion [13, 14] after the applications have been removed from the market itself.

The remote kill switch is however only useful against applications installed through the Google Play market. Applications installed through unofficial channels are not affected by this feature.

3.3.5 File system protection

Android protects the core system files of the operating system by storing them in a read-only partition of the hard drive. Additionally, the sandboxing feature mentioned above prevents the applications installed on the device from accessing each other's files unless the files are intentionally or unintentionally exposed by the developer, as explained in Section 3.4.9.

3.3.6 Google Bouncer

Google have responded to criticism about Google Play with introducing a new layer of security, named Bouncer [53]. Bouncer checks new applications when they are uploaded to the market to identify potentially malicious applications, even going as far as to simulate the application running on an Android device to catch any hidden behaviour.

This is however an automated process that uses the characteristics of known malware to analyse the applications, which means that novel malware will not be detected by the bouncer. As an example, Trend Micro found several malicious applications in the Android market as recently as May 2012 [66].

3.3.7 Anti-virus applications

Anti-virus applications are applications created by third-party companies designed to prevent malicious applications from being installed on a device. There are many anti-virus applications on the market, including applications created by industry giants like F-Secure and Norton.

The effectiveness of these applications has been debated, but it is apparent that as with their desktop counterparts, they are becoming necessary in today's application climate. They often come with secondary functionality, like remote wiping capabilities and the ability to locate the device in the case of loss or theft.

3.4 Android Threat Landscape

There are several threats facing the Android system, and the following sections will list and explain some of the more common threats. A single malicious application can represent more than one of these.

3.4.1 Trojans

Generally speaking, all Android malware are Trojans. Because of the sandbox, as covered in section 3.3.2, the attack vectors used by viruses and worms are largely unavailable to the malware developers. Utilizing Trojans have thus become the norm.

As with its desktop counterparts, the malicious code is usually included as part of an otherwise legitimate looking application or added on to legitimate applications which are then redistributed [18] as the original application. Applications misused for this purpose are often paid applications redistributed as free applications on third-party markets.

3.4.2 Spyware

One of the most common types of malicious applications for the Android platform, spyware, are designed to siphon off private information of one kind or another. Spyware comes in two flavours; commercial and malicious. Commercial spyware are applications installed on the user's handset manually by another person specifically to spy on the user, while malicious spyware operates in a similar

fashion as its desktop counterpart; covertly stealing data and transmitting it to a third party.

One of the more famous cases of commercial spyware was CarrierIQ [49], used extensively by various mobile device manufacturers and vendors. CarrierIQ had the capability to log everything that was done on a device, including web searches using the secure HTTPS protocol, and was allegedly used to increase customer satisfaction by logging dropped calls and similar information [15]. The problem was that the application had the capabilities for much more, and there was no way for the average user to get rid of it. Additionally, there was no way for the users to know what information the vendors deemed necessary to «increase the user experience».

3.4.3 Root exploit

Having root access to an Android device works the same way as on other Unix based platforms, and can be compared with having administrator rights on a Windows computer. By default, the user will usually not have access to this feature on an Android device, as it will be locked down by the vendor. This is done both to prevent the user from accessing parts of the operating system that can damage or even destroy the device, and to prevent the user from removing software placed on the device by the vendor.

Root exploits are in most cases created by legitimate members of the Android community in order to gain control of their own devices, but are considered a double-edged sword among the security community. While rooting can give the user control over a device, it also gives the same amount of control to any applications which gain access to the root rights. This means that root privileges given to a malicious application can completely compromise the device, as the application can theoretically remove the root privileges from the user.

Trojans misusing these root exploits are among the most dangerous malicious applications and can cause all kinds of havoc, completely out of sight from the user. Like most Trojans, the malicious application pretends to be normal until it is installed on the user's device. When installed, it attempts to use one or more root exploits to gain root access to the device.

An application with root access can replace, modify and install applications as

it wishes, and as an example, the DroidKungFu Trojan [24] installs a backdoor on the phone once it has gained root access. It then disguises this backdoor from the user both by using an innocent-looking name and hiding the application icon from the user. This backdoor can then be used to install other malicious applications on the device or simply stealing private information.

3.4.4 Botnet

A botnet is a network of compromised devices, usually computers, which an attacker can use for his own purposes; often to steal sensitive data or as part of a denial of service attack. The owners of the compromised devices might not even be aware of the infection beyond noticing that the device is operating slower than usual. The recent version of the DroidKungFu Trojan [18], mentioned earlier, was used to create a botnet consisting of compromised Android devices.

3.4.5 Premium SMS sender

Some malicious applications are rather straight-forward in their design, where they ask for permission to send SMS messages on install and use this capability to send SMS messages to premium rate numbers. The Rufraud Trojan [81] pretended to be free versions of popular applications, and once installed on the user's device it would send SMS messages to a premium rate number determined by the country the phone was located in.

3.4.6 Drive-by-download

Recently, the Android platform has also been targeted by a *drive-by-download* attack [55], where the user is presented with a download pretending to be a system update when visiting a compromised website. If the user installs this false security update, the device is infected with a Trojan.

3.4.7 Proof-of-concept

Proof of concept Trojans are usually the least dangerous, and do not usually lead to large outbreaks. These attacks usually have no payload beyond what they need to infect the devices, like the aforementioned Cabir attack, see Section 3.1. They are usually created for bragging rights or to demonstrate a vulnerability.

3.4.8 Destructive Trojans

Destructive Trojans aim to damage the infected devices, or data stored on a device, in some way or another. This can be via file corruption, phone wiping or similar attacks.

3.4.9 Other threats

In addition to malicious applications the Android platform is vulnerable to other attack vectors. Some of which will be detailed below.

Phishing. The Android platform is as vulnerable, if not even more, as it's desktop counterparts. As noted by Felt and Wagner [31] the small screen on mobile devices makes it in some cases harder than normal for a user to identify whether or not he is being spoofed. Additionally, there have been reports of fake applications pretending to be banking applications [85], which when used to access the bank would steal the users login information.

Capability leaking. The Woodpecker project [45] reports that applications are leaking access to privileged device features, providing other applications with access to features they should not have access to. This means that these applications are exposing restricted features through less restricted interfaces.

As an example, a flaw was discovered in the Power Control widget [16], which is standard on all stock Android devices. This flaw leaked access to interfaces on the widget, allowing applications that did not have access to these features to toggle features like the GPS on and off. While this does not sound like a major problem, this was an application that was present on all Android devices. The potential for abuse was therefore larger than if the vulnerability was found in a less distributed application.

Information leaking. Similarly to capability leaks above, information leaks expose sensitive data to other applications on the device. This can be due to storing sensitive data in unprotected areas, as demonstrated by Brodeur [10], or the application giving out the information to anyone who knows how to ask. An example of this was the logging tool HTC installed on their handsets [68]. This logging tool exposed large amounts of private data to anyone requesting it using a simple HTTP request, without any validation on whether or not they should

have access to the information.

Another source of information leaking is the READ_LOG permission [51]. This permission allows the application to access the system logs, which in some cases, depending on the applications running on the device, can provide the application with access to information equivalent of the GET_TASKS, DUMP and READ_HISTORY_BOOKMARKS. Additionally, third-party applications were seen writing information usually restricted to ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, READ_SMS and READ_CONTACTS to the system logs, providing equivalent access to these resources as well.

3.5 Machine Learning

The information on machine learning is based on the lectures given by Andrew Ng, Associate Professor at Stanford University, published on Coursera [64].

Machine learning is a way of training algorithms to increase our understanding of a certain set of data. More specifically, machine learning attempts to develop algorithms from evaluating a set of training examples. It can be used to predict the outcome of new data based on previously analysed data, or to find patterns of similarity in a data set. It can also be applied to tasks where computers are to learn a certain type of behaviour, e.g. maneuvering an autonomous helicopter, based on some empirical data obtained during a training phase. One of the strengths of machine learning lies in the ability to perform tasks without explicitly programming an algorithm. As Arthur Samuel put it [69]: *«Field of study that gives computers the ability to learn without being explicitly programmed.»* Samuel was able to develop a machine learning algorithm designed with the objective to learn how to play checkers. Even though Samuel himself was not a good checkers player, the machine learning procedure ended up beating Samuel in checkers in the end.

Tom M. Mitchell, in his book on machine learning, provided a widely quoted definition on machine learning [60]: *«A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.»*

Every type of machine learning algorithm have its own way of measuring its performance. This is called the algorithm's *cost function*. The function is defined

so that minimizing it will result in finding the best possible algorithm to apply to a specific problem.

Machine learning problems can be solved by applying one, or a combination, of several types of algorithms. Two of these approaches to machine learning are discussed in the subsequent sections.

3.5.1 Supervised learning

In the real world, people might improve their performance on a specific task by getting feedback on the work they are doing, be it negative or positive feedback.

An example of supervised learning might be a face recognition program. The training data will be pictures of faces, which are labeled as the *correct* or *expected* outcome. The learning algorithm will go through these pictures and learn what patterns to look for in order to characterize a face. The output of the learning algorithm will be a function which will take pictures as input, and give a response indicating if the picture is evaluated as a face or not.

3.5.2 Unsupervised learning

A training set might not always be labeled, i.e. the training set has no *right* or *wrong* samples. The objective of the machine learning task might not be to evaluate a potential solution, as supervised learning does. Instead, the objective might be to analyse the data in order to find patterns or structure. Unsupervised learning is applied to tasks where the objective is to find a structure in a given data set. It might not always be easy to identify patterns in large sets of data, and unsupervised learning helps to locate such *hidden* patterns.

An example of a task where unsupervised learning might apply is a market segmentation analysis. A clothing store desires a deeper understanding of their customers to better fit their needs. Unsupervised learning can be applied to create structure in their customer database based on the data they have on their customers.

A different example might be how to organize computing clusters. Say you have a large number of computing resources, and you want to find a way to better organize them so that they work more efficiently. A clustering algorithm can be applied to analyse the traffic flow between the computing clusters, and find out

which clusters are working together, placing them together in the same groups. These groups can form the basis of an adjustment in how the computing clusters are organized.

3.5.3 K-means

Several clustering algorithms exist, but the most widely used is k-means. It is an iterative algorithm, organising numerical data in k number of clusters. The numerical data, or training sets, are organised in vectors with a dimension equal to the number of features to be evaluated. K-means consists of two steps:

1. Calculating the distance from the training set vectors to each cluster centroid and assigning the training example to the closest cluster centroid.

2. Moving the cluster centroids to the mean of the respective cluster's *members*.

The steps are repeated until the algorithm converges. Convergence is achieved when the second step no longer assigns any vectors to new cluster centroids.

The distance between the n-dimensional vectors (x) and a given cluster centroid (μ)is given by

$$\| x - \mu \|^2 \tag{3.1}$$

where x is a training example, and μ is the cluster centroid. By convention, the squared distance between the cluster centroid and the training example is used. This adds to the weight of the distance, although the end result of the cost function calculation will point to the same solution for either case.

This distance calculation is done for all training examples (x). A training example gets assigned to the cluster with the shortest distance after calculating the distance to all cluster centroids. c denotes the index of the cluster closest to x. After all the training examples have been assigned to a cluster, the second step begins. This is where the cluster centroids are moved. Each cluster centroid (μ) takes up the value of the mean of all the vectors assigned to the cluster.

When the cluster centroids have been re-calculated, the process starts over. This time, the training sets starts off with a cluster assigned to it. When c does

not change its value after the distances have been calculated, the algorithm has converged.

Random initialisation of k-means. When the k-means algorithm runs through the first iteration, it needs to initialise the cluster centroids. This can be done several ways. One option is to choose random vectors in the $\mathbb{R}^n$ space. A second option is to randomly choose k vectors from the training set. This second approach is implemented as default in the statistical computing language R [32].

The random initialisation of the cluster centroids may lead to different results for every new run. The convention is to run k-means several times, and pick the run where the cost function is minimised to indicate the best fit.

K-means cost function. The cost function is defined as the sum of squares within all the clusters. Put in other words, the squared distance between each training example and its respective cluster is added up for every k to form the result of the cost function. The formal definition of k-means' cost function is shown in Equation 3.2. S_i refers to the k clusters formed by the training sets.

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x_j \in S_i} \parallel x_j - \mu_i \parallel^2 \tag{3.2}$$

3.6 Related Work

With the Android platform being as popular as it is, it has become the research target of different groups.

Is this App Safe? This is a paper by Chia et al. which investigated privacy risks associated with applications on the Android, Chrome and Facebook platforms [17]. The authors collected data on applications from each platform, both new and popular, and analysed the effectiveness of the permission systems of each of these platforms. They also analysed how some applications are attempting to trick users into granting them unsafe permissions.

Kirin. This project, by Enck et al., is a security service created to assess applications at install-time to determine whether or not an application is malicious

based on a predefined set of certification rules [19]. The permission rules used by Kirin are based on the potential for misuse, not whether or not the permission is generally used for this purpose. Compared to our method, Kirin assesses the application after it is downloaded, meaning that it can assess the APK file directly. It has as such more access to the inner workings of the application, in this case the Intents specified by the application.

Droidranger. Like our project, the Droidranger project by Zhou et al., examines the permissions requested by the applications on the Google Play and third-party markets [87]. The key differences between our project and the DroidRanger project is that while they have a larger data set than ours, their data set consists of free applications and include source code analysis of the applications. The third-party markets used in their project are largely Chinese, which we were unable to parse due to the language barrier.

The focus of the DroidRanger project is to identify malicious applications, while our project is identifying permissions which can indicate malicious behaviour. However, their permission-based filtering mechanism operates very similarly to our pattern analysis.

Stowaway. This project, by Felt et al., identifies hidden permissions in the Android platform in order to detect over-privileged applications [28]. These permissions have been a valuable source during analysis of our malicious data set, as it allowed us to verify the existence of undocumented permissions.

Mobile Malware Survey. This paper, by Felt et al., details their analysis of the incentive behind malicious applications and the effectiveness of measures taken to prevent infections and identifying malicious applications [30]. It explains the motivation behind several types of malicious behaviour, and the measures taken to prevent it. Unlike our project, it does not aim at identifying malicious behaviour or present new methods of identifying malicious applications.

Permission Visualization Using Self-Organizing Map. This paper, by Barrera et al., examines the permissions requested by Android applications by categories, and discusses ways in which the android permission model could be improved [9]. While our data set does include the categories of the application, our analysis did not take this into account due to time constraints, except in

special cases.

Chapter 4

Data Collection

As mentioned earlier in Section 1.2, achieving the goals of this project required two data sets of applications for analysis; one sample of real-world market data, from now on referred to as the market data set, and an infected data set for comparison. For the purpose of this thesis, an infected data set consists only of applications infected with malicious code.

4.1 Building the Data Set

We used different methods for retrieving the application samples from their respective websites, as well as retrieving information from the malicious applications.

4.1.1 Market data set

The gathering of the market data set was based on work done by Chia et.al. [17], using the data gathering application from their project as a starting point. Unlike their project however, this project required the collection of data from multiple markets, not only the official market.

We gathered a large data set from four different application markets, consisting of multiple features, ranging from developer identity to requested permissions. The markets were sampled every fifteen minutes during this period, with an update process being run every week (see Section 4.1.2). This sampling was done over a

period of two months, from the third of March until the seventh of May 2012.

The key features of this data set is that by sampling all the new applications added to each market, the data set represents the whole spectrum of applications and applications are included with no regard for popularity or usefulness. This means that we catch applications that act suspiciously, break rules or even applications that contain malware. These are applications that normally would not make it on to the "top applications" lists, and as such would not be caught if we only sampled the top applications.

Selecting suitable application markets. In order to obtain as many application samples as possible, four Android application markets were chosen; Google Play [39]/AppBrain [5], Amazon Appstore for Android [3], F-Droid [20] and SlideMe [52]. The third-party markets were chosen because they fulfilled the following requirements:

- English language
- Publishes a list of the permissions requested by the applications
- Keeps a list of the latest applications published on the market
- Hosts its own portfolio of applications, not just redirecting the user to download the application from Google Play

The language requirement was essential in order to keep the database as consistent as possible, while the permission requirement was essential in order for the data set to have any value for the analysis. As the samples should be of the new applications added to the market, the site needed to have a list displaying the latest applications added to the market, or another way to identify new applications quickly. Google Play does not fulfill this requirement, but this has been worked around as described in the next section. The final requirement was necessary in order to avoid duplicate data sets. If a market simply retrieved the applications from the Google Play market, the data set created from this market would be a near identical copy of the Google Play data set, depending on the method used to determine new applications.

Google Play/AppBrain. As mentioned earlier, in Chapter 3, the Google Play [39] market is the official marketplace for Android applications. However, as one

of the key features of our data set is that it contains applications published to the markets during our data gathering period, the source of the applications are the "newest applications" or a similar list on each market. Unfortunately, Google Play does not hold such a list, and as such the AppBrain [5] market has been chosen to represent Google Play. The AppBrain market deviates from our market rules in that it scrapes data from the Google Play market and redirects the users to Google Play during installation. However, AppBrain does have a "latest applications" list, which we have used to select which applications we should retrieve from Google Play.

Amazon Application Store. The Amazon Application Store [3] is the official application market of Amazon, with many applications aimed at their Kindle Fire devices. To download applications from the Amazon market, a separate market application is needed. This application is however not available to customers outside the United States.

SlideMe. SlideMe [52] is a third-party application store that wants to be the go-to place for applications based on location, payment methods and niche appeal. According to their own numbers the SlideMe market application has the second largest reach after Google Play, and are installed by default on the handsets of 120 different OEMs (Original Equipment Manufacturers).

F-Droid. The F-Droid [20] market is dedicated to free, open source applications. It provided us with a very small data set, as shown in Section 4.3. This meant that any results from this market would be negligible when compared to the three larger markets.

The small data set can be attributed to the submission rules for this market, as every application uploaded needs to have its source code attached. This prevents most commercial uses, and it can be assumed that only dedicated open source developers upload their applications to it.

Collected application information. We retrieved the following information from the markets, where available:

- The application name
- Version number

- Required Android version
- Which market the application was observed on
- The time when the application was observed on the market
- The category of the application
- The developer information; name, website and e-mail address
- The average rating and the number of votes
- How many times the application has been downloaded
- The application's package size
- The permissions requested by the application
- The price of the application, or lack thereof
- The time when the application was last updated

Not all of the collected information is used in the analysis, but it is kept in the database in case of future study. Of this information, the permissions were the most important information collected, as this was the most consistent between the various markets. While not completely identical, the permissions were internally consistent in the markets which meant that we could easily rewrite the permissions to one format. For the purposes of this project, the official format used in the Android permission manifest [40] was used.

Data collection programs and infrastructure. Each market required its own custom data collection program due to the difference in both style and quality of the HTML code. However, each program operates in the same fashion as the others except for the actual parsing of the HTML code of the market.

