A Detection Framework for Android Financial Malware

by

Andi Fitriah Abdul Kadir

Master of Computer Science, IIUM, 2013
Bachelor of Computer Science, IIUM, 2009

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

In the Graduate Academic Unit of Faculty of Computer Science

Supervisor(s):	Ali A. Ghorbani, Ph.D., Computer Science, Natalia Stakhanova, Ph.D., Computer Science
Examining Board:	Bradford Nickerson, Ph.D., Faculty of Computer Science, Rongxing Lu, Ph.D., Faculty of Computer Science, Constantine Passaris, Ph.D., Faculty of Arts
External Examiner:	Nur Zincir-Heywood, Ph.D., Faculty of Computer Science, Dalhousie

This dissertation is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2018

Abstract

As attempts to thwart cybercrime have intensified, so have innovations in how cybercriminals provision their infrastructure to sustain their activities. Consequently, what motivates cybercriminals today has changed: from ego and status, to interest and money; the cybersecurity researchers today have turned their attention to financial-related malware, especially on the Android platform. There are some major issues faced by researchers in detecting Android financial malware. Firstly, what constitutes Android financial malware is still ambiguous. There is a disparity in labelling the type of malware where most of the current detection systems emphasize the recognition of generic malware types (e.g., Trojan, Worm) rather than indicating its capabilities (e.g., banking malware, ransomware). Without knowing what constitutes financial malware, the detection systems are not capable of providing an accurate recognition of an advanced and sophisticated financial-related malware. Secondly, most of the current anomaly-based detection systems via machine learning suffer from inaccurate evaluation and comparison due to the lack of adequate datasets, which result in unreliable outputs for real-world deployment. Due to time-consuming processes, most of the available datasets are crafted mainly for static analysis, and those created for dynamic analysis are installed on an emulator or sandbox. Sophisticated malware can bypass these approaches; malware authors have employed obfuscation methods and included a wide range of anti-emulator techniques, where the malware programs attempt to hide their malicious activities by detecting the emulator. These deficiencies are some of the major reasons why Android financial malware is able to avoid detection.

A comprehensive understanding of the existing Android financial malware attacks supported by a unified terminology and high-quality dataset is required for the deployment of reliable defence mechanisms against these attacks. Therefore, we seek to understand trends and relationships between Android malware families and devise a taxonomy of Android financial malware attacks. In addition, a systematic approach to generate the required datasets is presented to address the need to use physical platforms instead of emulators. In this regard, an automated dynamic analysis system running on smartphones is developed to generate the desired dataset in a testbed environment. In order to correlate the generated dataset and the proposed taxonomy, a hybrid framework for malware detection is presented. We propose a novel combination of both static and dynamic analysis based specifically on features derived from the string literal (statically via reverse engineering) and network flow (dynamically on smartphones). This combination can assist security analysts in recognizing the threats effectively. We employ five common classifiers to construct the best model to identify malware at four levels: detecting malicious Android apps, classifying Android apps with respect to malware category and sub-category, and characterizing Android apps according to malware family. Specifically, a dataset containing over 5,000 samples is used to evaluate the performance of the proposed method. The experimental results show that the proposed method with a Random Forest classifier achieves an accuracy of over 90% with a very low false positive rate of 4% on average.

Dedication

O Lord, increase me in knowledge while keeping me humble

For my parents,

Andi Abdul Kadir & Niswa Mohd Noor,

who gave me wings.

For my late grandparents,

Andi Arfah Andi Saleh & Andi Dinar Andi Bombang,

Mohd Noor Usman & Siti Dinar Saide,

who taught me how to spread my wings.

For family, teachers, & friends,

who encouraged me to fly.

♥♥♥

Acknowledgements

My heartfelt appreciation to both of my supervisors, Dr. Stakhanova and Dr. Ghorbani for their dedicated guidance, encouragement, and warm support throughout my studies. It was a privilege to have been given the opportunity to study at the Canadian Institute for Cybersecurity (CIC). I gratefully acknowledge the help of my examining board, especially Dr. Lu and Dr. Nickerson for taking the time to provide me with constructive comments in completing my dissertation. I am indebted to my parents for their unconditional love and support. Without my father's encouraging words and my mother's caring nature, I would not have been resilient enough to make it through my four years here.

A very special thanks goes to my siblings, relatives, teachers, and friends for standing by my side through the good and bad times. They are indeed my sunshine. I thank as well my fellow colleagues at CIC, especially Hugo, Vaibhavi, Yan Li, Amir, Abdullah, Elaheh, Xi Chen, Alina, Samaneh, Laya, Nasim, Iman, Rasool, and Saeed. It was a great pleasure working with them, and I appreciate their ideas and good humour. I would like to thank all the people at the Faculty of Computer Science who help directly or indirectly, especially to Dr. Lashkari, Dr. Al-Hadidi, Dr. Mamun, Dr. Evans, the system administrator (Hamidreza, John, and Ivan), the Post-Docs (Ratinder and Marco), the Co-Op Students (Riley and Merghan), and the administration staffs (Ali, Pamela, Jodi, and Brenda).

