KEIO UNIVERSITY

BACHELOR THESIS

Investigating Potentially Harmful Applications on Android

Author:
Koh YOU LIANG

Chief Examiner:
Dr. Keiji TAKEDA
Co-Examiners:
Dr. Jun MURAI
Dr. Osamu NAKAMURA

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Arts, Environment and Information Studies

in the

takeda lab. ISC
Jun Murai's Internet Research Lab

徳田・村井・楠本・中村・高汐・バンミーター・植原・三次・ 中澤・武田
合同研究プロジェクト

July 25, 2018

KEIO UNIVERSITY

Abstract

Faculty of Environment and Information Studies
takeda lab. ISC

Bachelor of Arts, Environment and Information Studies

Investigating Potentially Harmful Applications on Android

by Koh YOU LIANG

This paper talks about the ever increasing number of malware-riddled applications on the Android Play Store, and Google's lack of effective measures to counter them. I will propose and execute a proof-of-concept of my solution, which involves testing a random sample of 60 applications per category, using a two-pronged approach in identifying potentially harmful applications by performing robust regression of their permissions to find outliers, and then confirming these outliers by comparing the amount of traffic sent of a control phone with few contacts and a test phone with hundreds of contacts. A value is then assigned as a gauge of their potential risk and published towards the end of this paper.

Keywords: Android, Playstore, Google Play, PHA, Potentially Harmful Apps, Malware, Privacy, Mobile communication, computer network security, invasive software, mobile computing, public domain software, telecommunication traffic, transport protocols, personal information, HTTP traffic, statistical information, mobile bots, x86 malware

Acknowledgements

First and foremost, I would like to thank my project supervisor and advisor, Keiji Takeda, for providing me an opportunity to research on mobile security, a field that most undergraduate students do not get exposed to. Your patience and advice has been invaluable in the successful completion of this project.

Secondly, I would like to thank my family, especially my Mum and Dad for letting me become fully independent at an age younger than all of my peers. Thank you for not giving up on me back when I dropped out of high school, and for encouraging me to pursue my interest in computers. Thank you for believing in me, and for letting me come to Japan despite never having taken Japanese lessons. Your support has led me to become what I am today. I would also like to thank my little sister, Angela, for inspiring me to try living overseas.

Thirdly, I would like to thank my friends from Singapore, who have spurred me on with their excellence in academia, especially Ian who made it to CERN; and Matthias and DS, who have cheered me up during my darkest moments. Also, special thanks to Raynold, for introducing me to the real life applications of cyber security.

Fourthly, I would like to express appreciation for other members of ISC - Korry, my Eigo Jedi friend from Hawaii, and Jojo, Aaron and Bradley for all the pasta sessions we have had.

Lastly, I would also like to thank my peers in the GIGA Program, particularly Nick and Ival, who have been my comrades in arms, as well as Jie, Jake and Shiina. I will never forget the times we have spent together and the experiences we have shared. You guys have made living in Japan a very fun experience.

Contents

List of Figures

List of Tables

List of Abbreviations

APK	**A**pplication **P**ackage **K**it
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
HTTPS	**H**yper **T**ext **T**ransfer **P**rotocol **S**ecure
IAP	**I**n **A**pp **P**urchase
IP	**I**nternet **P**rotocol
NLP	**N**atural **L**anguage **P**rocessing
OS	**O**perating **S**ystem
OSS	**O**pen **S**ource **S**oftware
OOTB	**O**ut **O**f **T**he **B**ox
PHA	**P**otentially **H**armful **A**pplications
SDK	**S**oftware **D**evelopment **K**it
SNS	**S**ocial **N**etworking **S**ites
SVM	**S**upport **V**ector **M**achine
SSL/TLS	**S**ecure **S**ocket **L**ayer / **T**ransport **S**ecurity
VPN	**V**irtual **P**rivate **N**etwork

List of Symbols

$\%\vec{\Delta}$	percentage difference	%

This is a disclaimer stating that this project is done for research purposes only. My research was done in goodwill for the advancement of science and I never had, and never have any intention to defame any applications or companies. Assigned ratings do not necessarily mean that the application is definitely a malware; it only a possibility that it has been exhibiting potentially harmful behavior.

Chapter 1

Introduction

1.1 About the Author

The author of this paper is a fourth year undergraduate student of Keio University's Environment and Information Studies in Tokyo, Japan. He belongs to the Takeda Laboratory under Jun Murai's WIDE Group, with a focus on Cyber Security research under Professor Takeda.

1.2 Inspirations

The work presented in this paper was inspired by previous works on malware detection and analysis on the Android platform, and more generally, the works of software developers with the aim of protecting consumer privacy, such as those of Adblock and uBlock Origin ([8]). The Android market has always interested me, seeing as there to be relatively few people working on it as the common misconception seems to be that only rooted phones are vulnerable to malware, or they could only get infected through third party application markets. However, from the ExpensiveWall ([4]) and Judy ([2]) malware outbreak that occurred just in 2017, we can see that it was not true.

I would also like to touch on four misconceptions that also served as an inspiration for my topic:

Firstly, antivirus apps does not protect against viruses. By the time it is on the phone, the system has already been infected and it will be too late to rectify. Moreover, even the top and leading antivirus apps have a very high false negative rate [22], and have low detection rates for actual malware, which leads us to question their detection methods. In addition to that, the top antivirus apps all have critical vulnerabilities, and none of the promoted security features are even sufficiently secure. [17] It has been proven possible, and very easy to disable the malware-scanning engine remotely or access confidential data via privilege escalation vulnerabilities such as broken SSL communication and self-developed crypto implementations.

Secondly, malware is not restricted only to rooted phones. As history has shown us[7], there have been outbreaks of malware in the Google Playstore which has affected many non-rooted phones. Also, the Google Playstore is just as vulnerable to malware as are third party application markets.

Thirdly, while not directly related to malware, firewall apps on the Android platform essentially acts as a VPN with traffic filtering options, requiring end-user knowledge in customizing these filters.

Thus, even with a top firewall app and a leading antivirus, neither would prevent PHAs from infecting the phone or sending privacy-violating information about the user.

1.3 Contributions

This research lays the basis for readers to do their own quantitative research on the Android Playstore. Upon completion, researchers would be able to adopt the proposed methodology used in order to analyze the entire application ecosystem, i.e. including the unofficial third party ones such as Aptoide. By using a physical device with the implementations of automation, researchers will find that the project is scalable, and can further expand upon this paper by using hosted cloud servers to gather the traffic data from multiple applications at the same time, thereby improving on the accuracy of the analysis by changing the time into a constant non-variable, thus streamlining the equation. Research scholars will be able to understand the differences between data gathering with a physical device and an emulated one, in that smart malware using user-agent checks to circumvent malware detection can be observed in its natural state.

The source code for the implementation has been made open source on Github.[1]

1.4 Thesis Structure

This thesis is structured as follows:

Chapter 1 is a brief introduction of the author's affiliations.

Chapter 2 gives the reader a cursory background of the Android platform and its importance in the relation to the whole market, as well as consumer behavior. It also mentions the existing works and how the author differentiates his thesis by improving on them and exploring the analysis from a new angle.

Chapter 3 introduces the key ideas and the formulation thereof. The main objective of the project is defined, which is to publish the a comprehensive statistic of potentially harmful applications (PHAs) in the Play Store.

Chapter 4 illustrates the implementation of the author's ideas using a two-pronged approach by firstly filtering applications that need to be analyzed meticulously, and then describing the utilization of Content Length Analysis as a means to analyze traffic flows.

Chapter 5 presents the results obtained from the experiments. Ratings are assigned linearly based on a histogram charted from the percentage difference in traffic volume when comparing between the real and test cases. A total number of percentage of applications per category is then published and evaluated for error.

Chapter 6 closes the paper by summing up the findings and yielding the conclusion drawn from the evaluation. It also includes future work that the author hopes to accomplish.

[1] Investigating Content Length: https://github.com/Isopach/bachelors-thesis

Chapter 2

Background

This chapter will introduce the basic concepts and conceived justifications that will be utilized the investigation.

2.1 Information

2.1.1 Market Share

Android is currently the most used smart phone mobile device platform globally, taking up 85.0% of the market share [11], followed by iOS at 14.7% - a large difference. As of now, there are over 3.7 million applications on the Play Store, with over 0.9% of them having over a million downloads. Monetization SDKs serving advertisements and sending user data are dominating the top 10 embedded SDK list, and the latest OS released almost a year ago still has not reached 5% penetration ([1]).

As of the end of last year, there are over 2.3 billion mobile phone users [21]. 85% of these would make up over 25% of the world's population, making it the second most consumed consumer market, second only to computer ownership that lies at 38%[6]. With the ever-growing market, it is critical that we treat threats to the general safety of its users as a cause for concern, which is the aim of this project.

2.1.2 Android Malware Trends

There are two billion monthly active devices running Android. In 2014, there were more than 650,000 individual pieces of malware for Android discovered. In 2016, PHAs (potentially harmful apps) were on 0.05% of devices compared to 0.15% in 2015. The raw percentage has decreased, but with the increase in devices, in reality it is still affecting millions of users. Furthermore, in 2017 alone there were over 40 million devices infected by the ExpensiveWall and Judy malware[7].

These frequent malware outbreaks occur when a group of applications slip through Play Protect's scanning. Bypassing Play Protect's scanning via various methods such as adding delays between code execution as elementary and thus it cannot be relied on to provide a safe environment for consumers. As such, this paper aims to bring attention to the risks users are putting themselves in by trusting Google's Play Protect as an all-round form of protection, which has no safeguard against privacy-violating applications. With that said, there is a general lack of Internet Law that prevents such issues from occurring, which is another problem in itself [10].

2.2 Malware

2.2.1 Types and Families

There are many types of malware and they can be grouped into different families; however, this thesis will focus on those related to the network and traffic communications. Botnets is one of the common family types when analyzing network flows, however this thesis will be putting its focus on the actual content of the traffic packet as opposed to a specific family.