For each third-party market, we use the Wget tool to download the "Latest Applications" or equivalent HTML page to the server. This page was then parsed using the Jericho HTML Parser [47], and links to the information page for each application was generated from the data. For each of these links, the corresponding HTML page was similarly downloaded from the markets and parsed for the relevant application information. The application information retrieved from the markets was then stored in an SQL database, see Section 4.2.

As covered in Section 4.1.1, for applications on the Google Play market the application retrieved the "Latest Android applications" list from the AppBrain site, which was then parsed and links to the equivalent Google Play application pages were constructed.

As each application was added to the data set while it was still on the "Latest" list, we would only have the initial information of the application while it was still on that list. This would lead to several issues which could degrade the value of the data set. First, this would misrepresent the popularity of the application, as the download count and average rating of the application would be stuck at the initial value. Secondly, we would not know whether or not the application had been removed from the market, which also would be interesting information that could indicate malicious behaviour (note that this would not be a definitive indicator of malicious behaviour, as the application could have been removed for other reasons, or even renamed). Finally, many applications have their permissions changed between updates which we would then not be aware of.

To solve this problem a second program was created to update the data set, from now referred to as the update program. The update program connects to the database and retrieves the application identifier (appid) from each application. From this, it generates a link to the application page of each application and repeats the process of the initial parser program.

In order to make the data collection continuous it was decided to use an external server for the data collection programs. For this purpose we decided to use cloud computing services, and our supervisor provided us with a server instance in the Amazon EC2 [4] which we could use for the duration of this project. This server provided us with the stability that running the applications from our own computers would not be able to provide.

Additionally, running the applications in the cloud means that when this project is finished, our supervisor can continue to gather data without the data set being disrupted in any way, if so desired.

The data collection was structured in a bash script running every fifteen minutes, which executed the java applications for each of the markets. If any errors occurred, the error message was written to a log file. This log was sent to the authors' e-mail accounts, along with additional information regarding the affected

application. The syslog was also included in order to pick up any other log entries from the server.

An unintended, but positive, side effect of using an Amazon EC2 instance was that the server farm the instance was located at was in the United States. This meant that sites like Google Play, which uses the IP address of the customer to identify his or hers nationality and uses this to determine which currency to display prices in, listed the prices of the applications in U.S. dollars. This brought it in line with the remaining markets which listed their prices in the same currency.

4.1.2 Malicious data set

Several sources were considered for use as a malicious data set. The candidates included Symantec's Threat Explorer database [82], F-Secures Threat Description database [27] and similar sources, in addition to the Contagio Mobile Dump [59]. The databases of Symantec and F-Secure were ultimately decided against because it was impractical to automatically collect information from these databases. Additionally, the technical details were written by hand by the researchers and as such the information was inconsistent as to whether or not they listed the permissions requested by the malware. In some cases the permissions were listed by a screenshot of the malicious application and not otherwise listed in the documentation, and in other cases the permissions were presented as a list of permissions "that the malware might ask for". As such, most of this work would have to be done by hand, and for that reason it was considered too time-consuming for the purposes of this project. Exceptions to this were made in special cases where the information from the anti-virus companies was compared to malicious applications from our data set.

Due to these issues, the Contagio website was used as the source of our malicious data set. Contagio collects and presents samples of malicious applications uploaded to the website by the public, and anyone can download these samples from their database. This allowed us to retrieve the permissions easily from each application, as described in the next section.

Collected application information. We manually downloaded a collection of 160 infected applications from the Contagio Mini Dump [59] website. These applications were used to create a data set containing only malicious applications,

with the following information:

- Malware name
- Permissions requested
- Original package name where possible

The name of the malware was retrieved alongside the APK file from the Contagio website. From this file, the original package name and the permissions requested were retrieved as described in the next section.

Despite its small size, this data set contained multiple instances of some Trojans, including Geinimi [34] and OzotShielder (also called Kmin) [79]. We initially assumed that this was because of the spread of the malware, but while this can be correct for Geinimi, which have received a lot of press attention, OzotShielder appears to have infected fewer devices.

Data collection programs. Unlike the process used with the market data set, the malicious applications were retrieved by hand from the Contagio website.

The objective became to identify which permissions were requested by each application without attempting to install each and every one on an Android device. For this reason the Apktool [11] program was used to retrieve the manifest XML file from the application. Apktool is a decompiler targeting Android applications, and makes it possible for us to easily extract the manifest XML file contained inside the APK. The manifest is the file that, as mentioned in Section 3.3.1, contains all the permissions requested by the application during installation.

To automate this process we wrote another custom program which leveraged both the Apktool program to retrieve the manifest and the Jericho parser to extract the permissions from the manifest. The information retrieved from the malicious applications was then labeled as malicious and inserted into the database alongside the applications collected from the regular markets.

4.2 Data Sorting

Before the information could be used for our purposes, it had to be sorted, duplicate information needed to be removed and inconsistent names had to be nor-

malised.

4.2.1 Removing duplicates

The market data set consisted of a lot of duplicate data, because the data collection was done every fifteen minutes even if there were no new applications added to the markets. The data were stored in two different tables, one containing the application information (from now referred to as app_info), and one containing the application permissions (from now referred to as app_permission).

In the app_info table, each application entry is a single row containing all the general information about the application, while in the app_permission table each permission has its own row with information connecting it to the specific application it belongs to. This translates to several more rows in the app_permission table compared to the app_info table. This was necessary because the alternative was to make an additional column for each permission. This would mean that the authors would need to know beforehand which permissions would be requested, which is practically impossible as discussed in Section 4.2.2. While this is done in the table combining the permissions and the application information (app_combined), this is done after evaluating all the retrieved permissions.

To remove duplicate entries from the data set the data entries from the application_info table are sorted into a new table (from now referred to as app_info_u), where duplicate entries are collapsed into unique entry. This table keeps the most recent information about an application, in addition to keeping track of the first and last time the application were spotted on the market it was retrieved from.

Note that when this process was run on the malicious data set, the data set was reduced to 105 applications. This was because it turned out that some of the malicious application files retrieved from Contagio were different samples of the same application. A good example of this was the OzotShielder Trojan [79], which as mentioned earlier was represented in the malicious data set by a large collection of samples. These samples turned out to be multiple samples of a small set of applications, and thus were collapsed into a few entries.

The app_info_u and app_permission tables were then merged into one table called app_combined, where the all the application information was stored in a single row. This made it possible to use the data in the clustering algorithms as covered

in Section 5.4.2. Instead of having one row for each of the permissions of the application, the presence or lack of a permission is indicated by a “1” or a “0” in the respective column, with “1” indicating that the application requested this permission, and “0” indicating that this permission was not requested. The next section will explain the process used to determine which permissions should be included.

4.2.2 Permission filtering

Inserting all of the permissions into one table presented a new challenge. The permissions would need to be sorted and examined, in order to decide which permissions should be included in the table. Including all the permissions regardless of their validity was considered, but it was feared that this would complicate both the statistical analysis and the machine learning process.

As explained in Section 3.3.1, third-party permissions, i.e. permissions declared by the developers, are not declared to the users during installation or some of the market websites, like Google Play. This also means that the web scrapers utilized by this project would also be unable to retrieve these permissions, except if they were explicitly stated by one of the third-party markets. In order to keep the markets as compliant to each other as possible, it was concluded that only the official Android permissions should be included. Essentially, this means that only permissions with the “android.permission” prefix were included in the permission list. However, this also removed permissions declared by Google applications like the stock Browser, the Gmail client and Google Maps.

Because the Android SDK does not inform the developer whether or not a permission native to Android requested in the manifest is written correctly or not, the permissions retrieved would need to be vetted before they could be used. If the permission is spelled incorrectly, Android simply assumes that the developer is requesting a custom permission. This problem, combined with the lack of any reliable documentation on which permissions are actually considered a part of the Android system, meant that every permission requested by applications from the third-party markets and the malicious applications from the Contagio website required a manual check in order to certify that it was an actual permission and not a mistake.

Initially, the only source of the permissions used for this check was the official

permission manifest [40], and any permissions not included in this list was assumed to be either mistakes or deprecated permissions. However, this turned out to be false, as this reference list is incomplete. The list was then expanded to contain all the permissions we could find documentation on.

The final list does contain deprecated permissions, but these permissions have been requested by applications submitted to the market during the two months the data gathering applications were running. As 93.3 percent of the Android devices in circulation are using older versions of Android than the newest version [42], 4.0.4 at the time of writing, this is to be expected. Additionally, some of the permissions from the permission manifest are not requested by any application in the data sets, but has been included for effect.

The permissions were checked against four sources:

- The official permission manifest provided by Google [40]
- The Google Play marketplace [39]
- A permission map provided by our supervisor [17]
- The Android Permissions Demystified Permission Map [29]

The permission was first checked against the official permission manifest provided by Google [40]. If the permission was not on this list, it was checked against the Google Play market. This was done first by checking its Google Play description in the data set against the list of permissions used as a part of the "Is this application safe?" project [17]. If the description was not found in that list, its accompanying permission identifier was checked against the "Android Permissions Demystified" permission map created by Felt et al. [29], to make sure that the identifier was correct.

In the cases where the permission description did not have a corresponding permission identifier in the permission list, a permission identifier was derived from the description based on other permissions and their accompanying Google Play description. This ID was then checked against the permission map. The reason this works, is because the permission descriptions used by Google are paraphrasing the permission identifier, making it relatively easy to derive the identifier from the description.

Data set	Count
Regular	26,333
Malicious	105

Table 4.1: The total number of applications in the data set

Market	Count
Amazon	483
F-Droid	39
Google Play	23,769
SlideMe	2,042

Table 4.2: The number of applications collected from each market (not counting malware)

In the cases where the permission did not have an accompanying description, i.e. when permissions were retrieved directly from the manifest file, the existence of the permission was checked against the permission map. Any permission which had at this point not been found was not included in the app_combined table. The removed permissions were however kept in the original table for further analysis.

The first permission list contained 151 permissions, but after these were checked against the documentation and all undocumented and misspelled permissions were removed, the list was reduced to 137.

This process uncovered weaknesses in the Google Play market which will be discussed in Section 6.1.1.

4.3 Final Data Set

After the data collection phase was finished, we ended up with a smaller malicious data set than we would have liked, as seen in Table 4.1. However, this should not interfere with the analysis.

It is important to note that while the malicious data set contains a wide variety of Trojans, the data set also contains a large samples of some specific infections, among them Geinimi and Pjapps.B. While this could theoretically skew the analysis results for some of the permissions, there are no duplicate malicious applications in the data set and the duplicate infections will be taken into account during the analysis.

As we see in Table 4.2, the amount of applications retrieved from each market are also very diverse, ranging from F-Droid, which only provided us with thirty-nine applications, even fewer than our malicious data set, to Google Play, which was the major contributor with 23,769 applications. As explained in Section 4.1.1, the various markets have different philosophies, and this affects the number of applications submitted and accepted to the markets.

The final list of permissions considered in the analysis can be found in Appendix B.

Chapter 5

Analysis

In this chapter we will use the data we have gathered and attempt to find trends and patterns which separate malicious from legitimate applications. For the sake of brevity, the "manifest.permission" part of the permission identificators has been omitted. This is illustrated by the "manifest.permission.INTERNET" permission, which is shortened to simply "INTERNET". This is only done for permissions native to Android; any third-party applications discussed will still have their complete designation. As an example, com.motorola.launcher.permission.READ_SETTINGS will not be shortened. This is done to avoid confusion when permissions from different third-parties have the same identifier.

5.1 Permission Statistics

Appendix B presents, in percentages, how many times each permission has been requested by an application in the regular market data set compared to the malicious data set. Due to the comparatively small size of the malicious data set, the numbers were converted to percentages in order to provide a better illustration of the differences between the data sets. From these percentages it can be observed that there is a clear divergence between the two samples, and that some permissions are over-represented in one sample compared to the other. While, as mentioned before, the malicious data set might be misrepresenting the number of times some permissions are present in the data set, it is a decent guideline when used alongside and compared to the regular data set which contains a larger and

Data set	Average # of permissions	Highest #	Lowest #
Markets	4.33	36	0
Malicious	10.35	106	1

Table 5.1: Average number of permissions by data set, and the highest and lowest number of requested permissions.

more balanced number.

An important point to make is that all the permissions requested by a malicious application are not necessarily required for it to do its dirty work. When a Trojan is attached to a legitimate application, the resulting application requires the permissions of the original application in addition to the permissions it requires for its own purposes. This translates to a significant deviation in the amount of permissions requested by each application. This is presented in Table 5.1, and the difference in the number of permissions requested have shown to be statistically significant (Student's t-test, $p < .0001$).

The extra "legitimate" permissions requested by malicious applications may hamper the work done in this project, as we will need to differentiate between permissions required by the malicious code and the permissions required by the original application. This is where the duplicate infections mentioned in Section 4.3 helps us, as we can discount any permissions not uniformly requested by the malware samples.

The subsequent section will look at specific subsets of the permissions.

5.1.1 Permissions used only by malware

Initially, the permissions used only by malicious applications were thought to be significant. This turned out to be wrong, as the most divergent permissions, like ACCESS_GPS and ACCESS_LOCATION, were removed from the data set because they were undocumented.

The remaining permissions requested only by malware are presented in Table 5.2, and are far less conclusive. This view is strengthened by the Arspam Trojan, which is requesting eleven of these permissions, only three of which are also requested by another application. The Arspam Trojan is designed to send spam text messages to the contacts on the device [74], but requests a total of 106

Permission	# of malicious applications
ACCESS_DOWNLOAD_MANAGER	1
BRICK	1
DIAGNOSTIC	1
GLOBAL_SEARCH_CONTROL	1
INJECT_EVENTS	2
INTERNAL_SYSTEM_WINDOW	1
MASTER_CLEAR	1
MOUNT_FORMAT_FILESYSTEMS	1
SET_ALWAYS_FINISH	2
SET_ANIMATION_SCALE	1
SET_PROCESS_LIMIT	2
SIGNAL_PERSISTENT_PROCESSES	1
SUBSCRIBED_FEEDS_WRITE	1

Table 5.2: Permissions used only by malicious applications

permissions. This implies that the developer of the Trojan was inexperienced.

5.1.2 Analysis of permissions

During the course of the analysis, three sets of permissions stood out due to either the frequency of the requests or inconsistencies in how they were used. This section will look at these permissions.

Text messaging permissions. This covers four permissions:

- SEND_SMS
- RECEIVE_SMS
- READ_SMS
- WRITE_SMS

These permissions are requested by a lot of the malicious applications, and comparatively few of the applications form the market data set. Of these permissions the SEND_SMS permission is the most prolific, with 1.55 percent of the market data set and 61.90 percent of the malicious data set, which makes sense as an attacker leveraging text messages need this permission to send them from the device.

The remaining permissions are not as frequently requested, but also contain capabilities that are useful for malware developers. The RECEIVE_SMS (1.34 percent of the market data set and 35.24 percent of the malicious data set) and READ_SMS (0.87 percent of the market data set and 48.57 percent of the malicious data set) permissions allow the application to intercept any responses to the initial text message, and effectively hide the text message exchange from the user. The WRITE_SMS is the least requested permission in this set, with 0.39 percent of the market dataset and 19.05 percent of the malicious dataset requesting it. This permission is required if the attacker intends to sign the victim up for a premium SMS service instead of utilizing the trojan to send single-charge messages.

READ_PHONE_STATE. Before Android version 1.6, this permission was automatically granted to all applications, alongside the WRITE_EXTERNAL_STORAGE permission [12]. It provides any application requesting it with access to a large amount of data, including among other things the IMEI and number of the device, and information from the SIM card like the IMSI number. This is information that should not be shared, yet this permission is often misused to provide a unique identifier for the device and/or user.

Out of the ten advertisement networks listed in Section 5.3.1, six of them request this permission, and it is optional for two more. These networks are using it to uniquely identify users; the AirPush and TapJoy networks use the IMEI number of the device [2, 83], and MobClix requires it for an unspecified unique identifier [61].

There is a built in function to uniquely identify users without the application needing the READ_PHONE_STATE permission, called the "ANDROID_ID", which would be uniquely generated by each device. This function is however seldom used, as it was discovered that under some circumstances the generated identifier would not be unique for the Android device [48]. This was fixed in Android version 2.3, but roughly twenty-five percent of all android devices [42] are still running version 2.2 or older of the operating system.

Install and delete packages. This covers two permissions:

- INSTALL_PACKAGES

- DELETE_PACKAGES

The INSTALL_PACKAGES permission is requested by 19.05 percent of the malicious applications, largely the Pjapps Trojan, while it is only requested by 0.49 percent of the regular data set. The DELETE_PACKAGES permission is however only requested by 1.35 percent of the market data set and 2.86 percent of the malicious data set.

The subsequent observations comes with a caveat; these permissions should never be usable by a malicious application because of the protection level of the permissions. However, malicious applications are requesting the permissions and analysis done by anti-virus companies on these applications [21, 77] imply that the applications are able to use these permissions. The authors have been unable to verify whether or not this is correct.

The INSTALL_PACKAGES permission, if granted, allows the application to install new applications on the device. According to anti-virus companies, Trojans are using this capability to install backdoors onto infected devices. It is important to note that an application installed via this method is not limited to the permissions of the "parent" application.

The DELETE_PACKAGES allow the application to remove other applications from the device. A malicious application could use this to remove anti-virus application or other applications limiting its capabilities.

In order for these permissions to be granted, the application need to either be installed in the system partition of the operating system, or signed with the same key, see Section 3.3.1 as the operating system. Neither of these conditions is met by malicious applications when they are installed on a device by a regular user. A Trojan utilising a root exploit could install itself in the system partition, but would technically not need the permission.

5.2 A Closer Look at Malicious Applications

Seven of the sample Trojans were chosen for detailed study. Five of the Trojans were selected due to the presence of more than one sample in the malicious data set, while the remaining two Trojans were selected because one or more anti-virus companies had published detailed analysis on the malicious application.

Trojans with more than one instance has been chosen because the samples can be compared to each other in order to weed out some of the permissions which are not required by the Trojan itself.

It must be stated that the malicious applications are requesting several other permissions in addition to the permissions we include in our analysis, as discussed in Section 4.2.2. These permissions are not included in our analysis. A full overview of the permission requested by the malicious applications is listed in Appendix C.

5.2.1 CounterClank/Apperhand

CounterClank is a special case of malware, as there is not consensus among antivirus providers as to whether or not it is a Trojan, as Symantec states [75], or very aggressive adware (advertisement software) as Lookout [54] and Sophos [73] states. Now, aggressive adware is the best description for it, as the functionality included in the Apperhand framework is becoming a part of legitimate advertisement networks like StartApp and AirPush, as described in Section 5.3.1.