Last but not least, I graciously acknowledge the funding from the International Islamic University Malaysia (IIUM) and Malaysia's Ministry of Education for granting me this opportunity to pursue higher education.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Android financial malware exists because information has value now

— Anonymous

The quote above sums up the entire reason why financial malware exists. Information about users and their activities has value and naturally, money is a motivating factor. This motivation is reflected in today's tendencies observed in the mobile malware domain. Mobile malware such as viruses, trojan horses, and worms have emerged as the fastest growing threat in the digital world. This malware exhibits malicious behavior targeting mobile phones without the user's consent by adding malicious code into a smartphone's software system. According to the *2018 McAfee Threats Report*, mobile malware has doubled in the third quarter of 2017 where over 16 million malware samples were detected by McAfee Labs. The report estimated that a single banking malware family known as *Android LokiBot*, which has targeted more than 100 financial institutions worldwide, has generated close to $2 million in revenue. What is more, the report highlighted that the number of mobile threat families and variants in the McAfee sample database today is more than 4000 [69]. Mobile malware has various ways of infecting smartphones' systems and propagating themselves. Some malware can infect systems by being bundled with other programs or attached as macros to files. Some can exploit the vulnerability of the systems

through several mediums such as mobile network services, Internet access, bluetooth, and Global Positioning System (GPS). Most mobile malware gain financial profit through pick pocketing, i.e., via Short Messaging Services (SMS) and Multimedia Messaging Service (MMS), or the ability to charge premium bills via SMS or calls. Apart from that, malware is used to steal information, send SMS spam, and install other malicious applications.

Due to its popularity and openness, the Android mobile Operating System (OS) has become the most targeted platform surpassing Apple iOS, Windows Mobile, Blackberry, and Symbian. Today, Google play (previously known as Android market) has become the top competitor to Apple's App store. Unlike Apple's App store, the Google Play system is more open for uploading applications (apps) e.g., allowing users to publish applications developed with the Android software development kit (SDK) without an extensive security review before they are made available in the store. This openness has led cybercriminals to exploit the service by injecting malware apps to the market's store. Although Google is actively introducing numerous security feature enhancements to the market, the number of Android malware apps continues to increase every year. Figure 1.1 compares the number of benign to the number of new malware apps on Google Play from 2011 until 2017. Despite a slight drop in 2017, the statistic shows that malware apps can sneak into the apps store aggressively. We believe that this drop was due to the deployment of Google Play Protect (Google's built-in malware protection) in early 2017.

In recent years, Kaspersky Labs reported that Android financial malware has become **the** tool for cybercriminals to gain financial profits where 60% of Android attacks leverage financial malware [94]. Hence, the security community turned its attention to Android financial malware. According to an IBM report [82], financial malware is specifically designed to enable fraudulent transactions. Fraudsters use sophisticated techniques to steal the credentials of online banking customers, and then reuse them to take over the victim's account and perform fraudulent transactions such as transferring money to new destinations.

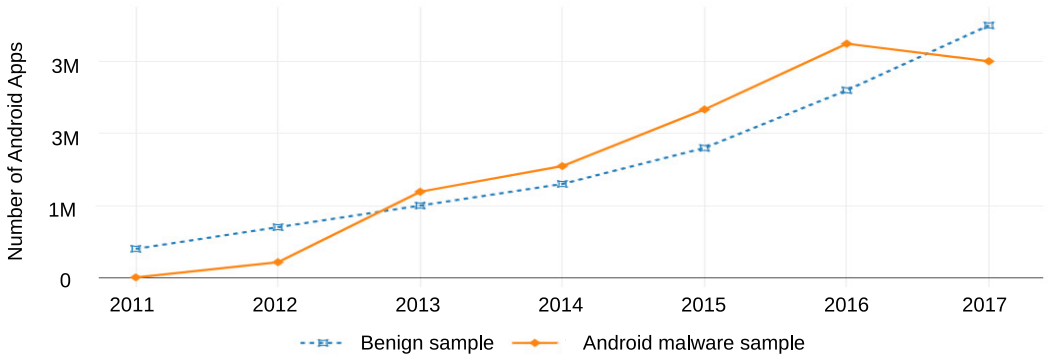

Figure 1.1: Number of apps on Google Play market vs number of new malware samples[1]

On the other hand, the European Union Agency for Network and Information Security (ENISA) defined financial malware as software specifically designed to steal credentials or perform man-in-the-middle attacks on financial applications or web services. Financial malware ranges from a keylogger collecting credit card numbers, to a more sophisticated trojan able to intercept SMS authentication codes to attack online banking applications [24]. In this research, we identify Android financial malware as the emerging trend of using specialized malware to gain information associated with financial transactions.