2.2.2 Attack Vectors

Gullible users is the first thing that comes to mind when talking about security. No matter how good a system may be, users are the weakest link when it depends on them to make decisions [9], such as whether to grant permissions to use an app, or whether to give personal information such as identification cards or credit card information for the purpose of verification, without first reading the lengthy privacy policy and terms of use.

Gaming the system with ranking fraud is also another commonly used tactic. In the Android Play Store and other markets, certain statistics such as user reviews and ratings, number of downloads, search frequency, page views are used to determine the ranking of an app [23]. Out of those, at least three - user reviews, ratings and downloads can be controlled by the app developer by either developing bots or by paying third-party sites to boost these statistics. This is exactly what happened with the ExpensiveWall and Judy malware[7] - when a certain download/rating threshold is reached, users tend to trust those apps even if the apps are newly published [15].

Misplaced trust in OSS is also another vector. While it is true that code may be reviewed when it is open-source, it is often not as there are far more developers writing code than there are those validating them - it is a fact that the code is not reviewed when uploaded to the Play Store; moreover, the burden of trust is placed upon the user, which is a justification of this point. In the most recent reported case [5], the ARC Welder Chrome extension app, which has over 900,000 downloads, has become a victim where the fake extension's developer has managed to remove the original app from the search engine indexer - i.e. when a user attempts to search for the app, only the fake one appears. In just a few days, it has achieved 32,000 installs and Google has taken no actions against the app or its developer.

LISTING 2.1: `malicious_excerpt.js`

```
var i = document.createElement("script");
i.id = "dexscriptid";
i.src = "https://votetoda.com/ext/script.php?id=ukr&
   track=true";
var a = document.getElementById("dexscriptid");
if (a === null) {
    document.body.appendChild(i)
    }
```

Listing 2.1 above shows an abstract of the malicious javascript code, which is injected into every page the user visits, tracking the user's activities and stealing login information.

2.3 Analysis Methods

2.3.1 Long Tail Outlier Analysis

Long tail analysis is an effective detection method when trying to find outliers. Given that most applications are not malware, apps within the same category but require some of the least frequently used permissions are determined to be outliers and are deemed reasonably suspicious to be a possible PHA. Hence, we will be using the long tail analysis in order to find these outliers.

However, while the traditional long tail analysis used in statistics as a gauge of popularity takes the cutoff with the Power Law in mind (Figure 2.3.1), at the point where both regions - the long tail and the dominating tail - is of equal area, our analysis takes the lowest common frequency in the long tail (Figure 5.1.1) and marks them for further analysis. This is done to minimize the false positives.

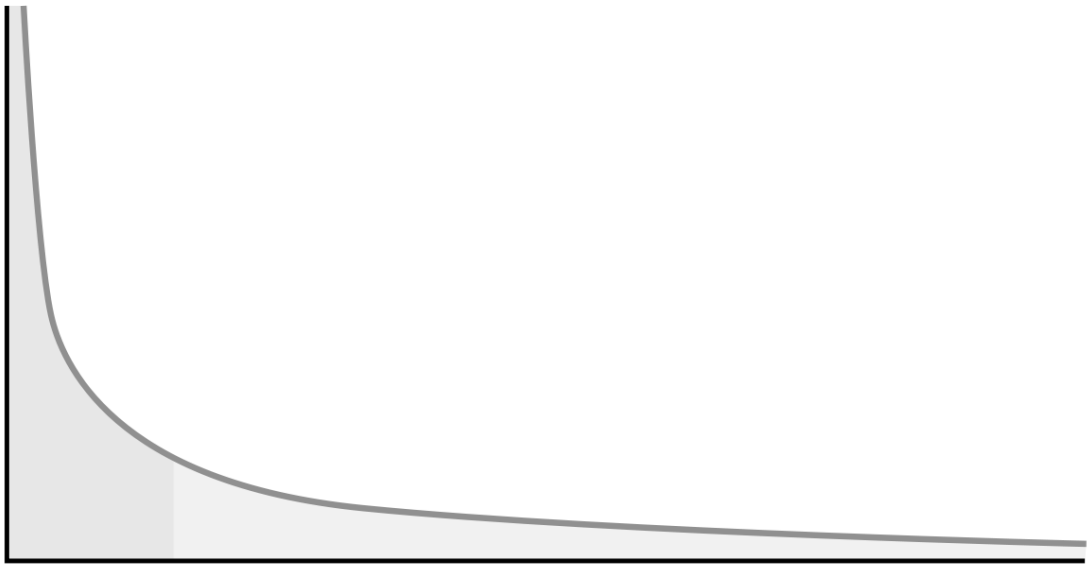

FIGURE 2.3.1: Long Tail Analysis Using Power Law [20]

2.4 Existing Works

There are a couple of existing works that I would like to discuss, since they have served as either a basis or inspiration for this paper.

2.4.1 Detecting Malware Leveraging Text Semantics of Network Flows

The closest related work to this project is a similar research [19] done by Chinese scholars, which was published in the IEEE earlier this year (2018). It agrees that there is a growing emergence of malicious applications, and that most malware applications rely on the network to perform their attacks and steal information. Its methods involve processing HTTP flows as text to extract features using NLP string analysis and the N-gram sequence generation. It claims to have achieved an accuracy of 99.15% detecting 5258/31706 apps as malicious using an SVM algorithm. With the help of machine learning, it can detect 54.81% of malicious apps in a real environment. The methodology used is similar to mine; however, it is tailored to detecting malware whereas my project is based on detecting PHAs based on privacy violations. Thus, while I only analyze the content length header, this project attempts to analyze every feature, which would theoretically make it relatively more accurate and fair. The entire project was done on an Android emulator, which has differences from a physical device which was used in my project, namely that its user agent is that of an emulator by default and cannot be realistically changed, as well as its base code is different from that of a stock Android-based phone, such as being rooted by default.

2.4.2 Data Mining of Permissions Mode

The inspiration for this paper is that permissions-based detection does not take into account permission patterns and only considers the permissions asked at installation (pre-Android 7.0), not those used at runtime [18]. Hence, it is superficial to use only permissions-based detection methods as it is trivial to hide dangerous permissions by requesting them at runtime. Furthermore, if a manifest scanning method is to be implemented alone, it has a tendency to generate many false positives as apps have a tendency to add more requested permissions than it actually needs or uses, which is often due to the result of manifest merger after the `build.gradle` is compiled. Hence, I will be using a two-pronged implementation as detailed in Chapter 4.

2.4.3 Explaining Black-box Android Malware Detection

Recent findings [14] have highlighted that despite impressive performance on benchmarked datasets, those results are very fragile, showing that very minute changes of Android malware may suffice to avoid detection. Hence, it can be inferred that the more accurate the results of an implementation it has seemingly achieved, the more easily it is to effect the results significantly by altering a single variable. This can be referenced back to Section 2.2.1, where the results in a real environment only has about a 50% success rate. Thus, the goal of my project is not to create a perfectly accurate but fragile implementation that detects and determines in a binary-fashion whether an app is a malware or not, but one of a confidence evaluation, where each suspicious app is assigned a rating to determine their potential harmfulness.

Chapter 3

Key Ideas and Objective

3.1 Problem

One common factor in the aforementioned existing research is that emulators are always used; this means that the user agent is always that of an emulator, which is often associated with testing/development environments and thus do not contain data of a real user, such as contacts and SNS communications.

A developer with the aim of circumventing traffic-based malware detection methods would hence be able to detect these emulators based on the User-Agent HTTP request header, and not execute its malicious/privacy violating activities when an emulator is detected. The headers can be spoofed however, but without significant alterations to the source code, or programmatically manipulating all requests being sent out through the emulator, it would be impossible to change the user-agent using an OOTB implementation. Hence, this project aims to fill the gap in the research by using an actual physical phone, as mentioned in Section 1.3 (Contributions).

3.2 Hypothesis

PHAs exhibit malware-like characteristics and behavior and are still very common because existing detection and screening systems such as Google's Play Protect is not advanced enough to detect these applications.

3.3 Goal

This project has two main objectives:

1. To identify the number of applications that are exhibiting malware-like behavior in 3 randomly selected categories.

2. To create and present a system for investigating the content length of any APK.

A rating would also be given to applications in (1.) to represent a gauge of their potential harmfulness. This thesis aims to published the definite, exact number of apps that may be a PHA per analyzed category of the bestselling apps, by assigning a standard deviation rating on a confidence scale.

Chapter 4

Design and Implementation

4.1 Environment

The experiments take place on a physical Samsung Galaxy Note 3 phone running the Android 5.0 Lollipop OS. Using a 64-bit Windows 10 Home Edition laptop as a proxy (Figure 4.1.1), HTTP traffic is routed through a TL-WR841N TP-LINK router, creating a local network environment where traffic can be monitored and intercepted. Burp Suite is the choice for the task. By installing an exported self-signed certificate on the Android device, HTTPS traffic secured by SSL/TLS certificate is able to be decrypted on the computer acting as a proxy host, granting full access to analysis and manipulation.