Despite this, CounterClank/Apperhand, and by extension, AirPush and StartApp, is not something a user would want to have on his device. The advertisements delivered by these advertisement networks are excessively intrusive; Apperhand has the capability to display ads in the notification bar on the device, add shortcuts to the home screen and add bookmarks to the browser.

The capabilities described here explain why Apperhand is requesting an excessive amount of third party permissions, which can be viewed in Appendix C. These permissions relates to both the stock launcher, Trebuchet, and third-party launchers like HTC, LG, Motorola and the highly popular ADW launcher in addition to the stock Android internet browser.

As mentioned earlier, in Section 4.2.2, we are not tracking third-party permissions at this point. Of our five CounterClank samples, the only constant Android permissions requested by the application are ACCESS_NETWORK_STATE, INTERNET and READ_PHONE_STATE. These are not specific enough to reliably identify CounterClank in the market, as are also shown in Table 5.3, these are very common permissions. This is demonstrated in Section 5.4.1.1.

The nature of the third-party permissions requested by CounterClank does how-

Permission	% in market	% in malware
ACCESS_NETWORK_STATE	56.62	41.90
ACCESS_WIFI_STATE *	10.08	25.71
INTERNET	84.05	91.43
READ_PHONE_STATE	44.03	78.10
WAKE_LOCK *	20.57	24.76
WRITE_EXTERNAL_STORAGE *	44.48	65.71

Table 5.3: Permissions requested by our CounterClank samples compared to the frequency of these permissions in the data sets. * not present in all samples

Permission	% in market	% in malware
ACCESS_NETWORK_STATE *	56.62	41.90
ACCESS_WIFI_STATE	10.08	25.71
CHANGE_WIFI_STATE	1.61	11.43
INTERNET	84.05	91.43
KILL_BACKGROUND_PROCESSES *	1.85	2.86
READ_CONTACTS *	6.40	43.81
READ_LOGS *	1.61	9.52
READ_PHONE_STATE	44.03	78.10
RESTART_PACKAGES *	0.13	10.47
WRITE_CONTACTS	30.56	30.48

Table 5.4: Permissions requested by our DroidDream samples compared to the frequency of these permissions in the data sets. * not present in all samples

ever mean that the applications could be recognizable with an extended data set containing third-party permissions, given that the third-party permissions can be retrieved similarly to the default Android permissions.

5.2.2 DroidDream/Rootcager

DroidDream [80] is a Trojan that leverages two root exploits, *rageagainstthecage* and *exploid*, to gain control of the device. As mentioned in Section 3.4.3, this means that DroidDream gained full control of the device if the root exploits succeeded. This Trojan is also special in that it is one of the few Trojans in the malicious data set that spread through the Google Play market, instead of just through third-party markets.

As DroidDream uses a root exploit to do its dirty work, it does not require a lot of permissions to run. And as it is attached to other, legitimate applications, most of the permissions requested are specific to the original applications. In our data

set we have to two samples of DroidDream, which only have the INTERNET, ACCESS_WIFI_STATE, READ_PHONE_STATE and CHANGE_WIFI_STATE in common, as shown in Table 5.4. As with CounterClank, and demonstrated in Appendix B, these permissions are relatively common. As we only have two samples, it is also possible that the presence of these permissions in both samples is purely accidental. However, it is rare for these permissions to be requested together, as demonstrated in Section 5.4.1.2.

5.2.3 Geinimi

Geinimi [34] is found attached to repackaged legitimate applications. The malware has not been found in the Google Play market, but has been found in third party markets, mainly Chinese. The Trojan has many capabilities, but these capabilities are hidden until it is instructed by a C&C server (Command and Control server) to utilize them. The capabilities range from making premium phone calls to stealing personal data like IMEI number and the geographic location of the device. As shown in Table 5.5, there are a lot of permissions requested by the Geinimi infected applications, but only a handful of them are requested by every sample.

As already mentioned, the Geinimi Trojan is well represented in the data set, with twenty-four samples. We can assume that the permissions shared by all the samples are required by the Geinimi Trojan, and this provides us with a good pattern as demonstrated in Section 5.4.1.3. Several permissions were removed due to being undocumented; ACCESS_COARSE_UPDATES, ACCESS_GPS, ACCESS_LOCATION, see Section 4.2.2.

5.2.4 GoldDream

The GoldDream [77] Trojan infects devices through repackaged versions of legitimate Android games. Once installed on a device, it begins to monitor phone calls and text messages, and transmits information like phone numbers, text message contents and duration of phone calls to an external server.

This Trojan is represented by two samples in the data set, one sample of GoldDream.A and one sample of GoldDream.B. The permissions requested by these samples are however nearly identical, as seen in Table 5.6, with the only difference being the WAKE_LOCK permission. For the purpose of this analysis this

Permission	**% in market**	**% in malware**
ACCESS_COARSE_LOCATION	11.70	41.90
ACCESS_FINE_LOCATION	22.77	39.05
ACCESS_NETWORK_STATE *	56.62	41.90
CALL_PHONE	8.21	31.43
CAMERA *	6.40	6.67
GET_ACCOUNTS *	7.55	5.71
GET_TASKS *	2.34	7.62
INTERNET	84.05	90.57
MODIFY_AUDIO_SETTINGS *	1.72	3.81
MODIFY_PHONE_STATE *	0.25	5.71
MOUNT_UNMOUNT_FILESYSTEMS	0.88	29.52
READ_CONTACTS	6.40	43.81
READ_LOGS *	1.61	9.52
READ_PHONE_STATE	44.03	78.10
READ_SMS *	0.87	48.57
RECEIVE_BOOT_COMPLETED *	18.85	22.86
RECEIVE_SMS *	1.34	35.24
RECORD_AUDIO *	2.86	3.81
REORDER_TASKS *	0.02	1.90
RESTART_PACKAGES *	0.13	10.48
SEND_SMS	1.55	61.90
SET_WALLPAPER	3.77	28.57
SYSTEM_ALERT_WINDOW *	0.96	3.81
VIBRATE *	24.52	20.95
WAKE_LOCK *	20.58	24.76
WRITE_APN_SETTINGS *	0.03	9.52
WRITE_CONTACTS	3.56	30.48
WRITE_EXTERNAL_STORAGE	44.48	65.71
WRITE_SMS *	0.39	19.05

Table 5.5: Permissions requested by our Geinimi samples compared to the frequency of these permissions in the data sets. * not present in all samples

Permission	% in market	% in malware
ACCESS_COARSE_LOCATION	11.70	41.90
ACCESS_FINE_LOCATION	22.70	39.05
ACCESS_NETWORK_STATE	56.62	41.90
ACCESS_WIFI_STATE	10.08	25.71
CALL_PHONE	8.21	31.43
DELETE_PACKAGES	1.35	2.86
INSTALL_PACKAGES	0.49	19.05
INTERNET	84.05	91.43
PROCESS_OUTGOING_CALLS	0.51	5.71
READ_PHONE_STATE	44.03	78.10
READ_SMS	0.87	48.57
RECEIVE_BOOT_COMPLETED	18.85	22.86
RECEIVE_SMS	1.34	35.24
SEND_SMS	1.55	61.90
WAKE_LOCK *	20.58	24.76
WRITE_EXTERNAL_STORAGE	44.48	65.71

Table 5.6: Permissions requested by our GoldDream samples compared to the frequency of these permissions in the data sets. * not present in all samples

minor difference was considered irrelevant, as a pattern based on the remaining permissions will still only match the two GoldDream samples (discounting the Arspam Trojan), as demonstrated in Section 5.4.1.4.

5.2.5 Pjapps

In the data set we can identify three different versions of the Pjapps Trojan; Pjapps.A [8], Pjapps.B [21] and Pjapps.C [56]. There are three samples of Pjapps.A, ten samples of Pjapps.B and two samples of Pjapps.C.

The Pjapps Trojan establishes a botnet of infected devices. When an application infected with Pjapps is installed on a device, the Trojan sends private information like the IMEI number and the SIM serial number to a C&C server and awaits further commands. These commands include sending text messages, installing applications and adding bookmarks to the browser.

Pjapps.A. The first version of Pjapps, named just Pjapps or in some cases Pjapps.A, was used to establish a botnet consisting of compromised Android devices [8]. The Trojan established a back-door on compromised devices, allowing the developer to send SMS messages, install applications and steal information

Permission	% in market	% in malware
ACCESS_NETWORK_STATE *	56.62	41.90
ACCESS_WIFI_STATE *	10.08	25.71
CHANGE_NETWORK_STATE *	0.52	4.76
CHANGE_WIFI_STATE *	1.61	11.43
DISABLE_KEYGUARD *	0.51	4.76
INSTALL_PACKAGES	0.49	19.05
INTERNET	84.05	91.43
READ_PHONE_STATE	44.03	78.10
RECEIVE_MMS	0.05	7.62
RECEIVE_SMS	1.34	35.24
SEND_SMS	1.55	61.90
SET_PREFERRED_APPLICATIONS *	0.28	1.90
VIBRATE *	24.52	20.95
WAKE_LOCK *	20.58	24.76
WRITE_APN_SETTINGS *	0.03	9.52
WRITE_EXTERNAL_STORAGE	44.48	65.71

Table 5.7: Permissions requested by our Pjapps.A samples compared to the frequency of these permissions in the data sets. * not present in all samples

like IMEI number, device ID and subscriber ID. As demonstrated in Table 5.7, Pjapps.A requests the INSTALL_PACKAGES permission, which means that it can install additional applications on the device without the involving the device owner. It is however unclear whether or not this capability would actually be granted to the application, as discussed in Section 5.1.2.

Note that while all our samples request the RECEIVE_MMS permission, this is not functionality which is a default part of the Trojan.

Pjapps.B. Pjapps.B operates in a similar manner as Pjapps.A. According to F-Secure, the biggest difference between Pjapps.B and Pjapps.A is that Pjapps.B requests to start at boot. This could indicate that the Pjapps.B samples have been given the wrong label, and might actually be samples of Pjapps.A, as only one of them are requesting the RECEIVE_BOOT_COMPLETED permission. However, comparing Table 5.7 and Table 5.8, the samples are so close to each other that this was not considered a problem, as any pattern based on these two applications would be virtually indistinguishable.

Pjapps.C. The Pjapps.C Trojan stands out, as there is very little information available beyond its name [56]. It is also deviating from the older versions of Pjapps, as only one of the two samples request the INSTALL_PACKAGES permission, as demonstrated in Table 5.9.

The three Pjapps versions also contained three permissions that were not included

Permission	% in market	% in malware
ACCESS_COARSE_LOCATION *	11.70	41.90
ACCESS_FINE_LOCATION *	22.77	39.05
ACCESS_NETWORK_STATE *	56.62	41.90
ACCESS_WIFI_STATE *	10.08	25.71
BLUETOOTH *	0.47	6.67
BLUETOOTH_ADMIN *	0.28	5.71
CAMERA *	6.40	6.67
DISABLE_KEYGUARD *	0.51	4.76
FLASHLIGHT *	2.94	2.86
INSTALL_PACKAGES	0.49	19.05
INTERNET	84.05	91.43
KILL_BACKGROUND_PROCESSES *	1.85	2.86
READ_CONTACTS *	6.40	43.81
READ_PHONE_STATE	44.03	78.10
READ_SMS *	0.87	48.57
RECEIVE_BOOT_COMPLETED *	18.85	22.86
RECEIVE_MMS *	0.05	7.62
RECEIVE_SMS	1.34	35.24
RESTART_PACKAGES *	0.13	10.48
SEND_SMS	1.55	61.90
SET_WALLPAPER *	3.77	28.57
VIBRATE *	24.52	20.95
WAKE_LOCK *	20.58	24.76
WRITE_EXTERNAL_STORAGE	44.48	65.71
WRITE_SETTINGS *	2.32	9.52
WRITE_SMS *	0.39	19.05

Table 5.8: Permissions requested by our Pjapps.B samples compared to the frequency of these permissions in the data sets. * not present in all samples

Permission	% market	% malware
ACCESS_COARSE_LOCATION	11.70	41.90
ACCESS_DOWNLOAD_MANAGER *	0	0.95
ACCESS_DRM *	0.004	0.95
ACCESS_FINE_LOCATION *	22.77	39.05
ACCESS_NETWORK_STATE	56.62	41.90
ACCESS_WIFI_STATE *	10.08	25.71
CHANGE_NETWORK_STATE *	0.52	4.76
INSTALL_DRM *	0.008	0.95
INSTALL_PACKAGES *	0.49	19.05
INTERNET	84.05	91.43
READ_CONTACTS *	6.40	43.81
READ_PHONE_STATE *	44.03	78.10
READ_SMS *	0.87	48.57
RECEIVE_BOOT_COMPLETED	18.85	22.86
RECEIVE_SMS	1.34	35.24
RECEIVE_WAP_PUSH *	0.004	4.76
SEND_DOWNLOAD_COMPLETED_INTENTS*	0.008	0.95
SEND_SMS	1.55	61.90
VIBRATE *	24.52	20.95
WRITE_APN_SETTINGS *	0.03	9.52
WRITE_CALENDAR *	0.67	1.90
WRITE_CONTACTS	3.56	30.48
WRITE_EXTERNAL_STORAGE	44.48	65.71
WRITE_SETTINGS *	2.32	9.52
WRITE_SMS *	0.39	19.05
WRITE_SYNC_SETTINGS *	0.15	3.81

Table 5.9: Permissions requested by our Pjapps.C samples compared to the frequency of these permissions in the data sets. * not present in all samples

Permission	% in market	% in malware
ACCESS_NETWORK_STATE	56.62	41.90
ACCESS_WIFI_STATE	10.08	25.71
BROADCAST_PACKAGE_REMOVED	0.01	2.86
CHANGE_WIFI_STATE	1.61	11.43
DEVICE_POWER	0.18	4.76
INTERNET	84.05	91.43
KILL_BACKGROUND_PROCESSES	1.85	2.86
READ_PHONE_STATE	44.03	78.10
READ_SMS	0.87	48.57
RECEIVE_SMS	1.34	35.24
SEND_SMS	1.55	61.90
WAKE_LOCK	20.58	24.76
WRITE_APN_SETTINGS	0.03	9.52
WRITE_EXTERNAL_STORAGE	44.48	65.71
WRITE_SMS	0.39	19.05

Table 5.10: Permissions requested by our adSMS sample compared to the frequency of these permissions in the data sets.

in the data set due to being undocumented; ACCESS_CACHE_FILESYSTEM, ACCESS_DOWNLOAD_MANAGER_ADVANCED, WRITE_OWNER_DATA. See Section 4.2.2 for details on why these permissions were discounted.

5.2.6 adSMS

While adSMS is only represented with one sample in the malicious data set, the FortiNet team has a good write-up on the malware [33] which will be used for this analysis. The adSMS Trojan pretends to be a system update, which in the malicious data set is represented with a file name that implies that it is issued by HTC. Once installed on a device, the malware waits until one of two conditions are met. When the device is rebooted or the device receives an SMS, the malware springs into action. It registers the phone to certain services using SMS, and proceeds to steal private information from the device. The malware will also attempt to stop other processes on the device, usually processes related to a Chinese messaging service called QQ.

Unlike the malicious applications analysed earlier in this chapter, this malware is represented by only one sample. This means that unlike them, all the permissions requested by this sample, shown in Table 5.10 are all significant for the malware. This is not saying that these permissions are all requested by all versions of

Permission	% in market	% in malware
SEND_SMS	1.55	61.90
INTERNET	84.05	91.43
RECEIVE_SMS	1.34	35.23

Table 5.11: Permissions requested by our JimmRussia sample compared to the frequency of these permissions in the market data set.

the malware, but the permissions are consistent between this sample and the permissions described by FortiNet [33].

5.2.7 JimmRussia

Like adSMS, this Trojan is only represented by one sample in the data set. This is accurate, as there is only one known Jimm infection, "Jimm Russia" [63]. The malware spread on Russian forums and on Russian websites, pretending to be a version of the messaging client "Jimm". When the Trojan is installed on a device it sends text messages to premium numbers.

This malware is very specialised, and only ask for three permissions, as shown in Table 5.11. This is lower than both the average of the market data set and the malicious data set, as seen in Table 5.1. The combination of the two text messaging permissions are however unusual, as demonstrated in Section 5.4.1.7.

5.3 A Closer Look at Potentially Suspicious Applications

This section will elaborate on advertisement networks and application builders.

5.3.1 Advertisement networks

In this analysis we also need to take advertisement networks into account, as many free applications implement these networks. Ironically, these advertisement networks are added on to applications in a similar fashion as Trojans, but unlike Trojans these are in general considered a reasonable trade-off for users as advertisement supported applications are usually free. Table 5.12 presents the ten most popular advertisement networks and the percentage of applications that contain them, as determined by a project done by AppBrain [84]. Based on Table 5.12, the permissions of these ten advertisement networks were retrieved,

Advertisement network	% of applications
Admob	39.2
AirPush	6.5
Millennial Media	5.1
LeadBolt	4.1
AdWhirl	3.9
MobClix	3.5
InMobi	2.6
MobFox	1.8
Tapjoy	1.5
Startapp	1.2

Table 5.12: The ten most popular Android advertisement networks, by the percentage of applications that use them [84]

see Table 5.13, and compared to our data set. In Table 5.13, LAUNCHER includes installing shortcuts, reading home settings and shortcuts, and uninstalling shortcuts. BROWSER includes read- and write-access to browser bookmarks and history.

Table 5.13 lists the permissions requested by the advertisement networks identified in Table 5.12. As shown in the table, the INTERNET permission is requested by all ten networks. This is expected, as the advertisement networks would need a way to retrieve new ads from the servers. Additionally, Appendix B shows that the INTERNET permission is requested by 84.05 percent of all the applications we sampled from the markets. Additionally, the READ_PHONE_STATE and the ACCESS_NETWORK_STATE permissions are well represented, with ACCESS_NETWORK_STATE present in the top four advertisement networks. Using Table 5.13, we can make a rough calculation using the percentage of applications that contain the top four networks and assume that 54.2 percent of the applications asking for INTERNET and ACCESS_NETWORK_STATE contain these four advertisement networks. Comparatively, of our data set, which admittedly uses a smaller data set than the 140,000 applications that Appbrain operates with, 56.6 percent of the legitimate applications and 41.5 percent of the malicious applications ask for the ACCESS_NETWORK_STATE permission.