There are many articles, reports, and books on the techniques of financial malware, but in-depth explorations of Android financial malware are limited. In the past decade, much research has focused on detecting mobile malware; researchers have proposed various techniques such as via app-hardening systems (DNADroid [49], AppInk [162]), through app-market analysis (RiskRanker [74], SCanDroid [67], FlowDroid [35], DroidScope [153]), by continuous runtime monitoring (TaintDroid [53], MockDroid [39]), and based on the install-time checking (Kirin [55], Pyandrazzi [95]). All these studies have focused on detecting malware in general and not specifically detecting Android financial malware. Perhaps this is one of the reasons why financial malware can still bypass detection systems on the app market. Another potential reason is the disparity due to the lack of a solid understanding of modern financial malware functionality and characteristics.

[1]Data are collected from https://www.statista.com and https://www.gdatasoftware.com respectively

This thesis addresses that gap by presenting concepts, characteristics, taxonomy, and detection methods for financial malware. Our research aims to tackle this problem by focusing on both static and dynamic analyses. The proposed hybrid approach is based specifically on features derived from the app's string and on network traffic.

1.1 Motivation

As cybercrime grows in both frequency and sophistication, so do the security challenges faced by researchers. One of the primary concerns is the immaturity of the field. This is combined with the widespread adoption of mobile platforms, the growth of malware, and lack of understanding of the malware (especially financial malware). Other challenges relate to the lack of highly efficient security solutions, and to high cost of attack recovery.

First, mobile phones are a rich source of sensitive information, traditionally not available to stationary computers (e.g., location information). This creates an opportunity for new context-aware mobile malware to access and exfiltrate information typically not monitored by traditional detection systems. This situation coupled with the widespread adoption of mobile platforms for online banking creates a new lucrative target for the underground world. Secondly, the growth of malware, especially financial malware, is increasing. In fact, the risks of being hacked are now widespread among governments, industry, and individual users [57]. With the ubiquitous shift to financial gain, Android financial malware has emerged as the fastest growing threat of all attacks targeting the mobile platform or individual users. The recent report by NSS Labs concluded that 99% of current mobile malware is aimed at the Android platform and what is more, about 60% of Android attacks use financial malware [6]. There is also a lack of understanding of mobile financial malware, i.e., to determine what constitutes financial malware. The absence of a clear understanding leads to current detection systems being inadequate in detecting advanced and sophisticated mobile financial malware.

Additionally, the lack of research into the field of Android financial malware presents a challenge for advancing the field. The current detection systems emphasize Android malware threats from a high-level perspective by analyzing malware types with several categories such as viruses, worms, trojans, and botnets. This is where the financial factor comes into play; we narrow these malware threats into a specific sector (financial malware) instead of working with the general categories. For instance, sophisticated mobile banking malware today is not only capable of stealing banking account information, but also of locking the mobile devices with ransomware. This hybrid of banking malware and ransomware is an example of the new and sophisticated financial malware which cannot be effectively tackled by existing systems. The problem is that existing systems are not accurately detecting financial malware. For example, most of the current anti-viruses such as AVG, Kaspersky, and F-secure only detect financial malware as a Trojan, Worm, or Botnet which are references to the broad categories of malware. Apart from that, the recovery cost in the aftermath of cybercrime is high, often more expensive than the crime itself. One study of the cost of cybercrime for Italy found that while the actual losses were only $875 million, the recovery costs reached $8.5 billion [6].

We propose to develop a framework for detection and analysis of Android financial malware, which is capable of analyzing Android applications in a comprehensive manner: detecting malicious Android apps, classifying Android apps with respect to malware category and sub-category, and characterizing Android apps according to malware family.

1.2 Research Question and Objective

Following the problem and motivation, we pose the following research questions (RQ) that underpin the proposed study:

- RQ1: What constitutes Android financial malware?

- RQ2: Does financial malware exhibit unique characteristics that can be used to differentiate it from other malware types?
- RQ3: Does using the real smartphone for malware dynamic analysis perform better than an emulator?
- RQ4: What features should be extracted to identify financial malware?

To answer the research questions, we focus on three aspects of Android financial malware: analyzing the characteristics, creating a taxonomy, and detecting the malware. With that, we define our research objectives (RO).

- RO1: Analyze and understand the unique characteristics of Android financial malware. The extracted characteristics lay a foundation for a more accurate detection system.
- RO2: Design and develop a taxonomy for the Android financial malware domain based on the defined characteristics. This taxonomy offers a comprehensive overview of the existing financial malware and provides us an understanding of the financial malware specifics.
- RO3: Analyze and develop the best set of features that can be used to identify advanced malware. Feature selection analysis and feature rank help in improving the prediction performance of the detection algorithm.
- RO4: Design and develop the framework for Android financial malware detection based on four levels of detection capabilities: (1) malware binary detection: detecting malicious Android apps, (2) malware classification and categorization: classifying Android apps with respect to malware category, (3) classifying Android apps with respect to malware sub-category, and (4) malware characterization: characterizing Android apps according to malware family.