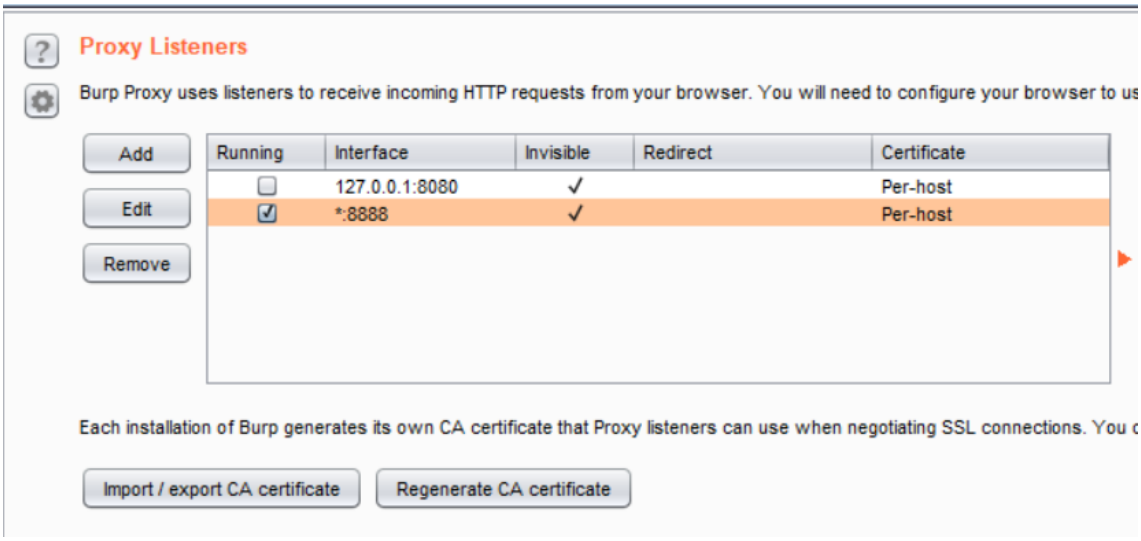

FIGURE 4.1.1: Proxy Listener Setting on Computer

Figure 4.1.2 shows the proxy settings on the phone. By entering the computer's IP on the local network as a manual proxy, we can route the traffic through the computer before the first hop. Notice that the port is set to 8888, matching the listener port on our traffic interception software.

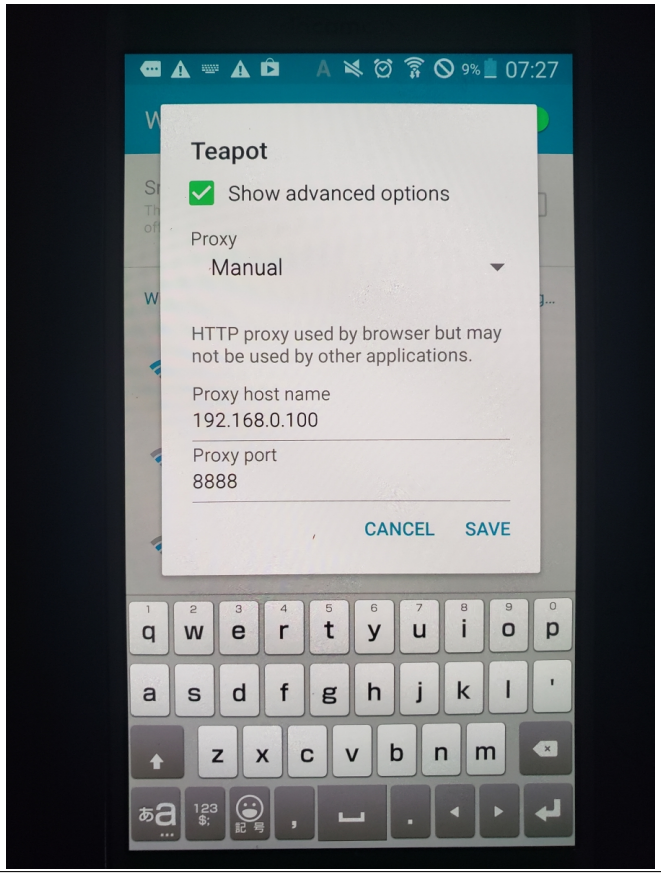

FIGURE 4.1.2: Setting up proxy on phone

To simulate user interaction, the Android application 'Tasker' is used to create macros of my touches on the screen (Figure 4.1.3) such that the same actions are taken in repeated experiments, eliminating changes in results caused by variations in user actions. Regarding repeated experiments, the author believes that since contact information seldom changes, or changes only very slightly, the app often only takes it once, if ever. Hence, the app data must be wiped between consecutive experiments to ensure accuracy.

Using a wait time of 10ms, a full cycle of screen touches takes approximately 164 seconds, or 2.73 minutes.

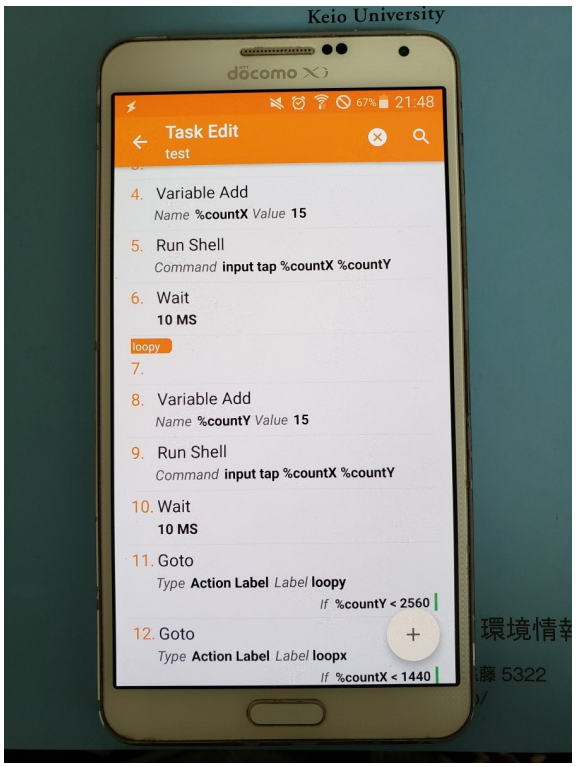

FIGURE 4.1.3: Tasker Profile on Phone

The pseudo code used in Tasker is based on the Python code shown in Listing 4.1. We use a nested loop in order to loop through the coordinates from (0,0) to (1440,2560), which is a representation of the screen resolution (pixels) of the phone.

LISTING 4.1: `tasker.py`

```
countx=range(1,1440)
county=range(1,2560)
y=1
for x in range(0,len(countx)):
    x+=1
    print(x,y) #Run Shell in Tasker
    for y in range(0,len(county)):
        y+=1
        print(x,y) #Run Shell in Tasker
```

With this, our environment set up is complete and we can now proceed to the implementation.

4.2 Two-pronged Methodology

The approach this paper uses for the detection of PHAs is based on permissions that are used by the PHA, and the volume of data being transferred. Permissions are detected in the manifest file and cross referenced with that in the Google Play Store to prevent obfuscation. By graphing and comparing permission frequency counts, network flows of applications are then analyzed using Burp Suite.

My hypothesis is based on the assumption that errant developers aiming to affect as many users as possible with their malware tendencies would choose a free/freemium model for their applications. As such, only the top 60 free applications in each category would be analyzed.

4.2.1 First prong - Permissions Frequency Analysis

Firstly, the package names of applications that are intended for analysis are scraped from the Play Store, via one request for each of the 53 categories. Using a list of categories in an array, packages' names are scraped with Python as shown in Listing 4.1:

LISTING 4.2: getapps.py

```
for i in range(0,len(category)):
    url = "https://play.google.com/store/apps/category/"
       ↪ +category[i]+"/collection/topselling_free?hl=
       ↪ ja"

    html = requests.get(url).text
    soup = bsoup(html)

    urlslist = soup.findAll("a", { "class" : "card-click
       ↪ -target" })
    urls = []
    #open the file to keep the list
    filename = url[44:-33] + ".txt"
    fo = open(filename, 'w')

    #Url list
    for a in urlslist:
        link = "https://play.google.com" + a['href']
        urls.append(link)
    url = urls[::4]
    for item in url:
        item = item[46:] #list as package name
        fo.write("%s\n" % item)

    fo.close()
```

Secondly, APKs are then downloaded from the Play Store using *gplaycli*, a Google Play Download via Command line ([13]). In the event the rate gets limited, a VPN is used to circumvent download restrictions. Permissions are also scraped from the Play Store, initially as an alternative way of obtaining data, but acting as a cross-reference as well as a backup in the later stages.

With a list of packages from the Play Store, we are able to scrape its permissions by appending it to a link programmatically, as shown in Listing 4.2 below:

LISTING 4.3: scraper.py

```
for p_name in package_names:
    row += 1
    col = 0
    id = p_name

    url = "https://play.google.com/store/apps/details?id
       ↪ ="+id
    driver.get(url)
    checker = 4
    while (True):
        try:
            element = driver.
               ↪ find_element_by_css_selector(
                "#fcxH9b > div.WpDbMd > c-wiz > div >
                   ↪ div.ZfcPIb > div > div.JNury.
                   ↪ Ekdcne > div > c-wiz:nth-child(" +
                   ↪  str(
                    checker) + ") > div > div.JHTxhe >
                       ↪ div > c-wiz > div > span > div
                       ↪  > span > div > a")
            print ("***************" + str(element.text
               ↪ ).strip())
            if (str(element.text).strip() == "View
               ↪ details"):
                print "***********Found"
                element.click()
                break
        except:
            checker -= 1
    time.sleep(5)
    new_element = driver.find_element_by_css_selector("#
       ↪ yDmH0d > div.llhEMd.bYEzqc.iWO5td > div > div.
       ↪ g3VIld.LhXUod.t89eC.Up8vH.J9Nfi.iWO5td")

    sttr = new_element.text
    str_list = sttr.split("\n")
    print str_list
    key = "perm"
    results[key] = []
    for i in range(1,len(str_list)):
        list_line = str_list[i].split(" ")
        if(len(list_line) > 1 and not str_list[i].
           ↪ __contains__("&") and
               not str_list[i].__contains__("/") and not
                  ↪  str_list[i].__contains__("In-app
                  ↪ purchases")
```

```
            and not str_list[i].__contains__("Wi-Fi
                ↪ connection information")):
                results[key].append(str_list[i])
        else:
            key = str_list[i]
            results[key] = []
    worksheet.write(row,col,p_name)
    for ke in results:
        col += 1
        new_str = ""
        new_str += (ke + ":    ")
        worksheet.write(row, col, new_str)
        for value in results[ke]:
            col += 1
            worksheet.write(row, col, value)


    results = {}
workbook.close()
driver.close()
```

This gives an unsorted raw permissions list as shown in Figure 4.2.1 below:

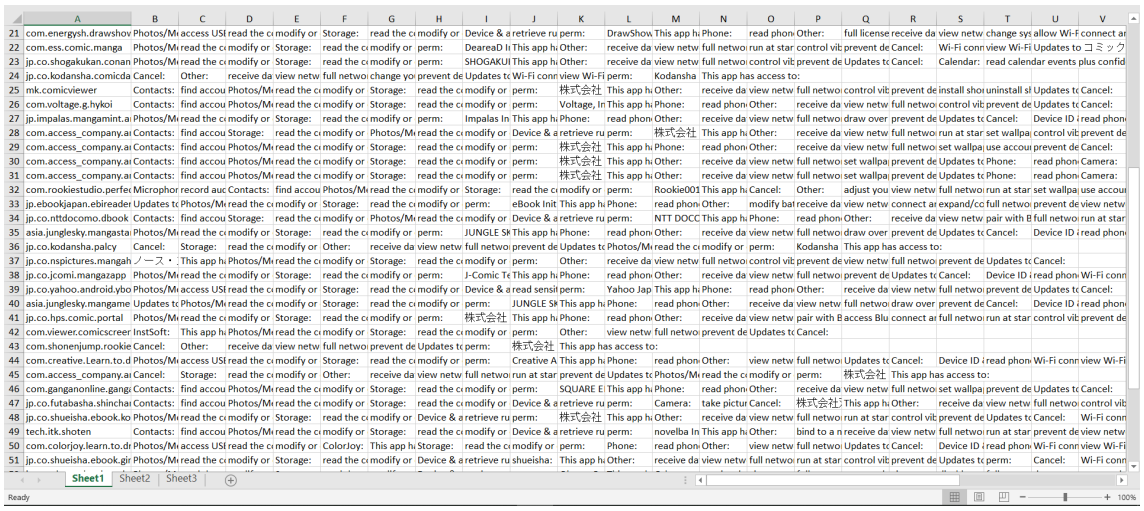

FIGURE 4.2.1: Raw Permissions Data (Comics)

The permissions are then charted in order of descending frequency. As there is a lot of unneeded data, we use a regex to match words containing category headers, share a *":"* in their names, and company names, or begins with *"Updates to"* - basically everything that is not a permission - and remove them from the frequency chart before we plot them. Table 4.1 shows the frequency table used in the plotting of the comics' category's frequency analysis, after cleaning the data.

Unique Values	Count
view network connections	60
full network access	60
receive data from Internet	55
prevent device from sleeping	55
read the contents of your USB storage	54
modify or delete the contents of your USB storage	53
view Wi-Fi connections	40
read phone status and identity	26
control vibration	25
find accounts on the device	24
run at startup	19
retrieve running apps	15
use accounts on the device	9
set wallpaper	8
draw over other apps	7
add or remove accounts	6
take pictures and videos	5
create accounts and set passwords	4
change your audio settings	4
modify system settings	4
install shortcuts	4
connect and disconnect from Wi-Fi	4
access USB storage filesystem	4
read Google service configuration	3
disable your screen lock	3
read sensitive log data	2
approximate location (network-based)	2
precise location (GPS and network-based)	2
read your own contact card	1
change network connectivity	1
reorder running apps	1
full license to interact across users	1
change system display settings	1
allow Wi-Fi Multicast reception	1
read calendar events plus confidential information	1
uninstall shortcuts	1
record audio	1
adjust your wallpaper size	1
modify battery statistics	1
expand/collapse status bar:	1
access Bluetooth settings	1

TABLE 4.1: Permissions Frequency (Comics)

The lower tail, or long tail of the approximately 30% least occurring permissions in terms of frequency count will be deemed suspicious regardless of its nature, and its corresponding applications will be analyzed in the second prong. A figurative representation is shown in Figure 5.1.1.

4.2.2 Second prong - Traffic analysis

Using the environment described in Section 4.1, we will prepare two profiles on the same phone. The first profile is to be used as a control and will be hereon known as *Control*, while the second profile will be used in an attempt to prove the hypothesis, hereon to be known as *Test*. *Control* will contain just one contact - the owner's phone number and email address; whereas *Test* will contain over a hundred of the author's real phone contacts, taken with permission. Google accounts will be used to switch between these two profiles by syncing and removing contacts as needed.

We will then route the traffic through a laptop as described, and monitor the flows through Burp Suite. On first starting up of the application, the data is always discarded as many applications have introductory tutorials on the usage of the application that is difficult to duplicate without reinstalling. Also, because of the fact that loaded resources may be cached as both GET and POST requests are being included in calculating the content length, it is crucial to let the resources load once and get cached in the application settings' storage in order to not let it inflate the content length artificially.

Running the Tasker script, user interaction is automatically simulated and traffic is recorded through Burp Suite for a time period of one minute. The raw traffic data, containing non CSS and image requests, from Burp Suite will then be exported. This is done to filter out resources that are aesthetic and hence irrelevant to the analysis. A sample of one item is shown in Listing 4.4. Request and response bodies have been truncated due to space constraints.

LISTING 4.4: `drawshow_test.xml`

```
<item>
   <time>Wed Jul 18 20:00:48 JST 2018</time>
   <url><![CDATA[https://d19hf05jf0kxo4.cloudfront.net/
      mobile/publicOauthServiceNew.php]]></url>
   <host ip="13.33.9.73">d19hf05jf0kxo4.cloudfront.net
      </host>
   <port>443</port>
   <protocol>https</protocol>
   <method><![CDATA[POST]]></method>
   <path><![CDATA[/mobile/publicOauthServiceNew.php
      ]]></path>
   <extension>php</extension>
   <request base64="true"><![CDATA[UE9TVCAvbW9iaWMA
      ==]]></request>
   <status>200</status>
   <responselength>539</responselength>
   <mimetype>script</mimetype>
   <response base64="true"><![CDATA[SFRUUC8=]]></
      response>
   <comment></comment>
 </item>
```

One *item* refers to a single packet being sent/received. The content lengths are summed programmatically in the script shown in Listing 4.5 below.

Using a regex to extract the content length by searching the inside of each *<response base64="true">* tag using the *CDATA[]* to retrieve its value, then decoding the string by Base64. The *Content-Length:* string is then searched for via regex again, and the rest of the code is self-explanatory. The value is then converted to a decimal and then written into an excel worksheet (not shown in the Listing).

The variable *xmlfilename* is changed according to the file being analyzed.

LISTING 4.5: `countlength.py`

```
# Read File
match = re.findall(r'CDATA\[\w+\==',open('xmlfilename','
    ↪ r').read())
# Removes string unused in Base64 code (but used for
    ↪ regex)
match = ([s.replace('CDATA[', '') for s in match])
# Flattens the list by joining elements
match = ' '.join(match)
# Decode into text
match = base64.b64decode(match)
# Finds value of Content-Length
match = re.findall(r'Content-Length: \d+',match)
# Extracts Content Length
match = ([s.replace('Content-Length: ', '') for s in
    ↪ match])
# Converts to integer
match = [int(x) for x in match]
# Sums all content length
count = sum(match)
print count
```

Given that all controllable variables - duration, environment and approximate time - are held constant, *ceteris paribus*, if a significantly (>2%) greater amount of data is transferred in the *Test* case, as compared to the *Control*, the application will be determined to be potentially harmful as it might be sending additional privacy-violating information. Ratings will be given in the results analysis in Chapter 5, based on percentage length difference in comparison to other PHAs of the same category.

4.3 Limitations

Firstly, the limitations of the Permissions Frequency Analysis has restricted my ability to analyze more than a small fraction of the Google Play Store. As this project has taken place over the last half a year, there has been updates to multiple applications. As such, I could not simply rely on the APKs that I have downloaded - which is a limitation in itself. I had to also scrape the Play Store in addition, and make sure that the permissions that I had obtained at the start were the same as they were at present time. As such, during the scraping stage, I actually noticed that a majority, if not all of the applications have "Updates to <app name> may automatically add additional capabilities within each group." listed in its Playstore permission list. This makes each application's analysis transient, because an application which is not initially a malware can suddenly become a malware after an update, and/or remove it to evade malware scanners with the aim of producing a report, such as this paper.

Secondly, this project concerns the Japan playstore, which means there will be a lack of third party research to confirm for malware. Error rate would hence be difficult to determine.

Thirdly, an app with malware behavior in IAPs will completely circumvent my detection method, as I have no budget and will not be purchasing anything.

Fourthly, despite the thesis being titled 'PHA', as it only focusing on HTTP network flows, PHAs that do not go through the usual network, e.g. the entire communication is done via WebSockets, then there would be no way of detecting them with my methods. Or, they could be PHAs in other ways not involving privacy abuse, which would be out-of-scope for this thesis.

Fifthly, apps with frequent updates where the developer adds and removes PHA behaviors in alternate updates would also circumvent my detection method, as the analysis is only done for one version of the app. However, this is unlikely without insider knowledge of them being scrutinized.

Sixthly, apps requiring registration only have their launch screen analysed because there is no way to automate that for now.

Seventhly, my data is taken over a range of several months, so the date/time analyzed may have affected the data integrity

Eighthly, the automated macro I have programmed only include taps. This means that swiping gestures which may be required to trigger certain functions were not used and hence not analyzed. As such, some function calls may have been left out as they could not be triggered.

Ninthly, app developers wanting to circumvent my method of detection could simply only use permissions that are within the top 20% of the frequency chart. However, even so if that is done, dangerous permissions such as READ_CONTACTS would generally only be used by a PHA in most categories, and hence we have done all we can to surpass this limitation by including those apps even if they are not in the lower bounds.