It is worth noting that at least two of the networks, AirPush [2] and StartApp [71], include functionality that was included in the Apperhand framework. These functionalities were the major contributing factors that made Symantec classify

Permission	AM	AP	MM	LB	AW	MC	IM	MF	TJ	SA
ACCESS_COARSE		O		O	O		O			
ACCESS_FINE_LOCATION		O		O	O		O	O		
ACCESS_LOCATION_EXTRA_COMMANDS				O						
ACCESS_NETWORK_STATE	R	R	R	R			O		R	
CALL_PHONE							O			
GET_TASKS						R				
INTERNET	R	R	R	R	R	R	R	R	R	R
READ_PHONE_STATE		R	R	R	O	R		O	R	R
RECEIVE_BOOT_COMPLETED		R		O						
VIBRATE		O								
WAKE_LOCK				O						
WRITE_EXTERNAL_STORAGE			R							
LAUNCHER										R
BROWSER										O

Table 5.13: The required and optional permissions requested by the advertisement networks. From left to right; Admob [38], AirPush [2], Millennial Media [57], Leadbolt [50], AdWhirl [1], Mobclix [61], Inmobi [46], MobFox [62], TapJoy [83] and StartApp [71]. O = Optional, R = Required.

it as a Trojan (see CounterClank, in section 5.2.1). This is worrying, as AirPush is the second largest network.

There have also been controversy surrounding what information the networks are retrieving from the devices they are installed on, and for the average user it is hard to gain insight into what information is retrieved and how to opt out of the information gathering done by the advertisement networks.

5.3.2 Application builders

During the analysis of the market applications there were some applications which were asking for a very peculiar set of permissions. These applications, while having different developers, had suspiciously common application identifiers. These applications turned out to have been built by an *application builder*, a web application repackaging websites as mobile applications.

Based on installation counts and application ratings, these applications appear to have a generally low standard. Based on their application IDs, three application builders were identified in the data set; Appmakr, Appsbuilder and Appsbar. The suspicious permissions were requested by the Appmakr applications, and while these applications do not appear to be malicious, the applications did not have any obvious reason for requesting these permissions. As an example, fifty-seven of the seventy-four Appmakr applications requested the CAMERA permission, while forty-four of the applications requested the READ_LOGS permission. As described in Section 3.4.9, this permission can be used to siphon data from the device.

Comparatively, the thirty-one Appsbuilder and 666 Appsbar applications did not ask for any unusual permissions, and the requested permissions generally matched the advertisement networks covered in Section 5.3.1.

5.4 Recognizing Bad Applications

The subsequent sections describe the methods used in order to identify malicious and suspicious applications, and the results obtained from applying these methods to the data set.

5.4.1 Recognising malware by permissions

For the purpose of this analysis, a pattern is defined as a selection of permissions that are requested by the malicious application. A pattern can consist of all the permissions requested by the application, or a subset of these permissions. The patterns were created based on the permissions requested by the malicious applications in Section 5.2, and tests were done by applying these patterns to the data set, after which the results were reviewed and evaluated based on the applications that matched the pattern.

The potential of these patterns lies in the theory that malicious applications can be recognised by the combination of permissions they request, not just which permissions they request.

The false positive rate is calculated using this formula:

$$\frac{x}{26,333} \times 100 \tag{5.1}$$

x is the number of applications from the market data set which matches the pattern, and 26,333 is the number of applications in the market data set. Note that we can not guarantee that the market data set are completely infection free, the false positive rate in this circumstance means that this many of the applications would need to be reviewed for infections.

For the purpose of this analysis we have not considered malicious applications caught by the pattern of another application as a false positive, as the intention is to find patterns which will catch as many malicious applications as possible.

5.4.1.1 Pattern - Counterclank

CounterClank pattern: ACCESS_NETWORK_STATE, INTERNET, READ_PHONE_STATE.

Using this pattern against the data set resulted in 8,142 legitimate applications and forty-three malicious applications identified as CounterClank, giving us a false positive of 30.92 percent. As mentioned in the CounterClank section above, this result was not surprising.

What is surprising is that the advertisement networks requesting this combina-

tion, as shown in Table 5.12 and 5.3.1, only add up to 17.2 percent.

5.4.1.2 Pattern - DroidDream

DroidDream pattern: INTERNET, CHANGE_WIFI_STATE, ACCESS_WIFI_STATE, READ_PHONE_STATE.

This pattern is rather interesting, as it consists of only four permissions, and only one of them is overrepresented in the malicious data set. Running this pattern against the complete data set resulted in twelve malicious applications caught. Comparatively, from the legitimate market data set the pattern matched 199 applications, mostly from the Google Play market, giving us a false positive rate of 0.75 percent.

5.4.1.3 Pattern - Geinimi

Geinmi pattern: ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, CALL_PHONE, INTERNET, MOUNT_UNMOUNT_FILESYSTEMS, READ_CONTACTS, READ_PHONE_STATE, SEND_SMS, SET_WALLPAPER, WRITE_CONTACTS, WRITE_EXTERNAL_STORAGE.

The Geinimi Trojan requests a rather distinct set of permissions, many of which are not often requested by legitimate applications, as seen in Table 5.5. Using this pattern on the data set resulted in no hits from the legitimate markets, only the malicious data set. The pattern matched two additional malware samples; Universal Androot, which is infected with the Lotoor Trojan, and the Arspam malware.

5.4.1.4 Pattern - GoldDream

GoldDream pattern: INTERNET, ACCESS_NETWORK_STATE, READ_PHONE_STATE, ACCESS_WIFI_STATE, WRITE_EXTERNAL_STORAGE, ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, RECEIVE_SMS, SEND_SMS, READ_SMS, CALL_PHONE, PROCESS_OUTGOING_CALLS, DELETE_PACKAGES, INSTALL_PACKAGES, RECEIVE _BOOT_COMPLETED.

This pattern returns only the GoldDream samples and the Arspam malware from the malicious data set. Like the Geinimi pattern, this pattern can almost uniquely identify GoldDream infected applications.

5.4.1.5 Pattern - Pjapps

Pjapps pattern: WRITE_EXTERNAL_STORAGE, RECEIVE_SMS, SEND _SMS.

The initial pattern derived from the Pjapps Trojan was based on all the different versions of the Trojan, and the resulting pattern contained, like the CounterClank pattern, only three permissions. Unlike the CounterClank pattern, this pattern matched twenty-six malicious applications and only fifty-eight legitimate applications. This translates to a 0.22 percent false positive rate.

Why this pattern caught fewer applications from the legitimate markets is easily determined from Appendix B, which demonstrates that while the WRITE_EXTERNAL_STORAGE permission is rather common, RECEIVE_SMS and SEND_SMS are far less common among the applications in the market data set.

A short demonstration of why this works: Using a pattern consisting only of the RECEIVE_SMS permission matches 354 applications from the market data set and thirty-seven malicious applications, while a similar pattern consisting only of the SEND_SMS matches 409 applications and sixty-five malicious applications. However, ask for these two together, and the resulting list is reduced to 116 market applications and thirty-five malicious. Add the WRITE_EXTERNAL_STORAGE permission, which as stated earlier is a lot more common than the text messaging permissions, and the result is reduced to fifty-eight market applications and twenty-six malicious applications.

5.4.1.6 Pattern - adSMS

adSMS pattern: SEND_SMS, READ_SMS, WRITE_SMS, RECEIVE_SMS, DEVICE_POWER, WRITE_APN_SETTINGS, ACCESS_NETWORK_STATE, BROADCAST_PACKAGE_REMOVED, ACCESS_WIFI_STATE, CHANGE_WIFI_STATE, WAKE_LOCK, INTERNET, WRITE_EXTERNAL_STORAGE, READ_PHONE_STATE, KILL_BACKGROUND_PROCESSES.

This pattern is even more specific to this Trojan than the Geinimi pattern is to Geinimi, see Section 5.4.1.3, and results in only the malware sample being returned from the malicious data set and no hits in the market data set. As this effectively means that the malware can be uniquely identified based only on its permission set, this pattern is effective in determining the presence of this malware.

Permission	G	DD	CC	P	aS	JR	GD
ACCESS_COARSE_LOCATION	x						x
ACCESS_FINE_LOCATION	x						x
ACCESS_NETWORK_STATE			x		x		x
ACCESS_WIFI_STATE		x			x		x
BROADCAST_PACKAGE_REMOVED					x		
CALL_PHONE	x						x
CHANGE_WIFI_STATE		x					
DELETE_PACKAGES							x
DEVICE_POWER					x		
INSTALL_PACKAGES							x
INTERNET	x	x	x		x	x	x
KILL_BACKGROUND_PROCESSES					x		
MOUNT_UNMOUNT_FILESYSTEMS	x						
PROCESS_OUTGOING_CALLS							x
READ_CONTACTS	x						
READ_PHONE_STATE	x	x	x		x		x
READ_SMS					x		x
RECEIVE_BOOT_COMPLETED							x
RECEIVE_SMS				x	x	x	x
SEND_SMS	x			x	x	x	x
SET_WALLPAPER	x						
WAKE_LOCK					x		
WRITE_APN_SETTINGS					x		
WRITE_CONTACTS	x						
WRITE_EXTERNAL_STORAGE	x			x	x		x
WRITE_SMS					x		

Table 5.14: The seven patterns compared to each other. From left to right: Geinimi, DroidDream, CounterClank, Pjapps, adSMS, Jimm Russia, Gold Dream

5.4.1.7 Pattern - JimmRussia

JimmRussia pattern: SEND_SMS, INTERNET, RECEIVE_SMS.

This pattern is very similar to that of Pjapps, and the same reasoning applies here. The INTERNET permission is very common, but the SEND_SMS and RECEIVE_SMS permissions are less common. Replacing the WRITE_EXTERNAL_STORAGE permission with the more common INTERNET permission results in 139 applications returned, with ninty-eight false positives which translates to a 0.37 percent false positive rate. This shows that the INTERNET permission is slightly more common in combination with SEND_SMS and RECEIVE_SMS than WRITE_EXTERNAL_STORAGE.

5.4.1.8 Comparing the patterns

When comparing the seven patterns to each other (see Table 5.14), two interesting things become apparent. First, all the permissions requested by the Pjapps and JimmRussia patterns matches the adSMS and GoldDream patterns as well, meaning that both the adSMS and the GoldDream Trojans would also be caught

by the Pjapps and JimmRussia patterns. However, Pjapps and JimmRussia would not be caught by each others patterns.

Second, the SEND_SMS and WRITE_EXTERNAL_STORAGE permission is shared by the Geinimi, Pjapps and adSMS patterns, meaning that a pattern made from these two permissions would catch all three Trojans. From the Pjapps pattern we know that a pattern using only the SEND_SMS permission results in 409 market and sixty-five malicious applications, as demonstrated in Section 5.4.1.5. Adding the WRITE_EXTERNAL_STORAGE permission to that, the result is reduced to 212 market and fifty-three malicious applications giving a false positive percentage of 0.81. The SEND_SMS and RECEIVE_SMS permissions however, are shared by Pjapps, adSMS, JimmRussia and Gold Dream, meaning that a pattern based on these two permissions would find all four of these Trojans. From the Pjapps pattern, in Section 5.4.1.5, it has been demonstrated that asking for these two permissions together gives a result of 116 market applications and 35 malicious applications. This gives a false positive rate of 0.44 percent.

5.4.2 Analysis using clustering algorithms

The subsequent sections will describe how machine learning has been applied to the Android application data set. A clustering algorithm, k-means, is used in the process of identifying patterns in the large amounts of data. This is an algorithm based on unsupervised learning, as described in Section 3.5.2. The objective of this approach is to verify if clustering algorithms can be applied to this type of a problem, in order to heighten our understanding of the data set. The method of adapting the k-means algorithm to fit our data set will be described in the next section. When the algorithm had been applied to the data set, the results were analysed to see if the clusters gave any valuable results. This is described in Sections 5.4.2.2 through 5.4.2.6.

5.4.2.1 Determine the k number of clusters

For k-means to perform a valid clustering on our data, an appropriate value of k must be determined. Apart from some minor parameter adjustments, k is the only input required for running k-means. The k-means algorithm organizes the input data set in a k number of clusters. Determining the value of k is not an intuitive task. Because of the great order of variation in possible input data sets, it is not possible for k-means to automatically assign an appropriate value of k.

There are a number of ways to determine k in order for the algorithm to fit the input data to a certain number of clusters. One way is to take into account the type of results you want to get out of the clustering algorithm. For illustrative purposes, an example of t-shirt sizes can be studied [64]. In this case, the value of k depends on how the owner wants to segment his or her market. In the case of five clusters (XS, S, M, L, XL) the t-shirt sizes would better fit the customers.

A larger number of sizes to pick from would result in the customers having much to choose from, and a higher probability of finding a t-shirt that would fit the customer perfectly. A downside to this number of clusters would be that the manufacturing process would become more expensive, having to produce five types of t-shirts instead of a lower number.

In the case of three clusters, the pros and cons would be reversed. The t-shirt sizes would not reflect the variation in the customers, but the manufacturing costs would be decreased. When choosing a number of clusters in this case, it all comes down to what you want your results to look like. The owner has a clear understanding of what the results might look like, and therefore be able to adjust the value of k to fit the desired outcome.

For the data set gathered in this project, an equivalent understanding of how the clustering results will look can not be made. Partly because of the high dimensionality of the data, and partly because the objective of the clustering task is to find patterns or groupings not obvious to us without actually performing the clustering. A different way of determining k is to study the cost function of several clustering attempts to see how it decreases. A plot of the cost function values for a range of k might give indications as to which value of k is fitting.

The cost function is a numeric value that describes how well the algorithm performs in fitting the data into clusters. A lower cost function value answers to a better fit to the data. In one extreme case, when k is equal to N (number of objects in the data set), the cost function will give 0 as the output, indicating a perfect fit. The cost function of k-means is represented by the total within-cluster sum of squares (SSE). The cost function is given by

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x_j \in S_i} \| x_j - \mu_i \|^2 \tag{5.2}$$

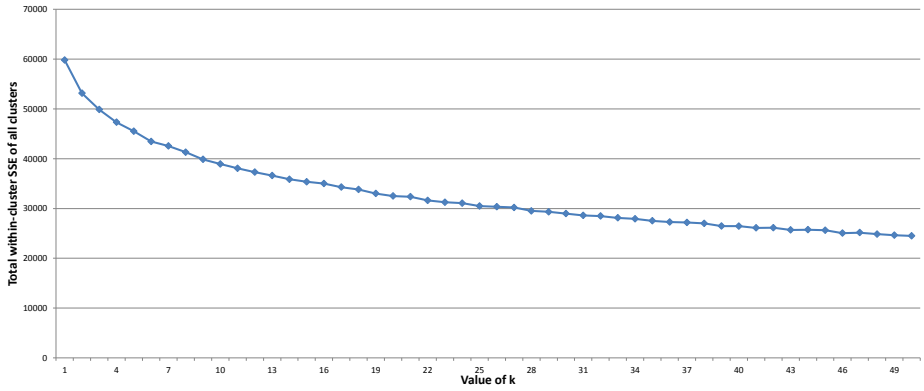

Figure 5.1: Total within-cluster sum of squares. Each point on the x-axis represents an increase in the number of k clusters.

where x_j are the data points, μ_i are the cluster centroids and S_i the sets containing the clustered data points. We calculated the cost function output for a range of k from two to fifty-one. For each k, the algorithm was run one hundred times for the purpose of increasing the confidence in the sampled data. The average for each k can be seen in Figure 5.1.

A desirable result would be that the graph suddenly levelled out after a rapid drop in the cost function, taking the form of an elbow. If the graph would have levelled out after such an elbow, we could with great confidence assume that this point was the most suitable value of k [64]. The graph clearly does not contain such an elbow.

We decided to look at how much the cost function decreases for each increase of k. That would give us an understanding of when the graph starts levelling out or to decrease by an observably steady rate. Each cost function value was subtracted with its predecessor to show how much the cost decreased between each number of k. Figure 5.2 shows the results from these calculations. Each column represents the difference in cost between a given k and its predecessor. A definite trend is evident, as the level of decrease from each k to the next is decreasing as the number of clusters increase. The graph shows a steady decrease from the first column up until column fifteen, marked red. From that point on, the cost difference varies. The average difference in decrease falls rapidly from the first to the fifteenth column. After that point, the average would be more levelled out if represented by a linear function.

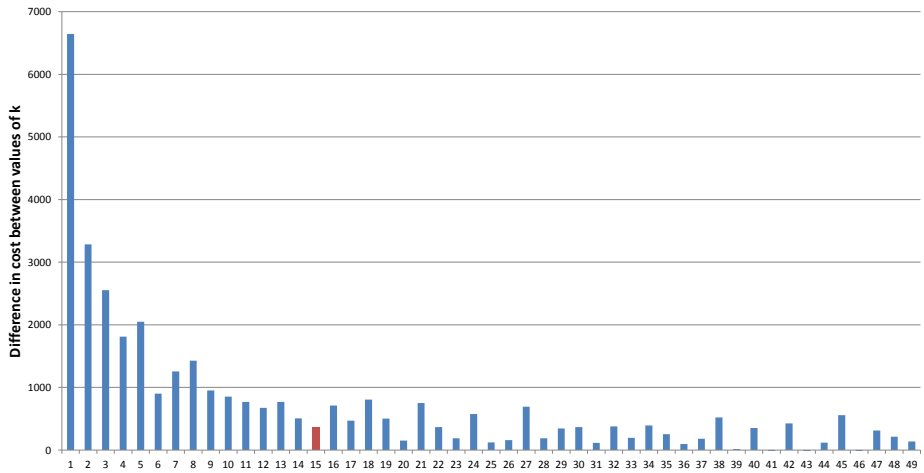

Figure 5.2: Difference in cost between each value of k. Each value on the x-axis should be read as follows: $x = cost(k+1) - cost(k)$. Column $x = 3$ therefore shows the difference in the cost between cluster 4 and cluster 3.

After studying Figure 5.2, we conclude that column 15 marks the appropriate point in which to stop. Column 15 represents the difference between k=16 and k=15. This gives k=16 as the most fitting value of k. We choose this value of k in the further work with clustering our data set by k-means.

5.4.2.2 Cluster analysis of the data set

When running the k-means algorithm on the Android applications data set, the different applications will be grouped together based on how similar they are in their permission patterns. The k-means algorithm only uses the permissions of each application as input vectors when performing the clustering. Each application is represented by a 137-dimensional vector describing each application's permission pattern. The results of the clustering algorithm therefore need to be analysed with the permissions patterns in mind. By looking at the composition of clusters, we hope to see if this type of clustering in reality groups similar applications together.

One immediate intuition is that applications with identical permission patterns will end up in the same cluster. The k-means algorithm assigns each application to the nearest cluster, based on the distance between the application and the cluster centroid vectors. Seeing as the applications vectors is composed of

the application's permissions, the distance from two applications with identical permissions to any given cluster centroid will always be the same. The results are expected to demonstrate this intuition in practice.

Figure 5.3 shows some results after the clustering has been completed. The graph is separated with respect to each market. Each column indicates the percentage of the market's applications assigned to a particular cluster. The columns are marked with its corresponding cluster number. This is done with readability in mind. For every market, the columns add up to one hundred percent of that particular market's applications.