1.3 Contributions

The contributions of this thesis can be summarized as follows:

1. Develop a taxonomy on Android financial malware. In order to understand the behaviors, characteristics, and functionality of Android financial malware, we conduct a thorough analysis of each malware family related to financial malware. This behavioral analysis helps us to develop a novel taxonomy of Android financial malware attacks, which provides a comprehensive guide for other researchers in understanding the threat landscape of Android financial malware.

2. Present the state-of-the-art dataset of Android financial malware. We gather more than 100 thousand Android samples and generate our own Android financial malware. We filter and remove the non-financially related malware according to the defined taxonomy. To foster research in this area, we release the accumulated dataset (.apk sample and .pcap network traffic) to the research community [2]. A key to building an effective solution for Android financial malware detection is to have a comprehensive and up-to-date dataset. Our accumulated dataset combines samples from several resources such as malware security blogs, security web-services, anti-malware vendors, and other researchers.

3. Design and develop a framework called *ND-droid*, an automated *N*etwork *D*ynamic analyzer that runs on An*droid* smartphones. We use *ND-droid* for capturing the network traffic of Android financial malware. Due to some limitations of using smartphones, such as the time-consuming installation processes, the sluggish interaction with the malware, and the limited guidelines for the event processes, most of the available datasets are crafted mainly for static analysis, and those created for dynamic analysis are installed on an emulator or sandbox. Sophisticated malware can, however, bypass this technique. In this thesis, we overcome the limitations and

[2] Datasets are available at http://www.unb.ca/cic/datasets/index.html

present a systematic approach to generate the required datasets and address the need of using smartphones instead of emulators. In this regard, *ND-droid* can be used to mimick a real Android user's behavior.

4. Propose a novel combination of both static and dynamic analysis based specifically on features derived from the string literal of non-executable code (statically via reverse engineering) and network flow (dynamically on smartphones).

5. Evaluate and demonstrate the effectiveness of our framework specifically with multiple machine learning classifiers namely Naive Bayes (NB), k-Nearest Neighbor (KNN), Logistic Regression (LR), Support Vector Machine (SVM), and Random Forest (RF). We employ these common classifiers to construct the best model to identify malware at the following four levels: (1) malware binary detection: detecting malicious Android apps, (2) malware classification: classifying Android apps with respect to malware category (3) malware categorization: classifying Android apps with respect to malware sub-category, and (4) malware characterization: characterizing Android apps according to malware family.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

- *Chapter 2: Background and Literature Review* discusses the background of Android, which includes a basic understanding of Android applications and security. A comparison is also made with previous and current research development on the Android malware studies including the analysis of the datasets, a comparative evaluation of malware sandbox and virtual environments, and a review of malware detection and mitigation.

- *Chapter 3: Proposed Taxonomy* presents the taxonomy of Android financial malware, which includes a motivating example, the definition, and a classification of Android financial malware into five categories (adware, banking malware, ransomware, scareware, and SMS malware).

- *Chapter 4: Proposed Framework* explains the framework of the study, which involves the combination of the static and dynamic modules. The framework consists of four stages: collector, filter, analytics engine, and detector.

- *Chapter 5: Implementation* demonstrates the approach used in evaluating the proposed framework of Android financial malware, which is through six sets of experiments and two sets of case studies. The chapter describes the datasets, system configuration, evaluation methodology and metrics, and the implementation details of static and dynamic modules.

- *Chapter 6: Result and Discussion* describes the results of the research implementation including the experimental outcomes, case study findings, and evaluation results. The chapter presents the comparative evaluation of Android malware detection and the system performance.

- *Chapter 7: Conclusion and Future work* concludes the paper with the finding's summary, challenges and limitation of conducting the research, and some remarks on future work.

Chapter 2

Background and Literature Review

Research is to see what everybody else has seen, and to think what nobody else has thought

— Albert Szent-Gyorgyi

2.1 Overview

Since the first appearance of Android in 2008, the rising number of Android malware has drawn wide attention from both academic and industrial fields. Android security rapidly became a concern. Several studies explored the current state of the art on the security of mobile devices as listed in Table 2.1, Table 2.2, and Table 2.3. Broad overviews of mobile malware characteristics were offered by Zhou et al. [164] and Alzahrani et al. [30]. The work by Zhou et al. was one of the early studies in this domain that aimed to give researchers an understanding of mobile malware through systematic characterization of the Android malware from various aspects. At that time, one of the main concerns was a timely detection of Android malware and one of the first attempts to provide that was offered by Bose et al. [41]. The work presented a behavioral detection framework based on logical ordering of application actions. This study was quickly followed by a series

of more advanced detection approaches focused on developing detection and mitigation techniques in various areas, e.g., mobile botnet detection [62, 146, 47], detection of privacy violations (TaintDroid [53], MockDroid [39], VetDroid [115]), and security policy violations [108, 134]. Accordingly, all of their findings provide insights on the changing nature of Android malware which lead us to do research on the Android financial malware.