Chapter 5

Results and Evaluation

5.1 Results: Comics Category

5.1.1 Permission Frequency Analysis

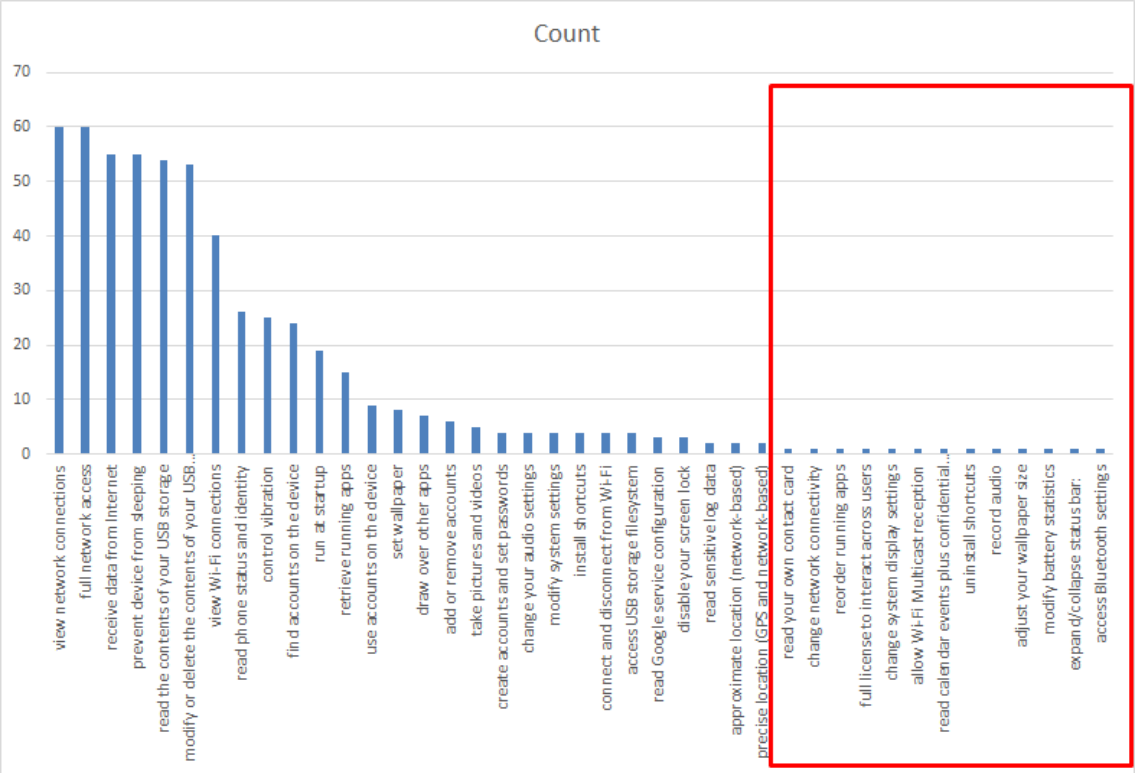

FIGURE 5.1.1: Permissions Frequency Analysis (Comics)

In Figure 5.1.1 above, these are the permissions for the top 60 bestselling free applications in the Comics category, arranged according to descending frequencies. We can see that at the long end of approximately 30% from the end of the tail, there lies a group of permissions with only a count of 1 - hence, we'll be analyzing the traffic of these applications in order to facilitate efficiency. 8 applications account for the 13 permissions tied for the lowest frequency count, highlighted by the red box drawn above.

5.1.2 Content Length Analysis

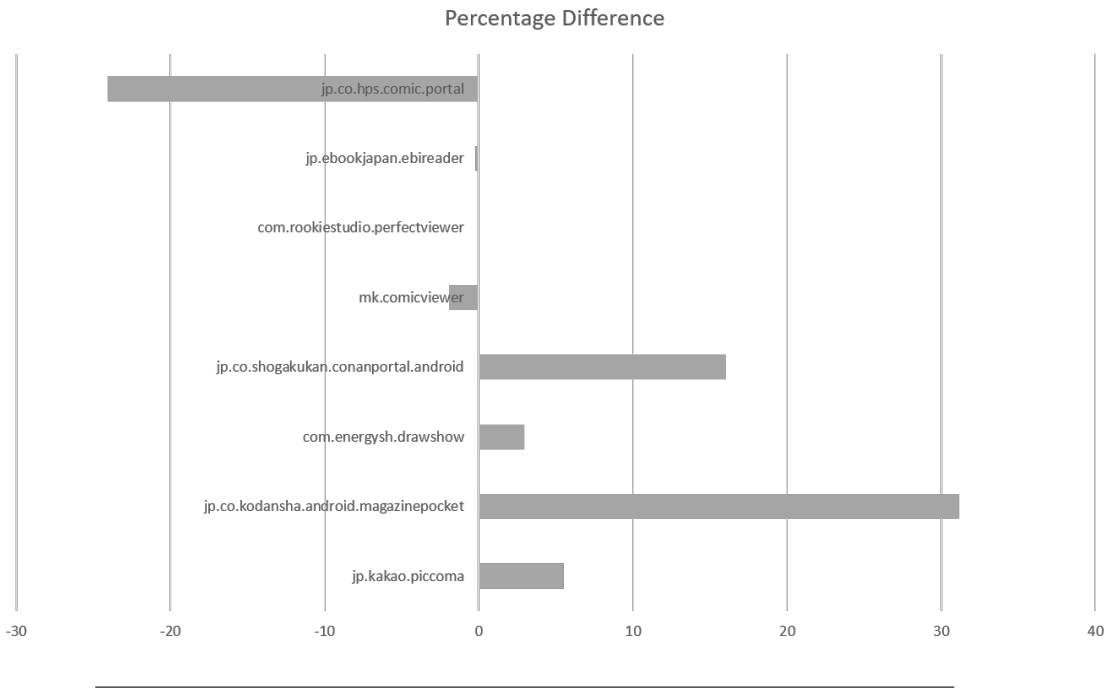

FIGURE 5.1.2: Content Length Analysis (Comics)

Figure 5.1.2 shows the result of the traffic analysis from Section 4.2.2. Those on the right are positive differences, i.e. the traffic for Real is greater than Control, while those on the left is the opposite.

As explained in my earlier plan, I took traffic data over an equal length of time for both *Control* and *Real*, and then divided the *Control* over *Real* multiplied by a hundred to find the difference.

$$\%\vec{\Delta} = 100 - (\frac{H_0}{H_1})x100 \tag{5.1.1}$$

Equation 5.1.1 shows how the percentage difference is calculated. H_0 denotes the null hypothesis, also known as the *Control* case, and H_1 denotes the alternate hypothesis, which in this case refers to the *Test* case.

5.1.3 Rating

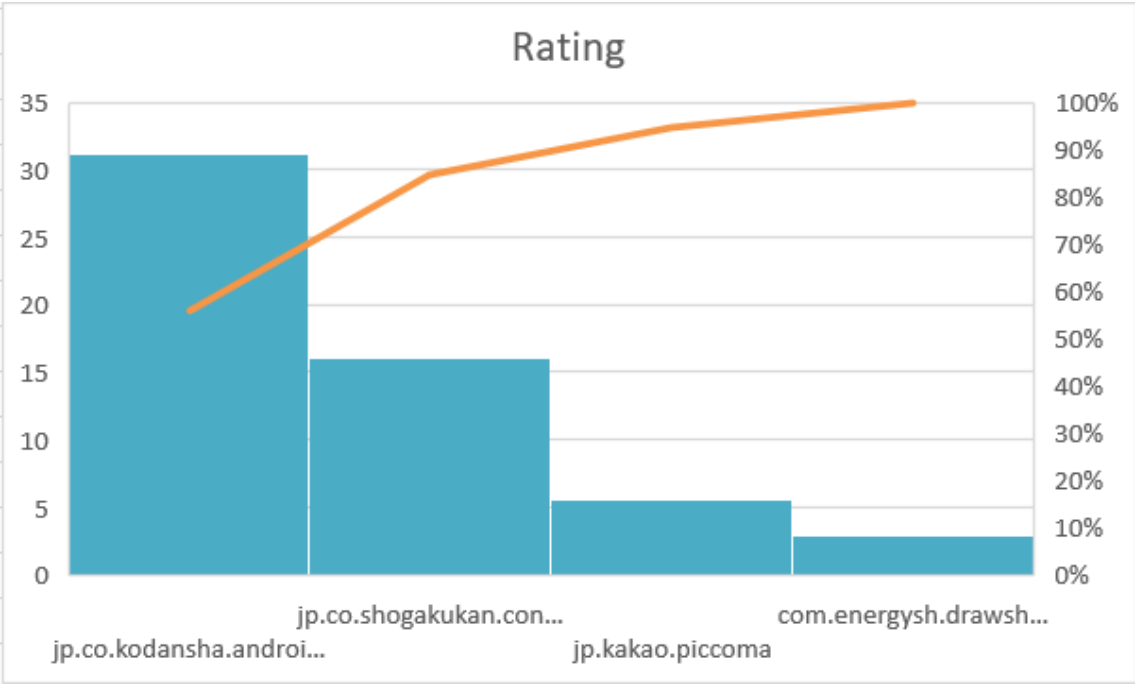

FIGURE 5.1.3: Percentage Difference (Comics)

Figure 5.1.3 above shows the percentage difference of each of the applications relative to one another. In the second and third rows of the plot, the difference was less than 2%, hence we will exclude them from rating as they are deemed to be equivalent. We can see that the *magazinepocket* application which has over 30% difference accounts for over half of the cumulative area in the histogram chart. Hence, we will rate it as it is and assign it a rating of 60. This is followed by *conanportal*, which will be assigned a value of 30; and then followed by *piccoma* with the value of 7; and lastly *drawshow* with the value of 3.

$$\lim_{r \to 100} f(x) = 1 \tag{5.1.2}$$

In Equation 5.1.2, as the rating r approaches 100, the more confident we are that it is potentially harmful. f(x) is illustrated by the green line in Figure 5.1.3.