The most apparent observation one makes from looking at the graph, is the spike in the applications from F-Droid. The spike simply implies that around sixty percent of the applications are assigned to cluster 10. If the total number of F-Droid applications in cluster 11 is considered, they only make up a mere 0.6 percent of the total number of applications in the cluster. The reason for this distinct spike is the comparatively small number of applications collected from the F-Droid market. These applications sum up to thirty-nine applications. This is a significantly smaller number than the total number of applications from the other markets.

5.4.2.3 Distribution of applications over the clusters

Figure 5.4 illustrates the distribution of the applications into different clusters after the k-means algorithm is applied to the data set. The points on the y-axis indicates the number of applications, and the x-axis is composed of the sixteen clusters. Some of the clusters clearly sticks out, in both ends of the scale. The graph gives an understanding of the composition of the clusters. Some questions present themselves: Why does cluster 9, 13 and 15 contain significantly fewer applications than the other clusters? Does a tall column suggest that a cluster is too general, thus being made up of permission patterns with great variance between patterns? By looking at the within-cluster sum of squares, we can determine the density of each cluster. Larger with-cluster sum of squares means a higher probability of finding applications with contrasting permission patterns within the cluster. A cluster with high density (low SSE) means a higher probability of finding applications with similar, and even close to identical, permission patterns.

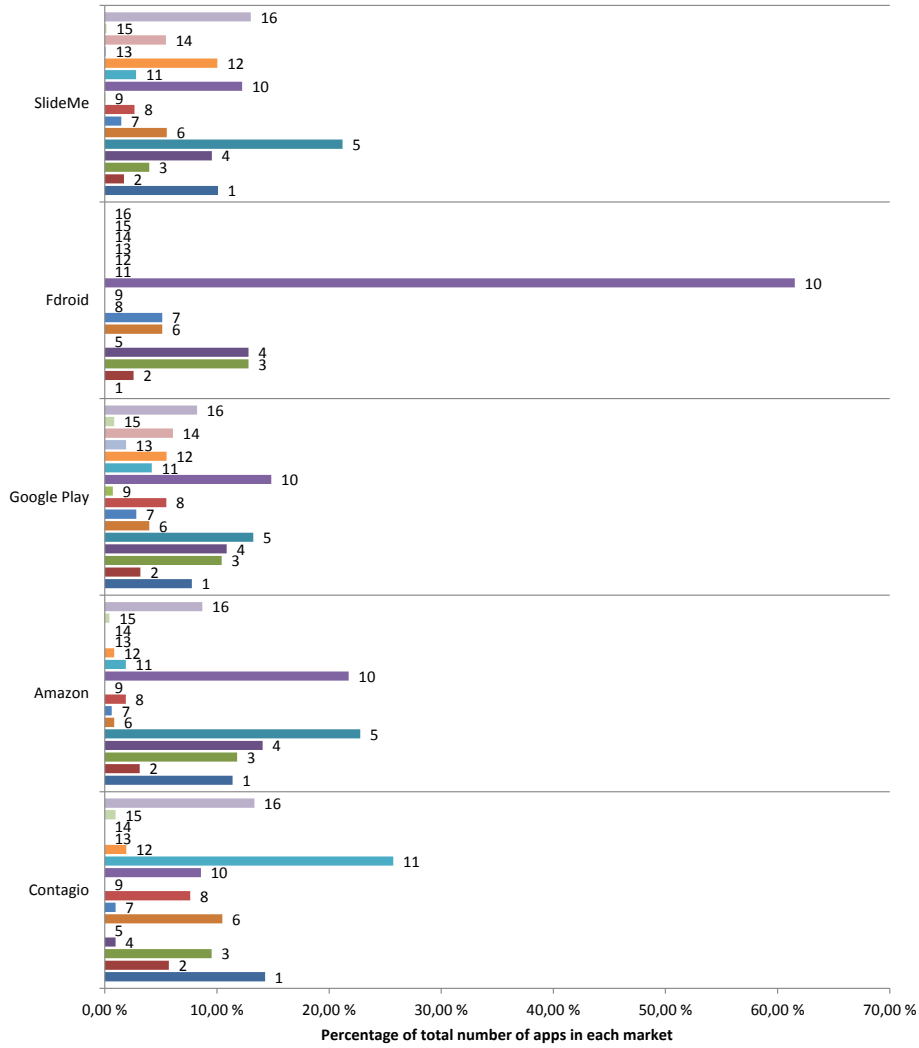

Figure 5.3: Results from running k-means with k=16. Shows the distribution of clusters for each market in percentage. Malware is represented by 'Contagio'.

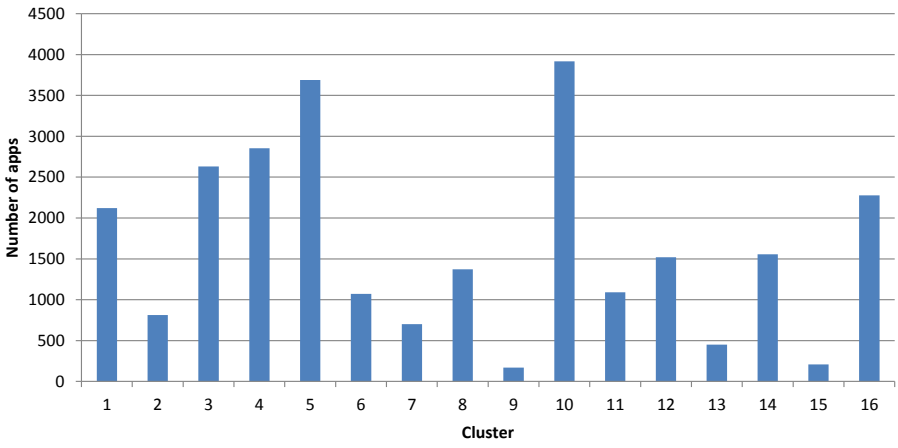

Figure 5.4: Distribution of applications in the sixteen clusters

5.4.2.4 Density of the clusters

Studying the density of each cluster might get additional insight in how the clusters are composed. As mentioned, a higher density (low within-cluster SSE) might imply that the cluster includes applications with close to identical permission patterns. Figure 5.5 shows a combined plot of the cluster size (in number of applications) and the within-cluster SSE. In the case of high density clusters, three clusters make an impression; cluster 9, 13 and 15. These clusters also include a significantly smaller number of applications than the average cluster. The combination of a small number of applications and a low within-cluster SSE points towards similar permission patterns in each of the clusters.

When it comes to clusters containing a large number of applications, cluster 5 and 10 are notable examples. They are comparatively similar in cluster size, and also show a below-average within-cluster SSE. These clusters are not as dense as the previously mentioned clusters, but because they are the largest clusters in size, it is assumed that these clusters represent more common permission patterns.

5.4.2.5 A closer look at interesting clusters

This section includes a closer examination of four clusters. The examination of the three last clusters include some figures showing the permission patterns of each cluster. The figures are intended as visual representations of the patterns in

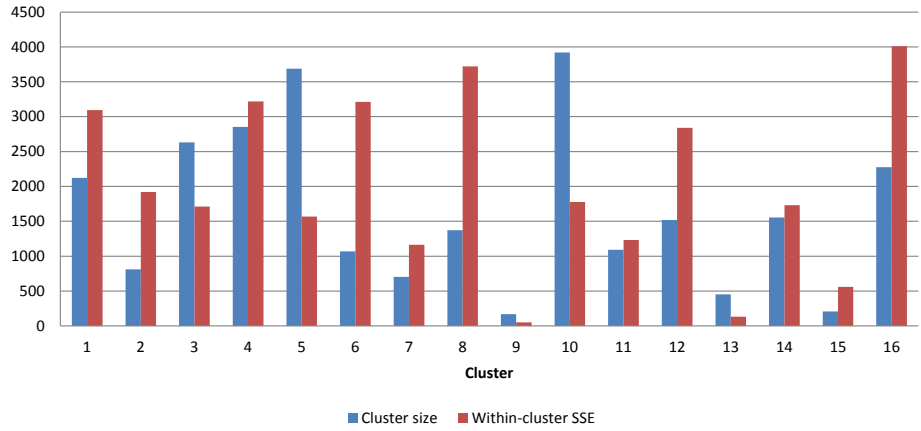

Figure 5.5: Comparison between number of applications and within-cluster sum of squares per cluster.

each cluster, and not for identifying specific permissions.

Cluster 10. The characteristic of cluster 10 is that the most popular permissions in the data set are under-represented. This is interesting due to the fact that cluster 10 is the largest cluster, containing 3,918 applications. Table 5.15 lists the most frequently requested permissions in the data set, together with a summary of the same permissions in cluster 10.

Cluster 10 equals to fifteen percent of the total number of applications in the data set. An interesting feature of this cluster is that none of the applications requests INTERNET and ACCESS_NETWORK_STATE. The INTERNET permission is present in the permission requests of eighty-four percent of the data set's applications. The ACCESS_NETWORK_STATE permission is present in fifty-four percent of the data set's applications.

Add up the applications in cluster 10 and the applications not requesting the INTERNET permission and you get close to the total amount of applications in the data set. This means that close to one-hundred percent of the applications requesting the INTERNET permission is assigned to cluster 10.

ACCESS_NETWORK_STATE enables the application to obtain information about the network interface of the Android phone. This permission is closely related to the INTERNET permission, seeing how ACCESS_NETWORK_STATE is prac-

tically always followed by INTERNET. ACCESS_NETWORK_STATE is requested 14,954 times in the data set, while 14,846 applications request both ACCESS_NETWORK_STATE and INTERNET. Out of 14,954 requests, merely 108 applications do not request INTERNET in addition.

This signals an unnecessary separation of two permissions. ACCESS_NETWORK_STATE is requested in order for the application to get information on the network connectivity. This feature implies that a network connection follows. The fact that only an insignificant number of applications in the data set request ACCESS_NETWORK_STATE without requesting INTERNET, should argue for the case that these be conjoined into one.

The reason that these two permissions are the most common in the data set is due to the fact that they are closely correlated. Practically every application that request ACCESS_NETWORK_STATE also request INTERNET. Still, the relationship is not one-to-one. INTERNET is still requested by about 7,000 more applications than ACCESS_NETWORK_STATE.

The two permissions are also present in a large part of the advertisement network libraries on the Android platform, and from the analysis in Section 5.3.1, INTERNET is required by all of the ad-network libraries. We can therefore say with certain confidence that cluster 10 is composed of applications that do not include ad-network libraries in their code. Consequently, the applications will be less likely to make use of personal information about either the user or the mobile device.

We do not imply that the applications in cluster 10 are excluded from this type of behaviour. However, we can be certain that ad-network libraries are not the reason for this type of behaviour. Although care should be taken when making the following argument, this may imply that applications in cluster 10 are less likely to "spy" on the user. We cannot conclude with the assumption of having found applications that are entirely free of spy-ware. A presentation at the Black-Hat conference in 2010 [51] describes how spying on a user can be accomplished without requesting any permissions.

WRITE_USER_DICTIONARY is requested by thirteen applications in the whole data set, and all occurrences are in cluster 10. It is not present in the documentation provided by Google [40], but the permission has been found on Google

Permission	# in data set	% in data set	# in c10	% in c10
INTERNET	22,230	84.08	0	0
ACCESS_NETWORK_STATE	14,954	56.56	0	0
WRITE_EXTERNAL_STORAGE	11,782	44.56	470	12
READ_PHONE_STATE	11,677	44.17	61	1.56
VIBRATE	6,478	24.50	248	6.33
ACCESS_FINE_LOCATION	6,038	22.84	20	0.51
WAKE_LOCK	5,445	20.6	224	5.72
RECEIVE_BOOT_COMPLETED	4,987	18.86	37	0.94
ACCESS_COARSE_LOCATION	3,126	11.82	9	0.23
ACCESS_LOCATION_EXTRA_COMMANDS	2,884	10.91	0	0
ACCESS_WIFI_STATE	2,681	10.14	20	0.51

Table 5.15: Comparison of top permissions in the total data set, including malicious applications, and cluster 10

Permission	# in cluster 10
WRITE_EXTERNAL_STORAGE	470
VIBRATE	248
WAKE_LOCK	224
SET_WALLPAPER	131
CAMERA	66
READ_PHONE_STATE	61
READ_CONTACTS	48
WRITE_SETTINGS	47
BLUETOOTH	42
CALL_PHONE	38

Table 5.16: Top 10 permissions of cluster 10

Play. As mentioned in Section 4.2.2, the permission was included in our analysis because it is accepted and presented as a valid permission when uploading an application to Google Play. The tenth most requested permission in the data set, ACCESS_LOCATION_EXTRA_COMMANDS, is not present in cluster 10. The permission is requested by ten percent of the applications in the data set.

While this is the largest cluster in number of applications, no clear permission pattern can be read from the applications assigned to the cluster. The within-cluster SSE of cluster 10 is below the average for all the clusters combined, but the distance between the applications within the cluster is still to significant to be able to represent one distinct pattern. The top 10 permissions in cluster 10 are listed in Table 5.16. The number of occurrences of each of the top permissions is too low and too unreliable to be able to identify a distinct pattern. As some of the subsequent sections will illustrate, patterns *can* be identified by using this approach although not in the case of this cluster.

Cluster 9. Due to the combination of small size and high density, cluster 9 indicates a potentially interesting grouping of applications. Given how the k-means algorithm works, a distinct permission pattern should present itself due to the low within-cluster SSE. The probability of this result appearing is increased because of the size of the cluster. 170 applications are assigned to this cluster, a mere 0.64 percent of the complete data set.

The most prominent characteristic of this cluster is the distinct set of permissions that form a clear and consistent pattern across nearly all of the applications in

Permission	# in cluster 9
VIBRATE	170
WRITE_EXTERNAL_STORAGE	170
INTERNET	169
KILL_BACKGROUND_PROCESSES	169
ACCESS_NETWORK_STATE	168
GET_ACCOUNTS	168
READ_CONTACTS	166
SYSTEM_ALERT_WINDOW	165

Table 5.17: Top 8 permissions of cluster 9

cluster 9. The top eight permissions in this cluster are listed in Table 5.17. Most of the permissions listed are common in the complete data set. Two of the permissions are not frequently requested. KILL_BACKGROUND_PROCESSES gives the application the permission to call killBackgroundProcesses(String packageName), killing all background processes associated with a given package. This is equivalent to the action the kernel performs in order to kill processes to reclaim memory. SYSTEM_ALERT_WINDOW allows the application to create alert windows even though the application is not in focus. This means that an application with this permission may create pop-up windows with advertisements while the user is browsing the web or writing a text message.

In the complete data set, KILL_BACKGROUND_PROCESSES is requested 491 times, while SYSTEM_ALERT_WINDOW is requested 258 times. The permission pattern observed in this cluster represents a large share of these requests, which is worth noting. The requests may still be justifiable. To verify, a further study of the applications in this cluster is necessary. To illustrate the obvious pattern shown in cluster 9, Figure 5.6 depicts a table with rows representing the applications in cluster 9, and the columns representing each permission. Black denotes permissions present in the application's request. Looking at the pattern, there can be no doubt that the clustering has resulted in identifying a distinct set of permissions, a pattern describing a certain type of applications. Since the number of applications in the cluster is not more than 170, Figure 5.6 can also be examined in order to identify applications that are not fitting one-hundred percent within the pattern. A short glimpse on the permission pattern of cluster 9 distinguishes seven applications that deviate a bit from the general pattern. The permission patterns of these applications are still evaluated as close enough for the clustering algorithm to organize these applications together with the rest

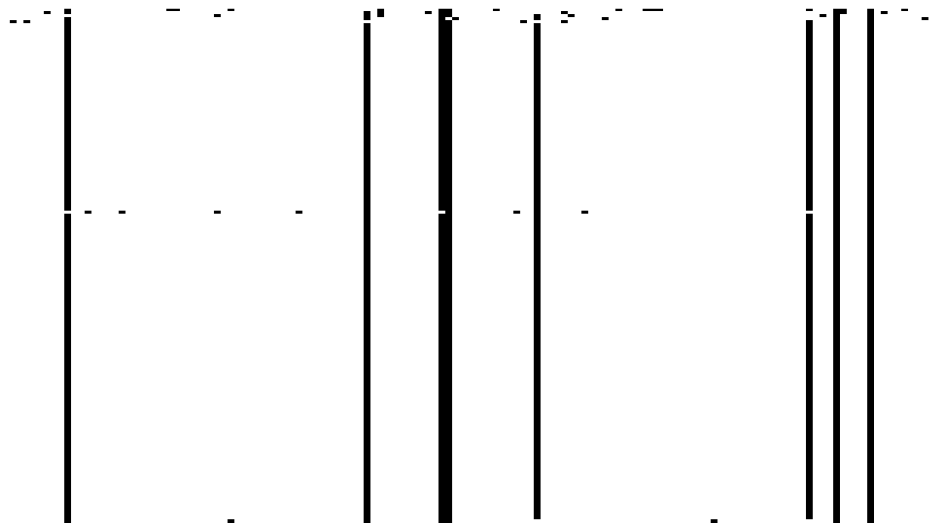

Figure 5.6: Permission patterns of the 170 applications assigned to cluster 9. The figure depicts a table where each column represents a permission, and each row represents an application in cluster 9. Black means that the permission is present in the permission requests of an application.

in cluster 9.

One element of interest is the distribution of different categories within the cluster. If a pattern or a similarity can be found in the distribution of categories, this can lead to interesting findings. Table 5.18 lists the categories found in cluster 9 in order of occurrences. Aside from Productivity, Communication, Education and Entertainment, the categories represent different types of games.

Seeing as the permissions in this cluster form a distinct pattern, and the ap-

Category	**Occurrence in cluster 9**
Arcade & Action	108
Casual	38
Brain & Puzzle	16
Productivity	3
Cards & Casino	2
Communication	1
Education	1
Entertainment	1

Table 5.18: Categories found in cluster 9

Developer	**# of applications in cluster 9**
Thio	60
FREEJOYO.COM	41
tinepeng215	41
FREEstudio	21
appbody	1
Blue Coat Systems	1
DuzonBizon	1
EmailTray	1
ETWAP	1
Mint Shirt Productions	1
TopWare Interactive AG	1

Table 5.19: Developers in cluster 9

plications are mainly games, a closer look at the developers may uncover more information on the applications assigned to this cluster. Table 5.19 lists the developer IDs from cluster 9. The Table shows that four developers are dominating this cluster. *FREEJOYO.COM* and *FREEstudio* both list the same developer URL. This gives reason to believe that these two developers actually are one developer with two different developer IDs.

At the time of writing, the applications from developers *FREEJOYO.COM*, *tinapeng215* and *FREEstudio* are all taken down from the market. A curious observation is that even though the permission requests from all these developers' applications are identical, the applications from *Thio* still exist on Google Play. While it has not been possible to provide definite answers as to why this is the case, the applications developed by *Thio* may be reviewed in order to give clues to why this has happened.