2.2 Background

Android Operating System (OS) is a modified version of the Linux 2.6 kernel[1] and other open source software such as the Open Handset Alliance (OHA) and Android Open Source Project (AOSP). Android was initially developed by Android Inc. and was sold to Google in 2005. It is designed primarily for mobile devices such as smartphones and tablets; low powered devices that run on battery and are full of hardware like Global Positioning System (GPS) receivers, cameras, light and orientation sensors, Wi-Fi and UMTS (3G telephony) connectivity and a touch screen.

Similar to other mobile OSs like Apple iOS, Blackberry, and WindowsMobile, Android enables applications to make use of the hardware features through abstraction and provides a defined environment for applications. These applications are written in Java and run in virtual machines. For this purpose Android features the Dalvik virtual machine (DalvikVM) which executes its own byte code. The applications for Android can be obtained from a central place called Google Play[2] (previously known as Android market). Due to this openness where anyone with a Google account can upload and download any applications, Android has become the most targeted platform by mobile malware attacks [66]. The next sections describe how Android works, including how security mechanisms are enforced.

[1] https://www.kernel.org/pub/linux/kernel/v2.6/
[2] https://play.google.com/store?hl=en

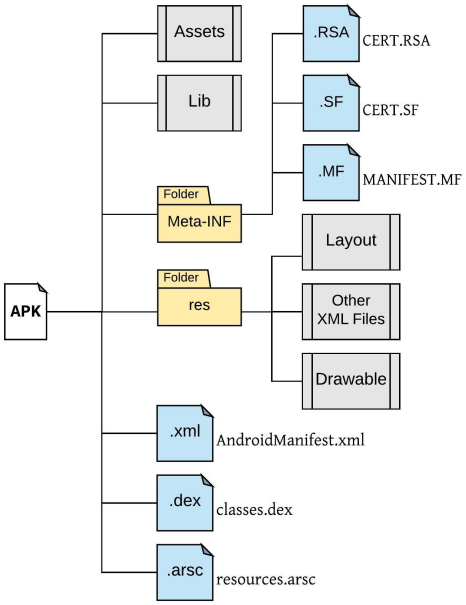

Figure 2.1: The structure of an Android Package Kit (APK)

2.2.1 Understanding Android Applications

An Android app is packaged into an Android Package Kit (APK), a zip archive file format consisting of several files and folders including the application code, resources, as well as the application manifest file as illustrated in Figure 2.1. Just like Windows (PC) systems that use an *.exe* file for software installation, Android systems use an *.apk* file for distribution and installation of mobile applications. There are two primary sources for applications:

- **Pre-installed applications**: Android includes a set of pre-installed applications as part of its open source platform or device manufacturer. e.g., phone, email, calendar, web browser, and contacts.

- **User-installed applications**: Android provides an open development environment that supports any third-party application from another market such as Yandex (Russian users) and 360 Mobile Assistant (Chinese users).

The structure of an APK file typically contains the following seven components:

1. **Assets**: The optional directory containing applications assets, which can be retrieved by *AssetManager*.

2. **Lib**: The optional directory containing compiled code. For example, native libraries that can be used through NDK (Native Development Kit).

3. **META-INF**: The directory containing several files responsible for ensuring the integrity and security of the application. It includes three files: (1) MANIFEST.MF — the manifest file which stores metadata of the application, (2) CERT.RSA — the digital certificate of the application, and (3) CERT.SF — the file containing a list of resources and SHA-1 digest. The signature of the APK is also stored in this directory.

4. **Res**: The directory containing noncompiled resources (resources not compiled into *resources.arsc*), which defines UI layouts, menus, animations, languages, sound settings, etc. Typically, the directory contains the following three sub-directories: (1) drawable, (2)layout, and (3) other extensible markup language (XML) files.

5. **AndroidManifest.xml**: An essential file in every Android application. It provides vital application details such as the unique application identifier, permissions required by the application, application version, referenced libraries, and description of several application components such as activities, services, broadcast receivers and content providers. These components [3] are discussed below:

    - Activity: An activity is the user interface component of an application that is launched using the Intents. The number of activities that can be declared within the manifest depends on the developer requirements.

[3] https://developer.Android.com/reference/Android/app/Activity.html

- Service: The service component represents one of the two tasks; either to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use. It also performs background tasks without any UI; e.g., playing an audio or downloading data from the network.
- Broadcast Receiver: This component can be used as a messaging system across apps where it listens to the Android system generated events. For instance, the Android system sends broadcasts when various system events occur, such as when the system boots up (BOOT_COMPLETED) or the device starts charging (BATTERY_CHANGED_ACTION).
- Content Provider: Content provider (also known as the data-store) is a standard interface that connects data in one process with code running in another process.

6. **classes.dex**: The *.dex* file is a core of an APK, which refers to a binary container for the code and the associated data. It includes a header containing meta-data about the executable followed by identifier lists that contain references to strings, types, prototypes, fields, methods and classes employed by the executable. The final part of the *.dex* file is the data section that contains the code and the data (i.e., URLs).