5.2 Results: Weather Category

5.2.1 Permission Frequency Analysis

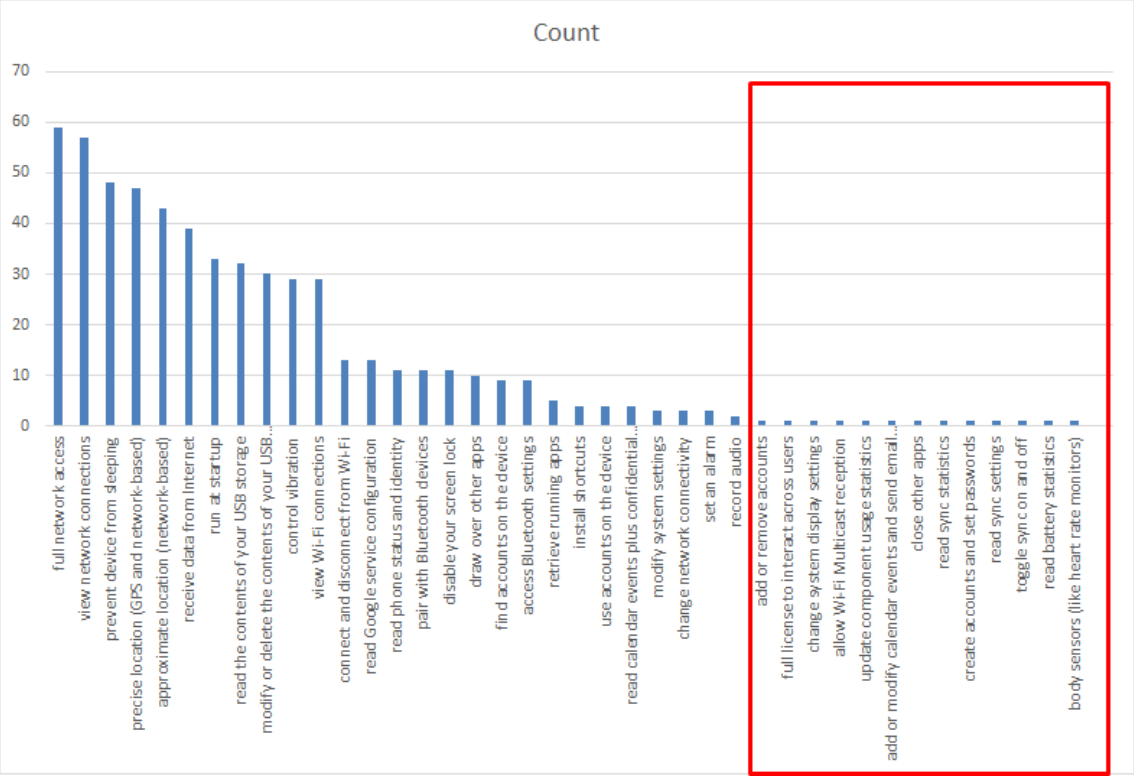

FIGURE 5.2.1: Permissions Frequency Analysis (Weather)

Just like in the previous section, shown in Figure 5.2.1 above are the permissions for the top 60 bestselling free applications, but in the Weather category. Arranged in order of descending frequencies, there are coincidentally again 13 frequencies belonging to 8 applications, which are tied for the lowest frequency count; this is highlighted by the red box drawn above.

5.2.2 Content Length Analysis

There was a significantly higher number of readings in this category where the control data has greater content length relative to the test case's data, or the total difference is close to 100%, meaning that one case has almost 10 times the content length, which is unlikely to be caused by privacy violations alone. We speculate that this is because the location data is being read and used to load resources on the screen, which are often script or json data and not images, hence they inflated the content length in a way that does not necessarily mean that it is sending the users' contact information.

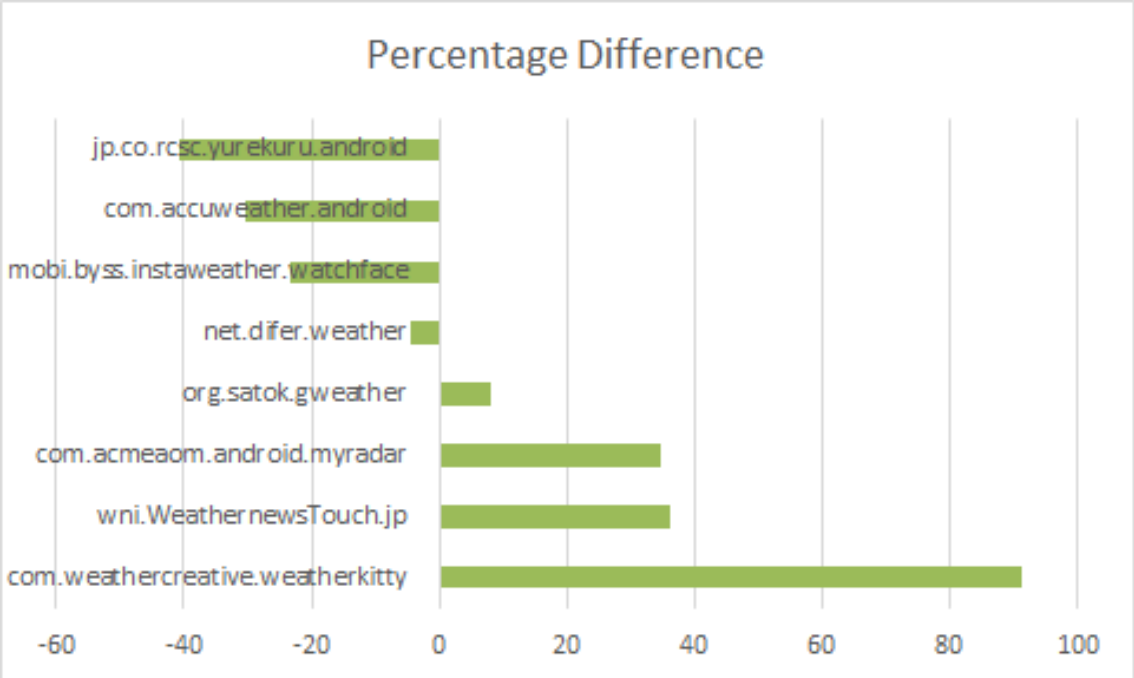

FIGURE 5.2.2: Content Length Analysis (Weather)

From Figure 5.2.2 above, we can see that 4 out of 8, or half of the applications analyzed in the second prong of our methodology has negative percentage difference. These 4 apps will hence not be analyzed in the evaluation stage, since it is deemed that the alternate hypothesis H_1 has been rejected, and thus H_0 is defaulted to be true.

5.2.3 Rating

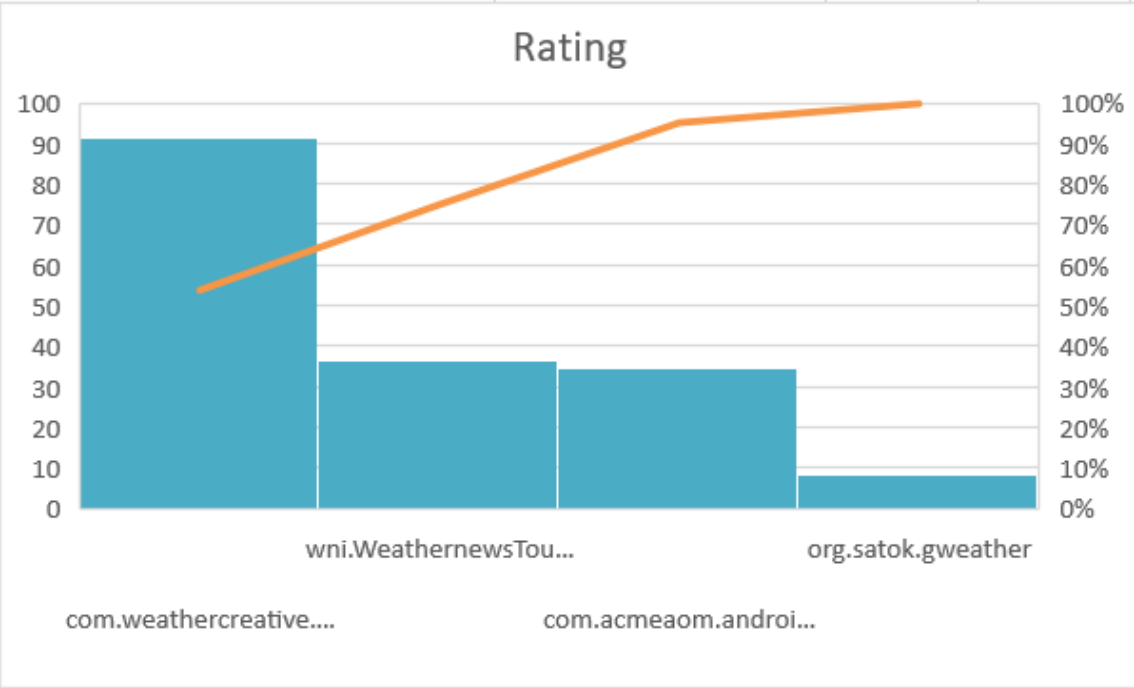

FIGURE 5.2.3: Percentage Difference (Weather)

Although Figure 5.2.3 shows a rating model based on previous experiments, our confidence rate for this category is lower than it is in 5.1.3, as the results seem skewered. We can see that again there is the app *WeatherKitty* is taking up over 90% of the rating - whether this is mere coincidence remains to be seen.