Apart from the dominating developers, and the distinct permission pattern of this cluster, the application IDs are also interesting. It turns out that of the 170 applications in this cluster, all of the applications developed by the top four developers share a similar feature. The application IDs all include one of the names in Table 5.20. The names appear in the following format: com.droidhen.swf[...], com.glu.swf[...], com.gameloft.swf[...]. The application IDs for these applications are all in this format, with the square brackets representing a string of random numbers making every application ID unique.

Six of these application IDs have been connected to companies who develop

application ID	Confirmed Android games developer
com.droidhen.swf[...]	X
com.ninja.swf[...]	
com.finditmario.swf[...]	
com.glu.swf[...]	X
com.ea.swf[...]	X
com.gameloft.swf[...]	X
com.rovio.swf[...]	X
com.roidgame.swf[...]	
se.illusionlabs.swf[...]	X

Table 5.20: Names from applications IDs of applications developed by the top four developers in cluster 9. The *X* marks that the name has been confirmed as a developer of Android games.

Android games. Combine this with the fact that the permission patterns are identical, and it is tempting to conclude that these applications are posing as legitimate applications in order to lure users into installing them.

Cluster 15. The second smallest cluster in size is cluster 15, containing 207 applications. The eleven most frequently requested permissions are listed in Table 5.21. The number of occurrences of each permission is not as consistent as in the case of cluster 9. Nevertheless, a pattern can still be observed. No clear result could be read from looking at the categories of cluster 15. When studying the overview of the developers present in this cluster, just one developer ID appeared noticeably more frequent than the others. The developer with ID *Balkanboy Media* has developed twenty-two applications, with twenty of them appearing in cluster 15. Such a high share of the developer's applications in one cluster is curious.

Balkanboy Media's applications have been removed from Google Play since the time when the application data was collected, though some of the applications still exist on the AppBrain market. Nineteen of the twenty applications in cluster 15 is in the News & Magazines category. This could justify the identical permission pattern, since the applications may just be the same application ported for different usage scenarios. It may also imply that the developer is spamming the market.

Permission	**# in cluster 15**
INTERNET	205
WRITE_EXTERNAL_STORAGE	205
ACCESS_FINE_LOCATION	204
ACCESS_NETWORK_STATE	199
ACCESS_LOCATION_EXTRA_COMMANDS	194
WRITE_CONTACTS	187
RECEIVE_SMS	175
MODIFY_AUDIO_SETTINGS	174
READ_PHONE_STATE	171
VIBRATE	163
GET_ACCOUNTS	138

Table 5.21: Top 11 permissions of cluster 15

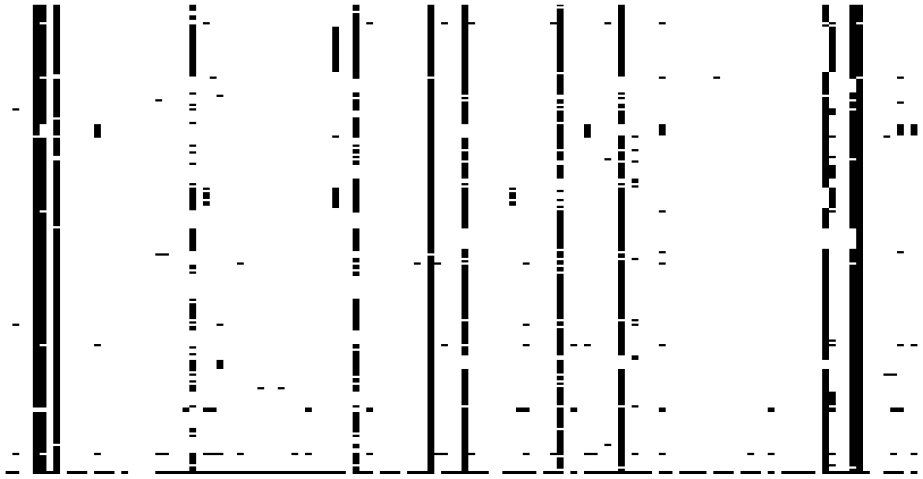

Figure 5.7: Permission patterns of the 207 applications assigned to cluster 15. The figure depicts a table where each column represents a permission, and each row represents an application in cluster 15. Black means that the permission is present in the permission requests of an application.

Figure 5.7 shows the permission pattern in cluster 15. While a pattern can be derived from this plot, it does not show the consistency as in the case of cluster 9. This observation is supported by the within-cluster SEE, compared to the same value for cluster 9. The within-cluster SSE is eleven times larger than the within-cluster SSE for cluster 9.

Permission	# in cluster 13
ACCESS_MOCK_LOCATION	451
ACCESS_NETWORK_STATE	451
CALL_PHONE	451
GET_ACCOUNTS	451
INTERNET	451
WRITE_EXTERNAL_STORAGE	451
WAKE_LOCK	450
WRITE_CONTACTS	446
CAMERA	434
READ_PHONE_STATE	434
DELETE_PACKAGES	340

Table 5.22: Top 11 permissions of cluster 13

Cluster 13. The third smallest cluster in size is cluster 13, containing 451 applications. The within-cluster SSE is the second smallest, and looking at Figure 5.5 suggest that this cluster should contain applications showing a similarity in permission requests when compared to each other. Table 5.22 lists the eleven most frequently requested permissions in this cluster. Among commonly requested permissions, the table shows permissions that are not common for the rest of the data set. ACCESS_MOCK_LOCATION is requested in total 606 times, with seventy-four percent of the requests coming from applications in cluster 13. WRITE_CONTACTS is requested in total 969 times, with forty-six percent of the requests coming from applications in cluster 13. DELETE_PACKAGES is requested in total 359 times, with ninety-five percent of the requests coming from applications in cluster 13.

Since DELETE_PACKAGES is a *signatureOrSystem* permission, applications will not be granted this permission if they are not signed by either the device manufacturer or the signed with the same certificate used to sign the version of the Android system installed on the device, as covered in Section 5.1.2. While the application will not have the permission to delete packages, it should still be mentioned because of the severe damage this could do to a device. The fact that this permission is even requested should be alarming.

Cluster 13 is not dominated by only a few developers, unlike cluster 9, but there are still a few developers that stick out from the crowd. *Digital Online Media* has developed 38 of the applications in cluster 13, the highest amount of applications in this cluster. A search on AppBrain and Google Play reveals that the developer

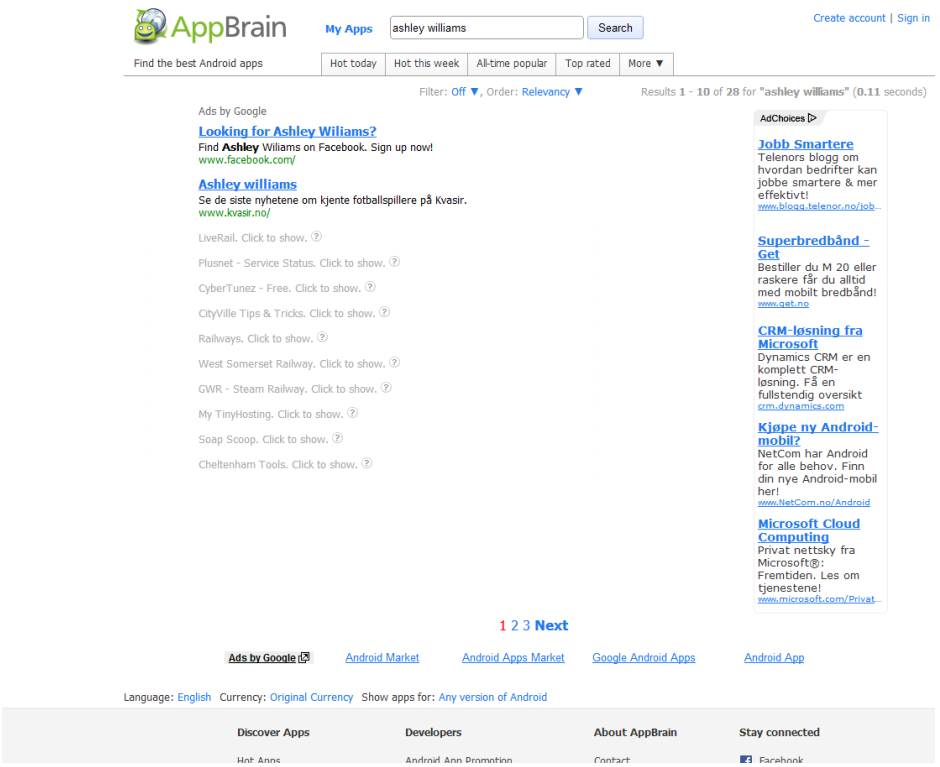

Figure 5.8: Searching for developer *Ashley Williams* on AppBrain. The applications are still visible, but marked as spam. (June 7th, 2012)

is no longer listed on the markets. *Ashley Williams* has developed 10 of the applications in cluster 13. This developer has been classified as a spammer on the AppBrain market. Developers with a combination of low popularity and poor ratings on AppBrain may be classified as such [6]. The applications are still listed on AppBrain, but the user must click to reveal the application because of the application having been classified as spam. Figure 5.8 shows the search results for *Ashley Williams* on AppBrain.

Figure 5.9 shows the permission pattern of cluster 13. As expected, the patterns are highly visible, because of the low within-cluster SSE of this cluster.

5.4.2.6 Occurrences of malware in the clusters

Some of the malicious applications studied in this project show very distinct permission patterns. The Geinimi Trojan has a very distinct set of permission

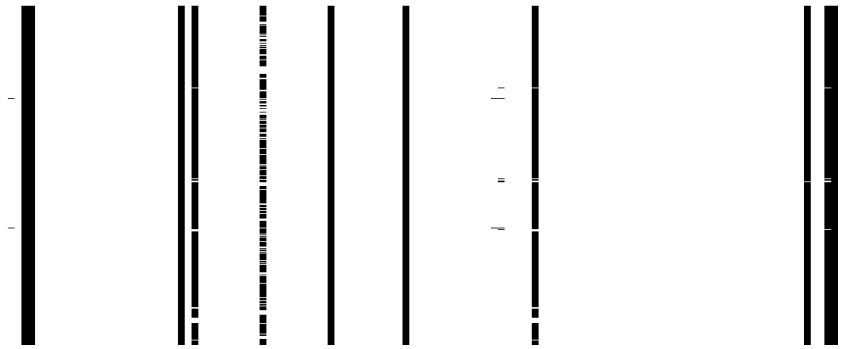

Figure 5.9: Permission patterns of the 451 applications assigned to cluster 13. The figure depicts a table where each column represents a permission, and each row represents an application in cluster 13. Black means that the permission is present in the permission requests of an application.

requests, although most of the permissions requested by the infected applications are commonly requested in the market data set. The Geinimi Trojan will be used as an example to find out if the clustering results can identify malware present in the clusters by looking at the clusters the malicious applications are assigned to.

Identifying malware in clusters. The Geinimi trojan's permission requests are shown in Table 5.5. As all the samples of Geinimi are quite consistent in their permission requests, the clustering task is expected to assign most of the Geinimi applications to the same cluster. K-means groups the applications based on their vector's distance to the rest of the applications. As discussed earlier, the more similar the applications are to each other in form of permission requests, the closer their respective vectors will be to each other.

The results of the clustering confirms the expectations regarding whether or not the Geinimi applications will be assigned mainly to the same cluster. Of twenty-four Geinimi samples, nineteen appears in cluster 11, two in cluster 6 and three in cluster 2.

This should not be interpreted as cluster 11 containing only Geinimi malware, despite the fact that nearly all of the Geinimi samples were assigned to this cluster. Concluding like this would classify over 1,000 market applications as being infected with the Geinimi Trojan.

This also goes for the other malware samples in the data set. Even though a

malicious application appears in the same cluster as other applications, this does not imply that all the applications in that cluster contains the same Trojan, or behaves in the same malicious manner.

If the set of malicious applications was substantially larger, such an approach might have given more significant results. The fact that the malicious applications used in this analysis are so few means that the clustering algorithm can not be used to obtain such results.

5.5 Summary of Findings

A summary of the results from the analysis will be presented in the subsequent sections.

5.5.1 Pattern analysis

The permissions requested by an application can be used to determine whether or not it is suspicious. This is because, as demonstrated in Section 5.4.1, even if the permissions requested by a malicious application are themselves rather common permissions, the combinations requested by malicious applications are atypical. The method is however not perfect, as there is a chance of false positives which varies with the uniqeness of the permission pattern. The more unique the pattern, the fewer applications are caught by the pattern both from the malicious data set and the market data set. This false positive range from the GoldDream pattern, which has a false positive rate of zero, to the DroidDream pattern with a 0.75 percent false positive rate. The CounterClank pattern did have a false positive rate of 30.9 percent, but as covered in Section 5.2.1, this pattern can be discounted due to not being a malicious application.

5.5.2 Clustering analysis

The objective of performing the clustering analysis on the data set was to evaluate the potential of applying machine learning methods on the data set, by looking at the results from the clustering. From the results presented in Section 5.4.2.5, several interesting features of the data set was identified. Groups of applications deviating from the most common permission requests were identified. In cluster 9, 170 applications potentially spamming the market were identified. Some of these applications are still listed on Google Play, while applications that were identical

in permission patterns and application ID have been taken down since the time when the data set was built. The clustering analysis also revealed about 450 applications with close to identical permissions patterns in cluster 13. Because of the identical permission patterns of these applications, combined with the fact that some of the applications have already been classified as spam by AppBrain, they show clear signs of low quality applications possibly spamming the market.

The results obtained from the clustering analysis would most likely not be obtained by manual inspection of the data set. K-means automatically provides us with a separation of the data set based on the similarity between each of the applications. This leads to patterns being identified by the clustering algorithm, providing us with a solid basis for further examination of the data set.

5.5.3 Comparing the analysis methods

The seven Trojans analysed in Section 5.2 were compared to the clustering results. The objective was to identify whether or not the clustering algorithm ended up grouping the Trojan samples in the same clusters. The results shows that this was a fact for the Trojans with several samples in the data set, such as Geinimi and Pjapps.B. Seven of ten Pjapps.B Trojans were assigned to cluster 16, and nineteen of twenty-four Geinimi Trojans were assigned to cluster 11. The Trojans with few samples in the data set did not display this tendency.

The reason for samples of the same malware not being assigned to the same cluster, can be attributed to two issues. First, the malicious data set consists of too few infected applications to be able to discern a pattern. Second, even if two applications are infected with the same Trojan, this does not mean that the permission requests will be identical, as demonstrated in Section 5.2. If two samples of the same Trojan show slightly different permission patterns, the k-means algorithm will not necessarily assign the two samples to the same cluster.

Chapter 6

Discussion

This chapter discusses the implications and potential limitations of the results and methods presented in this thesis.

6.1 Implications

The subsequent sections will present the implications the results presented in this thesis has for the Android community, and the measures that should be taken by Google and Android developers.

6.1.1 Signature and signatureOrSystem permissions and Google Play

During the project we uncovered some strange behaviour in the Google Play market.

When dealing with the *signature* and *signatureOrSystem* protection levels, both the Android platform and the Google Play market displays some very incongruous behaviour. This behaviour is displayed during both the presentation of the application on the market as well as during application installation. When requesting that the user grant the application permissions with these two protection levels, the permission requests are hidden from the user on the installation screen among the safe permissions. This means that the user is required to click *show*

all to display them.

It is assumed that this is because these permissions would under normal circumstances have no effect due to the protection level. Even so, the fact that it is even asking for such a permission should be considered a warning sign to the user.

6.1.2 Lack of sufficient documentation of the permissions

During the course of this project we were required to identify both how the security features of the Android platform interacts, as well as what the capabilities of the identified permissions are. We were surprised to discover that while the Android security architecture is rather well documented, the permissions are almost completely undocumented beyond the information available in the descriptions presented by Google Play. This is worrying, as this is the only feature that is presented to the user during normal operation.

To identify the permissions used in the analysis in this thesis, we were required to go through the process presented in Section 4.2.2. While most of the permissions are described in Google Play, once you know which permission you are looking for, the capabilities of the permissions are not adequately described. An example of this is the popular READ_PHONE_STATE permission, which is presented in Google Play [39] with the following description: «Allows to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and the like». As covered in Section 5.1.2, this permission also provides access to information like the IMSI and the IMEI numbers. This is information that is not included in the description, and as such the permission is presented as less invasive than it actually is.

6.1.3 Application builders used to spread malware

As mentioned in Section 5.3.2, during the analysis of the permissions we discovered a large sample of applications with similar application identifiers and the same set of permissions, but different developers.

It strikes us that a free application builder could be leveraged as a malware delivery vector. An attacker could create his own application builder, which would create applications for other developers, bundled with the attackers own malicious

code. The attacker could then have a large number of developers spreading his infected applications with very little personal intervention.

Because the developers are giving up control of the actual coding of the applications, there is little chance that the developers using the application builder is going to discover that their applications are infected before they get taken down. At which point the attacker will already have moved on to a new set of developers.

6.1.4 Value of pattern-based recognition

As the malware used here to create the patterns used for this method are "old", it should be made clear that these patterns are not intended to be used for identifying these specific malicious applications, but to demonstrate whether or not a system based on this method would work. As mentioned earlier, for the purpose of this discussion we will refer to any applications from the legitimate data set that are identified as a malicious application by the patterns as a false positive.

From the tests done in Chapter 5, we would argue that this is a valid method. The patterns did in general catch more of the malicious applications than just the specific malware they were based on, and had a relatively low false positive rate. The highest false positive of the patterns, not counting the CounterClank pattern, were 199 applications. This represents 0.75 percent of our legitimate data set, and applying this pattern to the entire Google Play market, which at the point of writing consists of roughly 444,000 [7] applications, would result in a list of roughly 3,330 suspicious applications.

The failure of the CounterClank pattern does however showcase the weakness of the method. The permissions that each CounterClank sample had in common were not distinct enough to accurately separate the CounterClank applications from the legitimate applications. As mentioned in Section 5.2, CounterClank is not necessarily a malicious application but an aggressive advertisement network. Comparing the CounterClank pattern to the requested permissions from the advertisement networks in Section 5.3.1, we see that the permissions requested by CounterClank are all permissions that are required by the AirPush network which, as mentioned in the same chapter, include functionalities similar to those of CounterClank.

While not able to distinctly identify malicious applications, this method can be used to flag suspicious applications for further review.

6.2 Potential Limitations

The subsequent sections will discuss the methods used during the analysis of the Android data set. The limitations of each method are presented, and arguments are made as to whether or not the methods produced valuable results.

6.2.1 Determining number of clusters for k-means

Since the elbow method did not give a clear image on which value of k to use, we had to make our own assessments on how to best determine the number of clusters to use in the k-means clustering. By studying the difference of the cost function from each value of k to the next, we looked for the point where the cost function started decreasing at a steady rate. The number of clusters were chosen after visual analysis of the graph showing the decrease for each value of k. Even though k=16 was the value used in the analysis, the graph does not show a clear point where the cost function starts to steadily decrease. This means that the number chosen is not a definitive value, and may be prone to errors. A choice was made based on the graphical representation of the decrease of the cost function, but other values of k might have given a different result.