7. **resources.arsc**: A file containing pre-compiled resources such as binary XML.

2.2.2 Understanding Android Security

In this section, we aim at providing a comprehensive overview of the vulnerabilities affecting Android users, the solutions devised so far, and future research directions. Studies on Android security started in 2008 [55, 126], and the number of papers in this field has grown. Most of the studies reported on the current state of the art on security of mobile devices [36, 38, 52, 54, 60, 63]. Some of them focused on threats models targeting mobile devices, while others concentrated on vulnerabilities of a specific platform.

Significantly, Mansour et al. [28] stated that the majority of security issues affecting Android systems are caused by the apps rather than the Android OS itself, which has led to a huge number of mobile malware attacks. For instance, modern malware is designed with mutation characteristics, namely polymorphism and metamorphism, which caused an enormous growth in the number of variants of malware samples [155]. To tackle this issue, the security team of the official Android market known as *Google Play Protect* has consistently improved their security features in order to protect Android users. Figure 2.2 shows the timeline of the most relevant Android security features offered by the team from 2011 until 2017.

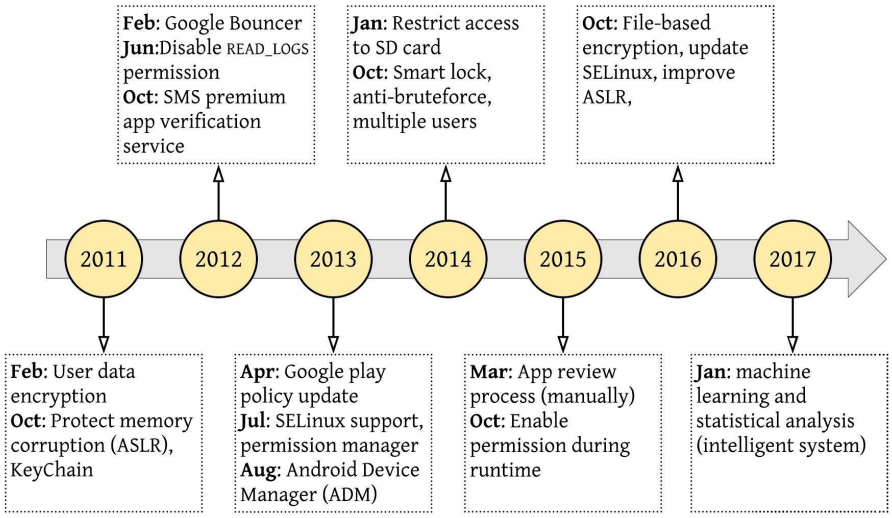

Figure 2.2: Evolution of Android security features from 2011 to 2017[4]

The first version of Android (Alpha Android 1.0) in 2008 included some basic security mechanisms such as (1) Discretionary Access Control (DAC), which controls file access through process ownership, (2) Mandatory Access Control (MAC) permission model, which protects sensitive interfaces, (3) application sandbox, which isolates apps from system resources, and from the execution environment of other applications running in the system, and (4) application signing, where applications are digitally signed with a cryp-

[4]Data are collected from Google blogs: https://source.Android.com/security/

tographic signature in order to establish trust relationships among the application and to verify their origin. Furthermore in 2012, Google introduced *Bouncer*, a dynamic analysis sandboxed environment that acts as an online security scanner. *Bouncer* vets the third party developer apps in order to thwart any malware from entering Google Play [20].

However, with more advanced and sophisticated malware, cybercriminals were gradually able to bypass those security mechanisms. This never-ending battle between cybersecurity organization and cybercriminals has always been like a game of cat and mouse. Significantly, in 2017, Google started implementing the machine learning approach (known as *Play Protect* [5]) to help detect and classify mobile threats automatically. Machine learning consists of training a computer algorithm to recognize malware behavior by giving the learning system hundreds of thousands of examples of malware behaviors. A machine-learning approach is essential for the computer security community especially when dealing with a massive number of malware on a daily basis.

In this thesis, we employ a similar approach of using machine learning in detecting malware. In addition, we expand the machine learning capabilities by exploring different techniques and leveraging new knowledge: we develop four levels of the detection framework related to Android financial malware that are capable of detecting the malware, its category, sub-category, and family. The four levels are level 1: binary detection, level 2: category classification, level 3: sub-category classification, and level 4: family characterization.

2.2.3 Understanding Android Malware

Android Malware. Mobile malware refers to any unwanted applications that are used to perform unauthorized and harmful activities on mobile devices [28]. Mobile malware specifically on Android gathers all valuable information and transmits it to remote servers by abusing apps; manipulating devices; and stealing sensitive data including passwords,

[5]https://www.android.com/play-protect/

credit card numbers, social security number, deleted files, bank account information, recent files, contacts, text messages, phone calls log, location, and visited sites. In addition, the attackers also exploit different vulnerabilities such as App-level privilege escalation and kernel-level vulnerabilities to gain root privileges. Although several works addressed these issues, the effective analysis of Android apps for malware detection still remains an open problem [54, 120, 164].