5.3 Results: Dating Category

5.3.1 Permission Frequency Analysis

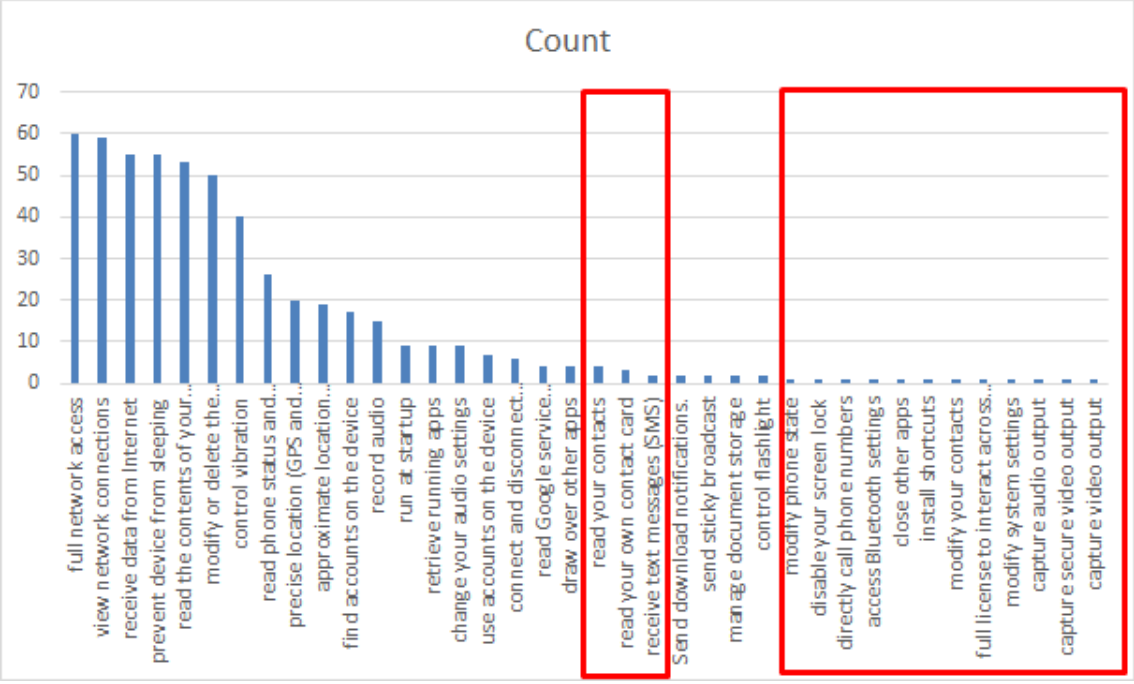

FIGURE 5.3.1: Permissions Frequency Analysis (Dating)

The long tail is larger in this category than in the others. Hence, instead of just taking the lower bounds of 30%, we shall take the extended long tail, to include several dangerous permissions such as READ_CONTACT and RECEIVE_SMS in the middle bounds area. The permissions to be analyzed are denoted by the red rectangles drawn in Figure 5.3.1 above. There are a total of 12 permissions belonging to 7 apps in the original lower bounds area and a further 3 dangerous permissions being used by 4 apps in the middle bounds area.

5.3.2 Content Length Analysis

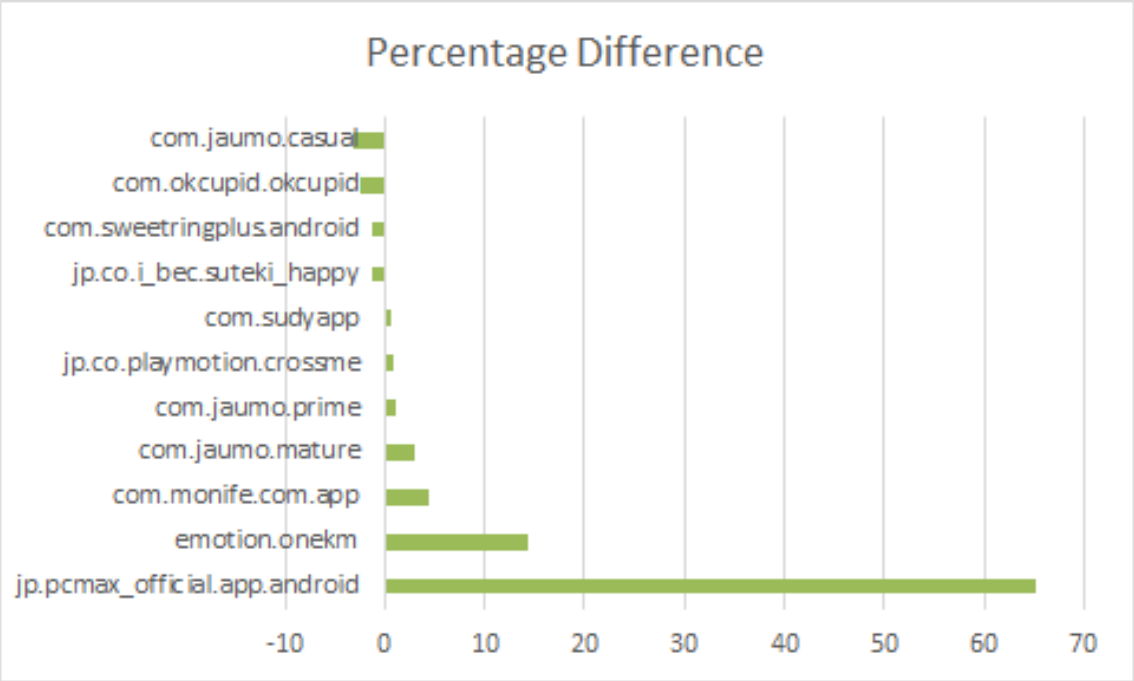

FIGURE 5.3.2: Content Length Analysis (Dating)

It is important to note that for this category, registration plays an important part in using the app and being recognized as an actual valid user. However, this was not done due to the lack of automation tools for it; as such, actions only took place on the landing page of the app. The general lack of significant content length difference can be attributed to barely any functions being activated.

5.3.3 Rating

7 applications crossed the 2% significance barrier, hence we have rated them. As *pcmax* has had a 65%$\vec{\Delta}$, it is determined to be the most suspicious in our experiment.

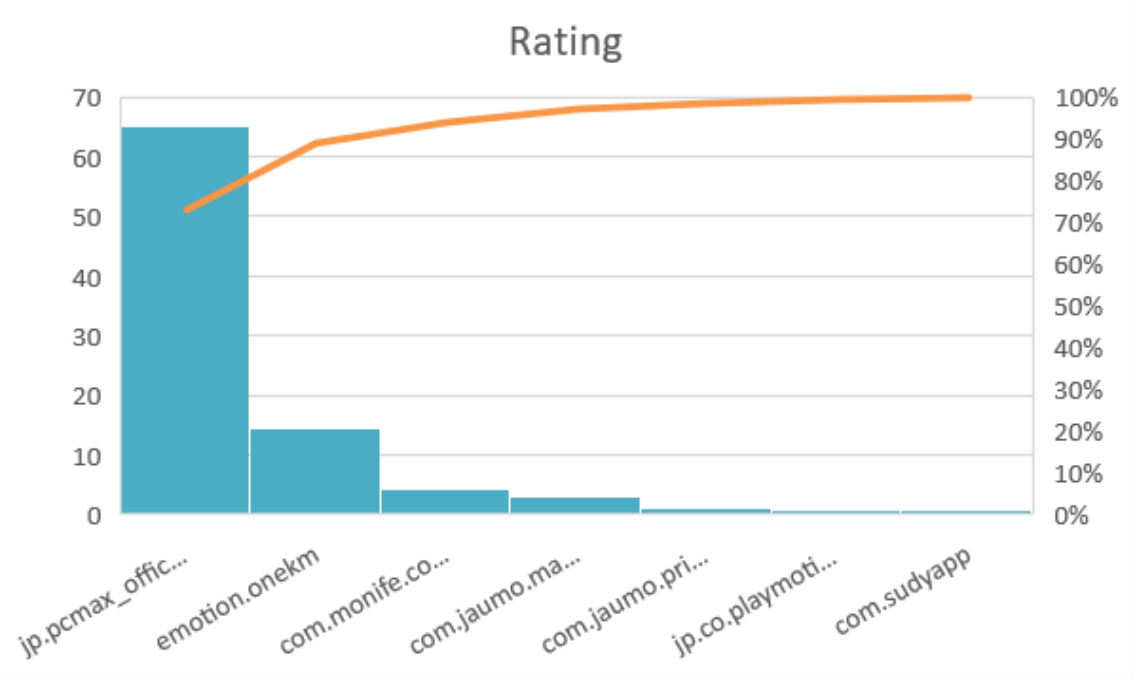

FIGURE 5.3.3: Percentage Difference (Dating)

However, as mentioned in the previous subsection, since no functions were invoked in the capturing of data, these ratings should not be taken at face value.

5.4 Overall Evaluation

I would like to preface this by mentioning that in order for the analysis to scale to the entire Play Store, the each step simply has to be performed the same way as it was, except instead of having only the top 60 apps in each category, it should be done with all the apps in a category, then repeated per category.

Regarding cross-referencing with VirusTotal for false positives, I have tried but it has not been fruitful as there was no data on over half the apps analyzed in Step 2. This is likely because the Japanese market has not been analyzed as thoroughly as the one in the USA, hence the preexisting data is very limited. Moreover, an app might not be a malware *per se*, but simply violating the user's privacy by sending unauthorized sensitive data. This is further proven when known privacy-violating apps such as Sarahah is checked with VirusTotal - it was marked as clean despite several reports [3] [12].

Permissions Frequency Analysis is limited by the fact that the analyzed number of applications are a small drop of the total in the playstore, hence it might not represent overall results. However, we can say for a certain that it shows accurate distribution for the top bestselling apps of each analyzed category.

Content Length Analysis is limited in that the data was taken over a few tries; if there was more time, I would have wanted to take it over a 100 times and taken the average, to eliminate errors based on time difference and human errors.

Chapter 6

Conclusion

6.1 Summary

This thesis has given a brief summary of the research progress in various subfields of Android malware and PHAs, and reviewed the advantages and shortcomings of each methodology used for the analysis.

In total, 180 apps were analyzed across 3 categories; 27 apps were scrutinized for their traffic, and 15 apps made it to the final round of ratings.

Using the results from each category, we can conclude that the category with the highest percentage of PHAs is the Dating category, with a total of 7/11 apps being potential PHAs.

Also, it seems that every category has at least 8/60 applications that are using uncommon permissions tied for the lowest frequency. This may be because of the sample size being small; the results are expected to be very different when scaled to the millions.

It is also interesting to note that there is currently no law in Japan that prohibits the sharing of consumer data without permission, hence this thesis should only be used for educational purposes.