A different number of clusters might have resulted in additional patterns being observed during the analysis. We must therefore be aware that while results were obtained from using sixteen clusters in the clustering, there is a chance that there may be additional findings in this data set.

6.2.2 On using k-means clustering for analysing Android applications

Based on our analysis, the process of identifying malware through the use of clustering algorithms is not likely to produce any valuable results. At least not when only taking the permissions requested by an application into consideration. Because some permissions give access to a larger range of capabilities than others, malware assigned to a cluster does not give any indication as to whether or not this is a cluster containing a lot of malware. As shown in the analysis of the clustering results of the Geinimi Trojan, it is clear that too much emphasis must

not be put on the reason for malware being assigned to certain clusters.

The objective of applying the k-means clustering algorithm to the data set, was to explore the possibilities of finding patterns and other interesting features in the collection of Android applications. The results clearly shows that the clustering task achieved this. Several distinct permission patterns were identified, uncovering developers spamming the market and suspicious application identifiers. We are confident that this type of analysis of Android applications can uncover properties within the data set that would not be found without performing the clustering. The clustering task is in itself not a time-consuming task, which means it can be applied to larger data sets than the one used in this project. It is hoped that this analysis will encourage further explorations in the field of machine learning in relation to investigating large sets of Android applications.

Chapter 7

Ideas for Future Work

We have considered some ways the project can be extended, and how the data set can be used for further research.

7.1 Is the Application Suspicious?

We propose a web based solution where both users and developers can check whether or not a combination of permissions can be considered to be a suspicious combination, with advice related to what capabilities the application would have, and the privacy and monetary risks these permissions would entail. While two permissions can be harmless when granted separately, the privacy and security risk can increase drastically if they are granted together. As an example, take the INTERNET and READ_SMS permissions; separately they are harmless, but combined with each other you get an application that can read your text messages and send the contents to a third party. There are many permission combinations that carry severe risks but are not presented as such to the user. A study can be performed considering these risks, using the patterns presented in this thesis as a starting point.

This would help in educating both users and developers as to what exactly the applications are requesting access to.

7.2 Including Third-party Permissions

As covered in Section 4.2.2, we did not put much emphasis on third-party permissions in the analysis. However, as the third-party permissions we collected are still included in the data set, a possible extension to this project is to use the same methods we have used for our analysis on the third-party permissions.

Note that as Google Play does not list any third-party permissions from applications not created by Google, the Google Play data set might not be useful for any analysis which includes other permissions. As Google Play has provided us with the largest number of applications, this would invalidate a large part of our data set.

7.3 Exploring Other Machine Learning Methods

While our machine learning analysis has focused on k-means clustering, other machine learning methods like neural networks can be applied to the problem as well. If the set of malicious applications had been sufficiently large, the applications can be used as training examples in a neural network approach. This is an example of supervised learning, as described in Section 3.5.1. If the malicious data set consisted of several hundreds, preferably thousands, of applications, a neural network could be constructed in order to classify new and unknown applications as malicious or legit. A larger set of possible outcomes should be used in order to represent a data set with such a high dimensionality.

Chapter 8

Conclusion

During the course of this project we created several data collection programs and the infrastructure required for building a data set of 26,333 applications from four Android application markets. During the analysis of this data set, we uncovered incongruous behaviour in the Google Play market regarding permissions with high protection levels. Additionally, we were able to identify applications built by third-party application builders.

Manual pattern recognition has proven to be effective in selecting applications for closer scrutiny. Patterns in the permission requests of malicious applications were identified, and can be applied to the complete data set in order to single out applications matching the identified patterns. This can not be considered a reliable way of uniquely identifying malicious applications in the complete data set, but applications matching the malicious patterns can be singled out for review.

K-means clustering has been successful in uncovering suspicious permission patterns in the complete data set, by grouping applications with similar permission requests together in the same clusters. The clusters are valuable starting points for closer examination of the data set. While k-means is not suitable for identifying malicious applications based on their permission requests, it has shown its value in uncovering low quality applications potentially spamming the market, as well as specific developers publishing such applications.

Both of these methods are valuable additions to the application review process,

and should be considered as improvements to the review process currently being used to prevent malicious and low quality applications from being accepted to the markets.

Bibliography

[1] AdWhirl. Android sdk setup. Available: https://www.adwhirl.com/doc/android/AdWhirlAndroidSDKSetup.html. Accessed 20.05.12.

[2] AirPush. Developers. Available: http://www.airpush.com/developers. Accessed 20.05.12.

[3] Amazon. Amazon appstore for android. Available: http://www.amazon.com/mobile-apps/b/ref=topnav_storetab_mas?ie=UTF8&node=2350149011. Accessed 25.01.12.

[4] Amazon. Amazon elastic compute cloud (amazon ec2). Available: http://aws.amazon.com/ec2/. Accessed 20.02.12.

[5] AppBrain. Appbrain. Available: http://www.appbrain.com/. Accessed 20.02.12.

[6] AppBrain. Appbrain launches automatic filtering of "spam" apps. Available: http://blog.appbrain.com/2010/03/appbrain-launches-automatic-filtering.html. Accessed 07.06.12.

[7] AppBrain. Number of available android applications. Available: http://www.appbrain.com/stats/number-of-android-apps. Accessed 25.05.2012.

[8] Mario Ballano. Android threats getting steamy. Available: http://www.symantec.com/connect/blogs/android-threats-getting-steamy. Accessed 10.05.12.

[9] David Barrera, H GÃŒnes Kayacik, Paul C van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based

security models and its application to android. Available: http://people.scs.carleton.ca/~paulv/papers/ccs10-som-android.pdf. Accessed 14.05.12.

[10] Paul Brodeur. Zero-permission android applications. Available: http://leviathansecurity.com/blog/archives/17-Zero-Permission-Android-Applications.html. Accessed 22.05.12.

[11] Brut.alll. android-apktool. Available: http://code.google.com/p/android-apktool. Accessed 01.03.12.

[12] Ed Burnette. What's new in android 1.6 (donut)? part 2: Developer features. Available: http://www.zdnet.com/blog/burnette/whats-new-in-android-16-donut-part-2-developer-features/1369?pg=3. Accessed 03.06.12.

[13] Rich Cannings. Exercising our remote application removal feature. Available: http://android-developers.blogspot.com/2010/06/exercising-our-remote-application.html. Accessed 22.03.12.

[14] Rich Cannings. An update on android market security. Available: http://googlemobile.blogspot.com/2011/03/update-on-android-market-security.html. Accessed 22.03.12.

[15] CarrierIQ. Know your customer experience. Available: http://www.carrieriq.com/know-your-customer-experience/. Accessed 02.04.12.

[16] ccjernigan. Security flaw in power control widget opens protected settings. Available: http://code.google.com/p/android/issues/detail?id=7890. Accessed 22.05.12.

[17] Pern Hui Chia, Yusuke Yamamoto, and N Asokan. Is this app safe? a large scale study on application permissions and risk signals. Available: http://q2s.ntnu.no/~chia/paper/app-www.pdf. Accessed 24.01.12.

[18] Graham Cluley. Android malware poses as angry birds space game. Available: http://nakedsecurity.sophos.com/2012/04/12/android-malware-angry-birds-space-game/. Accessed 01.04.12.

[19] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. Available: http://www.patrickmcdaniel.org/pubs/ccs09a.pdf. Accessed 29.02.12.

[20] F-Droid. F-droid. Available: `http://fdroid.org/`. Accessed 27.01.12.

[21] F-Secure. New pjapps variant. Available: `http://www.f-secure.com/weblog/archives/00002108.html`. Accessed 10.05.12.

[22] F-Secure. Threat description: Trojan-downloader:osx/flashback.c. Available: `http://www.f-secure.com/v-descs/trojan-downloader_osx_flashback_c.shtml`. Accessed 15.04.12.

[23] F-Secure. Threat description: Trojan-spy:w32/zbot. Available: `http://www.f-secure.com/v-descs/trojan-spy_w32_zbot.shtml`. Accessed 15.04.12.

[24] F-Secure. Threat description: Trojan:android/droidkungfu.c. Available: `http://www.f-secure.com/v-descs/trojan_android_droidkungfu_c.shtml`. Accessed 15.04.12.

[25] F-Secure. Threat description: Trojan:android/ginmaster.a. Available: `http://www.f-secure.com/v-descs/trojan_android_ginmaster_a.shtml`. Accessed 15.04.12.

[26] F-Secure. Threat description: Virus:boot/brain. Available: `http://www.f-secure.com/v-descs/brain.shtml`. Accessed 02.03.12.

[27] F-Secure. Virus and threat descriptions. Available: `http://www.f-secure.com/en/web/labs_global/threats/descriptions`. Accessed 01.03.12.

[28] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. Available: `http://www.cs.berkeley.edu/~afelt/android_permissions.pdf`. Accessed 30.02.12.

[29] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. A static analysis tool and permission map for identifying permission use in android applications. Available: `http://www.android-permissions.org/permissionmap.html`. Accessed 30.02.12.

[30] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steven Hanna, and David Wagner. A survey of mobile malware in the wild. Available: `http://www.cs.berkeley.edu/~afelt/mobilemalware.pdf`. Accessed 30.02.12.

[31] Adrienne Porter Felt and David Wagner. Phishing on mobile devices. In *Web 2.0 Security and Privacy*, 2011. Available: `http://w2spconf.com/2011/papers/felt-mobilephishing.pdf`. Accessed 02.05.12.

[32] R Project for Statistical Computing. K-means clustering. Available: http://stat.ethz.ch/R-manual/R-devel/library/stats/html/kmeans.html. Accessed 05.06.12.

[33] FortiNet. Android/adsms.a!tr. Available: http://www.fortiguard.com/av/VID2889424. Accessed 02.06.12.

[34] FortiNet. Android/geinimi.a!tr. Available: http://www.fortiguard.com/av/VID2374726. Accessed 13.03.12.

[35] Google. Android market developer program policies. Available: http://www.android.com/us/developer-content-policy.html. Accessed 22.03.12.

[36] Google. Android security overview. Available: http://source.android.com/tech/security/index.html. Accessed 04.03.12.

[37] Google. Developer distribution agreement. Available: http://www.android.com/us/developer-distribution-agreement.html. Accessed 22.03.12.

[38] Google. Google admob ads android fundamentals. Available: https://developers.google.com/mobile-ads-sdk/docs/android/fundamentals. Accessed 20.05.12.

[39] Google. Goolge play. Available: https://play.google.com/store. Accessed 25.01.12.

[40] Google. Manifest.permission. Available: http://developer.android.com/reference/android/Manifest.permission.html. Accessed 15.02.12.

[41] Google. <permission>. Available: http://developer.android.com/guide/topics/manifest/permission-element.html. Accessed 17.03.12.

[42] Google. Platform versions. Available: http://developer.android.com/resources/dashboard/platform-versions.html. Accessed 27.05.12.

[43] Google. Security and permissions. Available: http://developer.android.com/guide/topics/security/security.html. Accessed 17.03.12.

[44] Google. Signing your applications. Available: http://developer.android.com/guide/publishing/app-signing.html. Accessed 19.03.12.

[45] Michael Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. Systematic detection of capability leaks in stock android smartphones. Available: `http://www.csc.ncsu.edu/faculty/jiang/pubs/NDSS12_WOODPECKER.pdf`. Accessed 25.04.12.

[46] InMobi. Android. Available: `http://developer.inmobi.com/wiki/index.php?title=Android`. Accessed 20.05.12.

[47] Martin Jericho. Jericho html parser. Available: `http://jericho.htmlparser.net/docs/index.html`. Accessed 12.02.12.

[48] Trevor Johns. Android_id not random if ro.serialno isn't set. Available: `http://code.google.com/p/android/issues/detail?id=10639`. Accessed 03.06.12.

[49] David Kravets. Researcher's video shows secret software on millions of phones logging everything. Available: `http://www.wired.com/threatlevel/2011/11/secret-software-logging-video/`. Accessed 02.04.12.

[50] LeadBolt. What are the different android permissions needed for? Available: `http://qa.leadboltads.com/questions/24/what-are-the-different-android-permissions-needed-for`. Accessed 20.05.12.

[51] Anthony Lineberry, David Luke Richardson, and Tim Wyatt. These aren't the permissions you are looking for. In *BlackHat*, 2010. Available: `http://dtors.files.wordpress.com/2010/09/blackhat-2010-final.pdf`. Accessed 02.06.12.

[52] SlideME LLC. Slideme, your marketplace for android apps. Available: `http://slideme.org/`. Accessed 25.01.12.

[53] Hiroshi Lockheimer. Android and security. Available: `http://googlemobile.blogspot.com/2012/02/android-and-security.html`. Accessed 11.04.12.

[54] Lookout. Lookout's take on the 'apperhand' sdk (aka 'android.counterclank'). Available: `http://blog.mylookout.com/blog/2012/01/27/lookout%E2%80%99s-take-on-the-%E2%80%98apperhand%E2%80%99-sdk-aka-android-counterclank/`. Accessed 19.03.12.

[55] Lookout. Update: Security alert: Hacked websites serve suspicious android apps (notcompatible). Available: http://blog.mylookout.com/blog/2012/05/02/security-alert-hacked-websites-serve-suspicious-android-apps-noncompatible/. Accessed 10.05.12.

[56] McAfee. Virus profile: Android/pjapps.c. Available: `http://home.mcafee.com/virusinfo/virusprofile.aspx?key=580250`. Accessed 10.05.12.

[57] Millennial Media. Android. Available: `Androidhttp://wiki.millennialmedia.com/index.php/Android`. Accessed 20.05.12.

[58] Trend Micro. Trend micro develops advanced cloud-based mobile application scanning technology – with capacity to process 5,000+ apps daily. Available: `http://newsroom.trendmicro.com/index.php?s=43&news_item=948&type=archived&year=2012`. Accessed 02.03.12.

[59] Mila. Contagio mobile. Available: `http://contagiominidump.blogspot.com/`. Accessed 15.04.12.

[60] T.M. Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[61] MobClix. Technical. Available: `http://www.mobclix.com/faqs.html#faqs-5`. Accessed 20.05.12.

[62] MobFox. Mobfox sdk integration guide.

[63] AVG Mobilation. Malware information: Jimm. Available: `http://droidsecurity.appspot.com/securitycenter/securitypost_20110929.html`. Accessed 03.06.12.

[64] Andrew Ng. Machine learning. Available: `https://www.coursera.org/course/ml`. Accessed 13.06.12.

[65] Josh Ong. Android malware has jumped up 472% since july. Available: `http://www.appleinsider.com/articles/11/11/16/android_malware_has_jumped_up_472_since_july.html`. Accessed 16.02.12.

[66] Bob Pan. 17 bad mobile apps still up, 700,000+ downloads so far. Available: `http://blog.trendmicro.com/17-bad-mobile-apps-still-up-700000-downloads-so-far/`. Accessed 04.05.12.

[67] Linda Rosencrance. First trojan malware virus detected for android smartphones. Available: `http://www.securitynewsdaily.com/26-first-trojan-malware-virus-detected-for-android-smartphones.html`. Accessed 05.02.12.

[68] Artem Russakovskii. Massive security vulnerability in htc android devices (evo 3d, 4g, thunderbolt, others) exposes phone numbers, gps, sms, emails addresses, much more. Available: http://www.androidpolice.com/2011/10/01/massive-security-vulnerability-in-htc-android-devices-evo-3d-4g-thunderbolt-others-exposes-phone-numbers-gps-sms-emails-addresses-much-more/. Accessed 22.05.12.

[69] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206 –226, jan. 2000.

[70] Securelist. Worm.symbos.cabir.a. Available: `https://www.securelist.com/en/descriptions/old60663`. Accessed 02.03.12.

[71] StartApp. Permissions. Available: `http://www.startapp.com/permissions/`. Accessed 20.05.12.

[72] Tyler Style. Android 472% malware increase scare is sensationalist. Available: `http://www.nirdvana.com/2011/11/17/android-472-malware-increase-scare-sensationalist/`. Accessed 16.02.12.

[73] Vanja Svajcer. Android counterclank is (not) malware. Available: `http://nakedsecurity.sophos.com/2012/02/02/android-counterclank-is-not-malware/`. Accessed 19.03.12.

[74] Symantec. Android.arspam. Available: `http://www.symantec.com/security_response/writeup.jsp?docid=2011-121915-3251-99`. Accessed 07.06.12.

[75] Symantec. Android.counterclank technical details. Available: `http://www.symantec.com/security_response/writeup.jsp?docid=2012-012709-4046-99&tabid=2`. Accessed 18.03.12.

[76] Symantec. Android.ewalls technical details. Available: `http://www.symantec.com/security_response/writeup.jsp?docid=2010-073014-0854-99&tabid=2`. Accessed 05.02.12.

[77] Symantec. Android.golddream technical details. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2011-070608-4139-99&tabid=2. Accessed 03.06.12.

[78] Symantec. Androidos.fakeplayer technical details. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2010-081100-1646-99&tabid=2. Accessed 05.02.12.

[79] Symantec. Android.ozotshielder technical details. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2011-091505-3230-99&tabid=2. Accessed 10.05.12.

[80] Symantec. Android.rootcager technical details. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2011-030212-1438-99&tabid=2. Accessed 22.03.12.

[81] Symantec. Android.rufraud technical details. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2011-121306-2304-99&tabid=2. Accessed 19.03.12.

[82] Symantec. Threat explorer. Available: http://www.symantec.com/security_response/threatexplorer/azlisting.jsp. Accessed 05.02.12.

[83] TapJoy. Getting started with connect sdk (pay per install). Available: http://knowledge.tapjoy.com/integration-8-x/android/advertiser/getting-started-with-connect-sdk-pay-per-install. Accessed 20.05.12.

[84] AppBrain team. Infographic: The top 10 android ad networks. Available: http://blog.appbrain.com/2012/05/top-10-android-ad-networks.html. Accessed 20.05.12.

[85] Travis Credit Union. Phishing scam targeting android-based mobile devices. Available: https://www.traviscu.org/news.aspx?blogmonth=12&blogyear=2009&blogid=112. Accessed 20.03.12.

[86] Sara Yin. Android malware surges nearly five-fold since july. Available: http://securitywatch.pcmag.com/none/290654-android-malware-surges-nearly-five-fold-since-july. Accessed 16.02.12.

[87] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. Available: `http://www.csc.ncsu.edu/faculty/jiang/pubs/NDSS12_DROIDRANGER.pdf`. Accessed 30.04.12.

Appendix A

Surveyed Markets

Even with the large number of third-party android markets, very few of them are suitable for the purpose of this paper.