The main goal of apps analysis is to investigate if the app is benign or malicious. The analysis of benign apps, i.e., apps that do not exhibit malicious activities, may reveal vulnerabilities that might be exploited for malicious activities. Ahmadi [28] highlighted that typically these vulnerabilities are caused by developers who neglect the security aspects while developing the apps. These apps are usually referred to as Potential Harmful Applications (PHA). In this case, code review analysis is usually performed to discover unwanted vulnerabilities. On the other hand, the analysis of malicious apps, i.e., apps that exhibit malicious behaviors, may reveal the types of vulnerabilities employed by the attackers. Significantly, different levels of information can be extracted from apps when conducting in-depth analysis (Figure 2.3) including device, application, network, and OS levels. Some of the examples are monitoring sensors, speaker, and battery in the device layer; extracting various structures such as code (string, bytecode, opcode, Application Programming Interface (API)) and manifest (permission, intent, component) in the application level; monitoring HTTP traffic and analyzing network flow in the network level; and considering system calls and system metadata in the OS level.

Android Financial Malware. Since 2014, most of the security reports [16, 82, 14] have referred to Android financial malware as banking fraud. However, modern Android financial malware ranges from keyloggers to spyware to ransomware and botnets. For example, an advanced Android botnet called *Zitmo* is not only capable of stealing financial information but also launching of a banking malware attack. Similarly, *Svpeng* banking malware has ransomware in its arsenal.

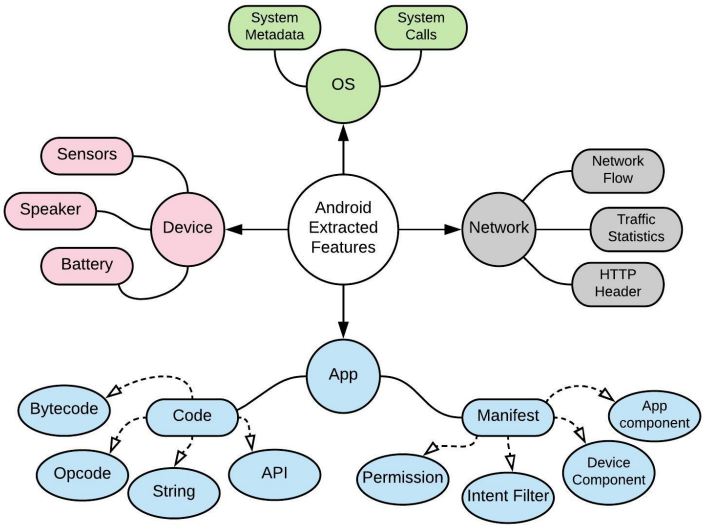

Figure 2.3: Classification of Android features [28]

The lack of a unified vocabulary and inconsistent understanding of such malware among security researchers led us to define the category of Android financial malware. Although the majority of existing studies in the field attempt to provide some classification, they often contradict each other in defining various attack types related to Android financial malware, resorting to inconsistent terminology and vague descriptions of any given attack types (Figure 2.4). For instance, a report by Kaspersky refers to *trojan SMS, trojan banker*, and *ransomware* as Android financial malware [16]. Sophos classifies mobile money making schemes into *premium-rate SMS, banking malware, ransomware, pay-per-click fraud, social media spam,* and *fake security software* [14].

On the other hand, several existing studies resort to a generic term to categorize all types of financial malware. For instance, IBM refers to the attack types of financial malware as *fraudulent transactions* [82]. This lack of unified vocabulary reveals the need for a comprehensive understanding of the existing Android financial malware. A unified terminology is a necessary foundation for the advanced development of an effective defense mechanism against these types of attacks.