6.2 Future work

A difference of less than 2% would mean that it would be excluded from being rated; however, that number was determined arbitrarily and as the total content length grows larger, particular when it is over 100,000, even a huge difference in contacts - i.e. 100 more contacts - should only cause a small percentage increase in content length. Hence, if time permits in the future, I would like to reanalyze the data I have obtained in order to determine an appropriate cut off percentage difference for analysis.

I would also like to consider cross referencing with apps that are definitely not PHAs (those that are proven in reports and thesis), and take the average percentage difference for each category as the error rate margin.

Regarding the lack of analysis in one of the most the popular categories - 'Games' - it is because games often require complex, non-random gestures, hence I left it out in my analysis as the barriers to entry of creating a specialized testing system just for the category is unrealistic. It can be considered if a greater time span was provided.

As personal future work, I intend to analyze the entire Play Store as part of my future graduate thesis. For now, the Top 60 bestselling free applications in each category demonstrates that there is a possibility of there being more PHAs than Google has anticipated or discovered.

Also, I would like to filter traffic based on specific hosts an app interact with. Due to the large amount of connections running at once, even when restricting background data of other apps, there were a lot of interference and I could only hope that it was equal throughout the different readings during my experiments.

6.3 Closing thoughts

I had initially planned to analyze the entire Playstore; however, this was not possible due to time limits and the manpower of a single person over slightly half a year. Although data was collected for over a hundred thousand applications, most of it was unusable due to an oversight on my part in not isolating the target application's traffic, leading to there being too much noise to be programmatically filtered out. Also, due to the limited storage space capacity, applications had to be uninstalled after the testing has been done, creating an overhead cost needed to repeat the same test.

A possible improvement would be to only look at hosts related to the application itself. This would require creating a whitelist of hosts related to the application provider, which can be programmatically obtained under *url* in Listing 4.4.

Another improvement would be to only read the packets that uses the POST method to send data. As our aim is to detect privacy violations, it would make more sense to only look at data being sent to the server, and not data being retrieved. However, this was not done because there is also a method of sending data not via POST but by the GET method, which is done via appending a query string to the request, that gets sent to the server as a part of the URL.

Appendix A

Frequently Asked Questions

Why did you choose the top 60 apps in each category? When the 'top bestselling' category is selected, the page that loads shows the top 60 applications by default. Hence, it was chosen for convenience and the fact that Google made it 60 might have some meaning behind it.

Why Android 5.0 Lollipop OS? At the point of starting the project in 2017, Android 5.0/5.1 was the most widely used OS version. Moreover, currently over half the market uses Android 5.1 and below [16].

Why did you select those categories? The categories are selected completely at random using a pseudo-random number generator in Python. Each category is sorted alphabetically and the minimum and maximum indexes are used as ranges for the python script, which produces a random number used to select each category.

Why only the Japan Playstore? Most analysis have been done on the global Playstore market or the one in the USA. Very few analysis have been done specifically for the Japan Playstore.

What was your budget? As for the budget, I have none so I only focused on the free apps.

Have you considered about the In-App Purchase Contents? Having a paid IAP rigged with malware would completely circumvent my detection method.

Why is it only restricted to apps with a server connection? I feel that is it an important research as this type of malware involves the most damaging type of privacy violation as it sends the information to a third party.

Appendix B

Glossary

barriers to entry are the costs or other obstacles that prevent new competitors from easily entering an industry or area of business. [1]
botnet is a number of Internet-connected devices, each of which is running one or more bots. [2]
bots is a device or piece of software that can execute commands, reply to messages, or perform routine tasks, as online searches, either automatically or with minimal human intervention. [3]
ceteris paribus stands for 'all other things being unchanged or constant'. It is used in economics to rule out the possibility of 'other' factors changing. [4]
Content-Length The Content-Length entity-header field indicates the size of the entity-body, in decimal number of OCTETs, sent to the recipient or, in the case of the HEAD method, the size of the entity-body that would have been sent had the request been a GET. [5]
Freemium is a pricing strategy by which a product or service is provided free of charge, but money (premium) is charged for additional features, services, or virtual (online) or physical (offline) goods. [6]
GET is a HTTP Method used to request data from a specified resource.
n-gram is a contiguous sequence of n items from a given sample of text or speech. [7]
Power Law states that a relative change in one quantity results in a proportional relative change in another. [8]
POST is a HTTP Method used to send data to a server to create/update a resource.
Python is a programming language
Regex is a regular expression. In theoretical computer science and formal language theory, it is a sequence of characters that define a search pattern. [9]
Whitelist is the practice of identifying entities that are provided a particular privilege, service, mobility, access or recognition. [10]

[1] definition taken from Investopedia article 'Barriers to Entry', retrieved on 2018-07-18.
[2] definition taken from Wikipedia article 'Botnet', retrieved on 2018-07-18.
[3] definition taken from Dictionary.com article 'Bots', retrieved on 2018-07-18.
[4] definition taken from Economic Times article 'Freemium', retrieved on 2018-07-18.
[5] definition taken from RFC 2616 Fielding, et al., retrieved on 2018-07-18.
[6] definition taken from Wikipedia article 'Freemium', retrieved on 2018-07-18.
[7] definition taken from Wikipedia article 'N-gram', retrieved on 2018-07-18.
[8] definition taken from Statisticshowto article 'Power Law', retrieved on 2018-07-18.
[9] definition taken from Wikipedia article 'Regular Expression', retrieved on 2018-07-18.
[10] definition taken from Wikipedia article 'Whitelisting', retrieved on 2018-07-18.

Bibliography

[1] D. Abbot. "Google Play Store Data: 3.7mn Apps, $14.6bn spends, 60bn Installs, Top Categories, CPI Worldwide and over 100 data points". In: *Medium* (Apr. 2018). URL: https://growthbug.com/google-play-store-data-3-7mn-36331f2c8b26.

[2] Intan Nurfarahin Ahmad et al. "Android Mobile Malware Classification using Tokenization Approach based on System Call Sequence". In: *Proceedings of the World Congress on Engineering and Computer Science*. Vol. 1. 2017.

[3] Natasha Bach. "Sarahah Has Been Downloading All the Data In Your Address Book". In: (2017). URL: http://fortune.com/2017/08/28/sarahah-app-downloading-contact-list/.

[4] David Bird. "Mobile Threat". In: *ITNOW* 59.4 (2017), pp. 46–47.

[5] Dr. Walter Bishop. "PSA: Avoid the fake Android App Runtime (ARC Welder) extension (with over 32,000 installs!) on the Chrome Web Store, as it injects ads and trackers into every webpage you visit. It is the only one that shows up in search results and the real one appears to be installable, but delisted/"noindex"-ed". In: (2018). URL: https://www.reddit.com/r/Android/comments/8uucob/psa_avoid_the_fake_android_app_runtime_arc_welder/.

[6] Pew Research Center. "Global Computer Ownership". In: (2015). URL: http://www.pewglobal.org/2015/03/19/internet-seen-as-positive-influence-on-education-but-negative-influence-on-morality-in-emerging-and-developing-nations/technology-report-15/.

[7] Thomas Fox-Brewster. *Google Is Fighting A Massive Android Malware Outbreak – Up To 21 Million Victims*. [Online; accessed July 20, 2018]. 2017. URL: https://www.forbes.com/sites/thomasbrewster/2017/09/14/massive-google-android-malware-expensivewall/#51971848477f.

[8] gorhill. "uBlock Origin". In: *Github Repository* (June 2018). URL: https://github.com/gorhill/uBlock.

[9] John D Halamka. "Privacy and Security". In: *Key Advances in Clinical Informatics*. Elsevier, 2018, pp. 79–86.

[10] ICANN. "ICANN Is Not the Internet Content Police". In: (2015). URL: https://www.icann.org/news/blog/icann-is-not-the-internet-content-police.

[11] IDC. "Smartphone OS market share". In: *IDC Research Report* (Mar. 2017). URL: https://www.idc.com/promo/smartphone-market-share/os.

[12] Mohit Kumar. "Beware! Viral Sarahah App Secretly Steals Your Entire Contact List". In: (2017). URL: https://thehackernews.com/2017/08/sarahah-privacy.html.

[13] matlink. *Google Play Downloader via Command line*. Github Repository. Tools. Feb. 2018.

[14] Marco Melis et al. "Explaining Black-box Android Malware Detection". In: *arXiv preprint arXiv:1803.03544* (2018).

[15] Ilung Pranata, Rukshan Athauda, and Geoff Skinner. "Determining trustworthiness and quality of mobile applications". In: *International Conference on Mobile Wireless Middleware, Operating Systems, and Applications*. Springer. 2012, pp. 192–206.

[16] Statista. "Android operating system share worldwide by OS version from 2013 to 2018". In: *The Statistics Portal* (June 2018). URL: https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/.

[17] Siegfried Rasthofer Stephan Huber. "How to Do it Wrong: Smartphone Antivirus and Security Applications Under Fire". In: *DEF CON* (2018). URL: https://www.defcon.org/html/defcon-24/dc-24-speakers.html#Huber.

[18] Chao Wang et al. "Research on data mining of permissions mode for Android malware detection". In: *Cluster Computing* (2018), pp. 1–14.

[19] Shanshan Wang et al. "Detecting android malware leveraging text semantics of network flows". In: *IEEE Transactions on Information Forensics and Security* 13.5 (2018), pp. 1096–1109.

[20] Wikipedia, the free encyclopedia. *An example power-law graph, being used to demonstrate ranking of popularity. To the right is the long tail, and to the left are the few that dominate (also known as the 80–20 rule).* [Online; accessed July 12, 2018]. 2006. URL: https://commons.wikimedia.org/wiki/File:Long_tail.svg.

[21] Sue York and Ray Poynter. "Global Mobile Market Research in 2017". In: *Mobile Research*. Springer, 2018, pp. 1–14.

[22] Yajin Zhou et al. "Hey, you, get off of my market: detecting malicious apps in official and alternative android markets." In: *NDSS*. Vol. 25. 4. 2012, pp. 50–52.

[23] Hengshu Zhu et al. "Discovery of ranking fraud for mobile apps". In: *IEEE Transactions on knowledge and data engineering* 27.1 (2015), pp. 74–87.