A.1 Selected Markets

Amazon AppStore http://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011. Currently only available to US residents, but we were still able to parse the site.

AppBrain http://www.appbrain.com/. Links to android market, but used to get the latest apps.

F-Droid http://f-droid.org/. FOSS (Free/Open Source Software) applications.

Slide Me http://slideme.org/

A.2 Not Selected Markets

1Mobile http://www.1mobile.com/. Reasons for non-selection: Does not list app permissions.

92Apk http://www.92apk.com/. Reasons for non-selection: Chinese language.

Akillirobot http://www.akillirobot.com/. Reasons for non-selection: Turkish.

Android Application Online https://www.andapponline.com/. Reasons for non-selection: Does not list app permissions.

Android Gateway https://www.androidgateway.com/. Reasons for non-selection: Does not list app permissions.

Android Pit http://www.androidpit.com/. Reasons for non-selection: Does not list app permissions.

Android Tapp http://www.androidtapp.com/apps/. Reasons for non-selection: Few apps, no latest list.

Android Zoom http://www.androidzoom.com/. Reasons for non-selection: Permissions list does not appear legit.

AndroLib http://www.androlib.com/. Reasons for non-selection: Links to market.

Androvation http://androvation.com/. Reasons for non-selection: Pirate site, but does not list permissions.

Androwire http://androwire.jp/. Reasons for non-selection: Japanese language.

AppChina http://www.appchina.com/. Reasons for non-selection: Chinese language.

Applanet http://applanet.net/. Reasons for non-selection: Pirate site, uses an application to access the applications.

Appoke http://appoke.com/. Reasons for non-selection: Does not list app permissions.

Appolicious http://www.androidapps.com/. Reasons for non-selection: Does not list app permissions.

AppsLib http://appslib.com/. Reasons for non-selection: Does not list app permissions.

AppstoreHQ http://android.appstorehq.com/. Reasons for non-selection: Links to market.

App Town http://www.apptown.com/Android/. Reasons for non-selection: Very little app information, does not list permissions.

BlapkMarket http://blapkmarket.com/. Reasons for non-selection: Pirate site. Login required, makes scraping the data infeasible. Links back to the Google Play market for permission information.

Brothersoft Mobile http://mobile.brothersoft.com/android/. Reasons for non-selection: Does not list app permissions.

Camangi http://www.camangimarket.com/. Reasons for non-selection: Does not list app permissions.

Droidmill http://droidmill.com/. Reasons for non-selection: Does not list app permissions.

EOE Market http://www.eoemarket.com/. Reasons for non-selection: Chinese language.

ESDN http://www.esdn.ws/. Reasons for non-selection: Does not list app permissions.

Get Jar http://www.getjar.com/. Reasons for non-selection: Does not list app permissions.

GoApk http://www.anzhi.com/. Reasons for non-selection: Chinese language.

Handango / Pocket Gear http://www.handango.com. Reasons for non-selection: Does not list app permissions.

Handster http://www.handster.com/. Reasons for non-selection: Does not list app permissions.

HiMarket http://apk.hiapk.com/himarket. Reasons for non-selection: Chinese language.

Hypermarket http://www.hyper.gen.tr/. Reasons for non-selection: Turkish language.

Indiroid https://indiroid.com/. Reasons for non-selection: Turkish language.

Insyde Market http://www.insydemarket.com. Reasons for non-selection: Does not list app permissions.

LG World http://uk.lgworld.com/web.main.dev. Reasons for non-selection: Does not list app permissions.

Mikandi http://www.mikandi.com/. Reasons for non-selection: Adult apps only. Requires application.

Mobihand / OnlyAndroid http://www.mobihand.com/. Reasons for non-selection: Does not list app permissions.

Nduoa http://www.nduoa.com/. Reasons for non-selection: Chinese language.

Nexva http://www.nexva.com/. Reasons for non-selection: Does not list app permissions.

No Crappy Apps http://nocrappyapps.com/. Reasons for non-selection: Links to android market.

Pdassi http://pdassi.de/. Reasons for non-selection: German language. Does not list app permissions.

Phoload http://www.phoload.com. Reasons for non-selection: Does not list app permissions.

Playandroid http://www.playandroid.com/. Reasons for non-selection: Games. Does not list app permissions.

Podnova http://android.podnova.com/. Reasons for non-selection: Does not list app permissions.

Samsung Apps http://www.samsungapps.com/. Reasons for non-selection: Does not list app permissions.

Soc.io http://mall.soc.io/apps. Reasons for non-selection: Does not list app permissions.

Ten Cents http://open.app.qq.com/. Reasons for non-selection: Chinese language.

Turkcell T-Market http://www.t-market.com/storefront/. Reasons for non-selection: Turkish language.

Ubinuri / Tstore http://www.tstore.co.kr/. Reasons for non-selection: Korean language.

Appendix B

Permissions

Percentage of applications requesting each permission by data set:

Permission	**Normal**	**Malicious**
ACCESS_ALL_DOWNLOADS	0.01	0.00
ACCESS_BLUETOOTH_SHARE	0.01	0.00
ACCESS_CHECKIN_PROPERTIES	0.06	0.95
ACCESS_COARSE_LOCATION	11.70	41.90
ACCESS_DOWNLOAD_MANAGER	0.00	0.95
ACCESS_DRM	0.00	0.95
ACCESS_FINE_LOCATION	22.77	39.05
ACCESS_LOCATION_EXTRA_COMMANDS	10.94	1.91
ACCESS_MOCK_LOCATION	2.30	0.95
ACCESS_NETWORK_STATE	56.62	41.90
ACCESS_PROVIDER	0.03	0.00
ACCESS_SURFACE_FLINGER	0.03	0.95
ACCESS_WIFI_STATE	10.08	25.71
ACCOUNT_MANAGER	0.02	0.95
ADD_VOICEMAIL	0.00	0.00
AUTHENTICATE_ACCOUNTS	0.17	0.95
BATTERY_STATS	0.32	1.90
BIND_APPWIDGET	0.12	0.95

BIND_DEVICE_ADMIN	0.01	0.00
BIND_INPUT_METHOD	0.02	0.95
BIND_REMOTEVIEWS	0.00	0.00
BIND_TEXT_SERVICE	0.00	0.00
BIND_VPN_SERVICE	0.00	0.00
BIND_WALLPAPER	0.04	0.00
BLUETOOTH_ADMIN	0.28	5.71
BLUETOOTH	0.47	6.67
BRICK	0.00	0.95
BROADCAST_PACKAGE_REMOVED	0.01	2.86
BROADCAST_SMS	0.05	1.90
BROADCAST_STICKY	1.60	2.86
BROADCAST_WAP_PUSH	0.02	0.95
CALL_PHONE	8.21	31.43
CALL_PRIVILEGED	0.53	0.95
CAMERA	6.40	6.67
CHANGE_COMPONENT_ENABLED_STATE	0.34	0.95
CHANGE_CONFIGURATION	1.25	1.90
CHANGE_NETWORK_STATE	0.52	4.76
CHANGE_WIFI_MULTICAST_STATE	0.10	0.95
CHANGE_WIFI_STATE	1.61	11.43
CLEAR_APP_CACHE	0.06	0.95
CLEAR_APP_USER_DATA	0.06	0.95
CONTROL_LOCATION_UPDATES	0.16	0.95
DELETE_CACHE_FILES	0.06	1.90
DELETE_PACKAGES	1.35	2.86
DEVICE_POWER	0.18	4.76
DIAGNOSTIC	0.00	0.95
DISABLE_KEYGUARD	0.51	4.76
DUMP	0.03	1.90
EXPAND_STATUS_BAR	0.03	1.90
FACTORY_TEST	0.00	1.90
FLASHLIGHT	2.94	2.86
FORCE_BACK	0.00	0.95
FORCE_STOP_PACKAGES	0.02	0.00

GET_ACCOUNTS	7.55	5.71
GET_PACKAGE_SIZE	0.05	0.95
GET_TASKS	2.34	7.62
GLOBAL_SEARCH_CONTROL	0.00	0.95
GLOBAL_SEARCH	0.00	0.95
HARDWARE_TEST	0.04	2.86
INJECT_EVENTS	0.00	1.90
INSTALL_DRM	0.01	0.95
INSTALL_LOCATION_PROVIDER	0.05	0.95
INSTALL_PACKAGES	0.49	19.05
INTERNAL_SYSTEM_WINDOW	0.00	0.95
INTERNET	84.05	91.43
KILL_BACKGROUND_PROCESSES	1.85	2.86
MANAGE_ACCOUNTS	0.20	0.95
MANAGE_APP_TOKENS	0.01	0.95
MASTER_CLEAR	0.00	0.95
MODIFY_AUDIO_SETTINGS	1.72	3.81
MODIFY_PHONE_STATE	0.25	5.71
MOUNT_FORMAT_FILESYSTEMS	0.00	0.95
MOUNT_UNMOUNT_FILESYSTEMS	0.88	29.52
NFC	0.04	0.00
PACKAGE_USAGE_STATS	0.00	0.00
PERSISTENT_ACTIVITY	0.08	1.90
PROCESS_OUTGOING_CALLS	0.51	5.71
READ_CALENDAR	0.18	0.95
READ_CONTACTS	6.40	43.81
READ_FRAME_BUFFER	0.02	0.95
READ_HISTORY_BOOKMARKS	2.77	0.00
READ_INPUT_STATE	0.00	0.95
READ_LOGS	1.61	9.52
READ_PHONE_STATE	44.03	78.10
READ_PROFILE	0.01	0.00
READ_SMS	0.87	48.57
READ_SOCIAL_STREAM	0.00	0.00
READ_SYNC_SETTINGS	0.15	2.86

READ_SYNC_STATS	0.03	0.95
REBOOT	0.09	1.90
RECEIVE_BOOT_COMPLETED	18.85	22.86
RECEIVE_MMS	0.05	7.62
RECEIVE_SMS	1.34	35.24
RECEIVE_WAP_PUSH	0.00	4.76
RECORD_AUDIO	2.86	3.81
REORDER_TASKS	0.02	1.90
RESTART_PACKAGES	0.13	10.48
SEND_DOWNLOAD_COMPLETED_INTENTS	0.01	0.95
SEND_SMS	1.55	61.90
SET_ACTIVITY_WATCHER	0.03	0.95
SET_ALARM	0.04	0.00
SET_ALWAYS_FINISH	0.00	1.90
SET_ANIMATION_SCALE	0.00	0.95
SET_DEBUG_APP	0.04	1.90
SET_ORIENTATION	0.11	0.95
SET_POINTER_SPEED	0.00	0.00
SET_PREFERRED_APPLICATIONS	0.28	1.90
SET_PROCESS_LIMIT	0.00	1.90
SET_TIME_ZONE	0.00	0.95
SET_TIME	0.01	0.00
SET_WALLPAPER_HINTS	0.19	1.90
SET_WALLPAPER	3.77	28.57
SIGNAL_PERSISTENT_PROCESSES	0.00	0.95
STATUS_BAR_SERVICE	0.04	0.00
STATUS_BAR	0.13	0.95
STOP_APP_SWITCHES	0.00	0.00
SUBSCRIBED_FEEDS_READ	0.01	0.95
SUBSCRIBED_FEEDS_WRITE	0.00	0.95
SYSTEM_ALERT_WINDOW	0.96	3.81
UPDATE_DEVICE_STATS	0.00	0.95
USE_CREDENTIALS	0.36	0.95
USE_SIP	0.10	0.00
VIBRATE	24.52	20.95

WAKE_LOCK	20.58	24.76
WRITE_APN_SETTINGS	0.03	9.52
WRITE_CALENDAR	0.67	1.90
WRITE_CONTACTS	3.56	30.48
WRITE_EXTERNAL_STORAGE	44.48	65.71
WRITE_GSERVICES	0.01	0.95
WRITE_HISTORY_BOOKMARKS	6.79	0.00
WRITE_PROFILE	0.00	0.00
WRITE_SECURE_SETTINGS	0.24	1.90
WRITE_SETTINGS	2.32	9.52
WRITE_SMS	0.39	19.05
WRITE_SOCIAL_STREAM	0.00	0.00
WRITE_SYNC_SETTINGS	0.15	3.81
WRITE_USER_DICTIONARY	0.05	0.00

Appendix C

Malware Permission Sets

The complete permission list of each malicious application examined in chapter 5. This includes both documented and third-party permissions.

DroidDream/Rootcager
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
CHANGE_WIFI_STATE
com.android.browser.permission.READ_HISTORY_BOOKMARKS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
com.android.launcher.permission.INSTALL_SHORTCUT
INTERNET
KILL_BACKGROUND_PROCESSES
READ_CONTACTS
READ_LOGS
READ_PHONE_STATE
RESTART_PACKAGES
WRITE_CONTACTS

Table C.2: Permissions requested by DroidDream/Rootcager

CounterClank/Apperhand
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
com.android.browser.permission.READ_HISTORY_BOOKMARKS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
com.android.launcher.permission.INSTALL_SHORTCUT
com.android.launcher.permission.READ_SETTINGS
com.android.launcher.permission.UNINSTALL_SHORTCUT
com.fede.launcher.permission.READ_SETTINGS
com.htc.launcher.permission.READ_SETTINGS
com.lge.launcher.permission.INSTALL_SHORTCUT
com.lge.launcher.permission.READ_SETTINGS
com.motorola.dlauncher.permission.INSTALL_SHORTCUT
com.motorola.dlauncher.permission.READ_SETTINGS
com.motorola.launcher.permission.INSTALL_SHORTCUT
com.motorola.launcher.permission.READ_SETTINGS
INTERNET
org.adw.launcher.permission.READ_SETTINGS
READ_PHONE_STATE
WAKE_LOCK
WRITE_EXTERNAL_STORAGE

Table C.4: Permissions requested by CounterClank/Apperhand

Geinimi
ACCESS_COARSE_LOCATION
ACCESS_COARSE_UPDATES
ACCESS_FINE_LOCATION
ACCESS_GPS
ACCESS_LOCATION
ACCESS_NETWORK_STATE
CALL_PHONE
CAMERA
com.android.browser.permission.READ_HISTORY_BOOKMARKS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
com.android.launcher.permission.INSTALL_SHORTCUT
com.android.vending.CHECK_LICENSE
com.google.android.googleapps.permission.GOOGLE_AUTH
GET_ACCOUNTS
GET_TASKS
INTERNET
MODIFY_AUDIO_SETTINGS
MODIFY_PHONE_STATE
MOUNT_UNMOUNT_FILESYSTEMS
READ_CONTACTS
READ_LOGS
READ_PHONE_STATE
READ_SMS
RECEIVE_BOOT_COMPLETED
RECEIVE_SMS
RECORD_AUDIO
REORDER_TASKS
RESTART_PACKAGES
SEND_SMS
SET_WALLPAPER
SYSTEM_ALERT_WINDOW
VIBRATE
WAKE_LOCK
WRITE_APN_SETTINGS
WRITE_CONTACTS
WRITE_EXTERNAL_STORAGE
WRITE_SMS

Table C.6: Permissions requested by Geinimi

GoldDream
ACCESS_COARSE_LOCATION
ACCESS_FINE_LOCATION
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
CALL_PHONE
DELETE_PACKAGES
INSTALL_PACKAGES
INTERNET
PROCESS_OUTGOING_CALLS
READ_PHONE_STATE
READ_SMS
RECEIVE_BOOT_COMPLETED
RECEIVE_SMS
SEND_SMS
WAKE_LOCK
WRITE_EXTERNAL_STORAGE

Table C.8: Permissions requested by GoldDream

Pjapps.A
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
CHANGE_NETWORK_STATE
CHANGE_WIFI_STATE
com.android.browser.permission.READ_HISTORY_BOOKMARKS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
DISABLE_KEYGUARD
INSTALL_PACKAGES
INTERNET
READ_PHONE_STATE
RECEIVE_MMS
RECEIVE_SMS
SEND_SMS
SET_PREFERRED_APPLICATIONS
VIBRATE
WAKE_LOCK
WRITE_APN_SETTINGS
WRITE_EXTERNAL_STORAGE

Table C.10: Permissions requested by Pjapps.A

Pjapps.B
ACCESS_COARSE_LOCATION
ACCESS_FINE_LOCATION
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
BLUETOOTH
BLUETOOTH_ADMIN
CAMERA
com.android.browser.permission.READ_HISTORY_BOOKMARKS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
com.android.launcher.permission.INSTALL_SHORTCUT
com.android.launcher.permission.UNINSTALL_SHORTCUT
com.estrongs.android.pop.PERMISSION
DISABLE_KEYGUARD
FLASHLIGHT
INSTALL_PACKAGES
INTERNET
KILL_BACKGROUND_PROCESSES
READ_CONTACTS
READ_PHONE_STATE
READ_SMS
RECEIVE_BOOT_COMPLETED
RECEIVE_MMS
RECEIVE_SMS
RESTART_PACKAGES
SEND_SMS
SET_WALLPAPER
VIBRATE
WAKE_LOCK
WRITE_EXTERNAL_STORAGE
WRITE_SETTINGS
WRITE_SMS

Table C.12: Permissions requested by Pjapps.B

Pjapps.C
ACCESS_CACHE_FILESYSTEM
ACCESS_COARSE_LOCATION
ACCESS_DOWNLOAD_MANAGER
ACCESS_DOWNLOAD_MANAGER_ADVANCED
ACCESS_DRM
ACCESS_FINE_LOCATION
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
CHANGE_NETWORK_STATE
com.android.browser.permission.READ_HISTORY_BOOKMARKS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
INSTALL_DRM
INSTALL_PACKAGES
INTERNET
READ_CONTACTS
READ_PHONE_STATE
READ_SMS
RECEIVE_BOOT_COMPLETED
RECEIVE_SMS
RECEIVE_WAP_PUSH
SEND_DOWNLOAD_COMPLETED_INTENTS
SEND_SMS
VIBRATE
WRITE_APN_SETTINGS
WRITE_CALENDAR
WRITE_CONTACTS
WRITE_EXTERNAL_STORAGE
WRITE_OWNER_DATA
WRITE_SETTINGS
WRITE_SMS
WRITE_SYNC_SETTINGS

Table C.14: Permissions requested by Pjapps.C

adSMS
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
BROADCAST_PACKAGE_ADDED
BROADCAST_PACKAGE_REMOVED
CHANGE_WIFI_STATE
DEVICE_POWER
INTERNET
KILL_BACKGROUND_PROCESSES
READ_PHONE_STATE
READ_SMS
RECEIVE_SMS
SEND_SMS
WAKE_LOCK
WRITE_APN_SETTINGS
WRITE_EXTERNAL_STORAGE
WRITE_SMS

Table C.16: Permissions requested by adSMS

JimmRussia
SEND_SMS
INTERNET
RECEIVE_SMS

Table C.18: Permissions requested by JimmRussia