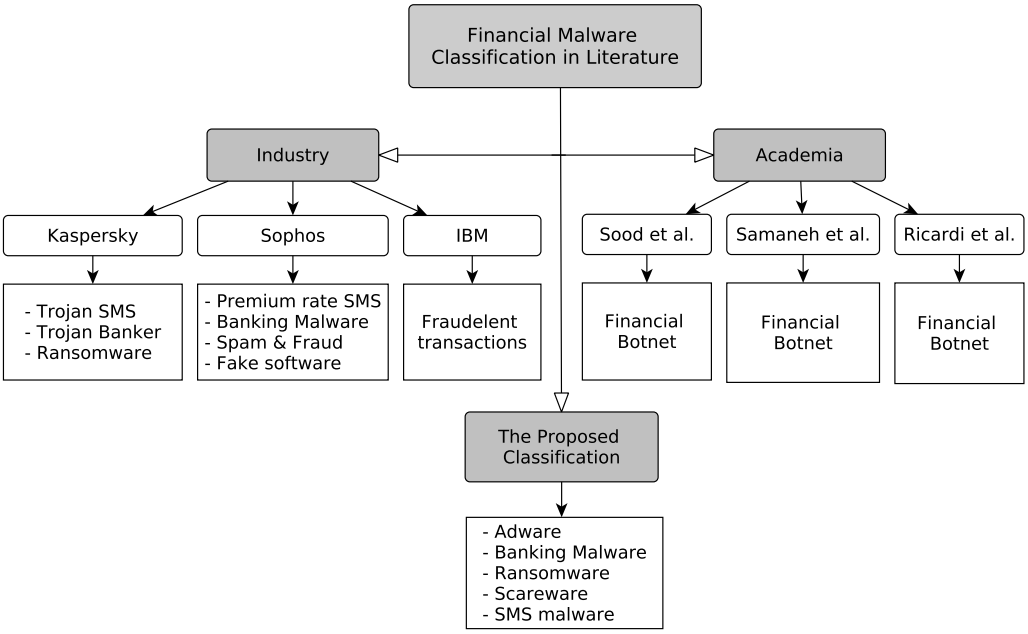

Figure 2.4: Studies of Android financial malware

For that reason, we categorized Android financial malware into the following classes: adware, banking malware, ransomware, scareware, and SMS malware. A further explanation of each category is presented in the next chapter (Chapter 3: Proposed Taxonomy).

2.3 Literature Review

Android Malware. Mobile malware was almost non-existent before the official release of the Android platform in 2008, and those few studies [44, 96, 41, 86, 81] that were conducted focused on other platforms popular at that time. In fact, the first computer worm that infected mobile phones was targeting Symbian. With the rapid development of mobile platforms and the increase in the number of mobile threats, the number of studies in the field of mobile malware, specifically Android malware, has been steadily increasing. In this thesis, we review several areas related to our work, such as Android malware, Android financial malware, and malware targeting other mobile platforms such as Windows mobile, iOS, and Blackberry.

Table 2.1: Studies focusing on Android malware detection

Year	Authors	Publication Title	Focus Area	Detection Technique
2015	Alzahrani et al. [29]	SMS mobile botnet detection using a multi-agent system	SMS detection	Signature-based and anomaly based
2013	Wang et al. [148]	What you see predicts what you get; Lightweight agent-based malware detection	Malware detection on SMS, MMS	Agent-based
2013	Badrul et al. [62]	A study of machine learning classifiers for anomaly-based mobile botnet detection	Mobile botnet detection	KNN, Naïve Bayes, DT, MLP, SVM
2013	Choi et al. [47]	Detection of mobile botnet using VPN	Mobile botnet detection	Whitelist and signature
2012	Hua et al. [78]	A SMS mobile botnet using flooding algorithm	SMS Detection	Botnet topologies
2012	Dini et al. [51]	MADAM: A multi-level anomaly detector for android malware	Outgoing SMS detection	Kernel and user level monitoring
2012	Nguyen & Pan [110]	Detecting SMS-based control commands in a botnet from infected android devices	SMS detection	Android radio logs
2012	Zeng et al. [156]	Design of SMS C&C and P2P structured mobile botnet	SMS,P2P detection& mitigation	Host-based
2010	Mulliner et al. [105]	Rise of the ibots; owning a telco network	P2P, SMS, SMS-HTTP detection	Tree topology network with iPhone modem
2010	Vural et al. [146]	Mobile botnet detection using network forensics	Mobile botnet detection	Fuzzy technique

With the rapid advancement of Android devices, researchers have focused their attention on Android malware. Broad overviews of mobile malware characteristics were offered by Alzahrani et al. [30] and Zhou et al. [164]. The work by Zhou et al. was one of the early studies in this domain that aimed to give researchers an understanding of mobile malware through systematic characterization of the Android malware from various aspects. At that time one of the main concerns was a timely detection of Android malware and one of the first attempts to provide that was offered by Bose et al. [41]. The work presented a behavioral detection framework based on logical ordering of application actions. This study was quickly followed by a series of more advanced detection approaches focused on developing detection and mitigation techniques in various areas, e.g., mobile botnet detection [78, 156, 146, 47, 105], detection of privacy violations (TaintDroid [53], MockDroid [39], VetDroid [157]), and security policy violations [108, 127]. These studies are listed briefly in Table 2.1, Table 2.2, and Table 2.3.

Table 2.2: Studies Focusing on Android Malware Mitigation

Year	Authors	Publication Title	Focus Area	Mitigation Technique
2012	Conti et al. [48]	CRêPE: A System for Enforcing Fine-Grained Context-Related Policies on Android	Android policy enforcing	Enforce fine-grained context-related policies that can be set by both the phone users and the authorized third parties
2010	Nauman et al. [109]	Apex: Extending android permission model and enforcement with user-defined runtime constraints	Android permission framework	Propose a comprehensive and user-centric extension to the Android permission framework
2010	Ongtang et al. [111]	Porscha: Policy oriented secure content handling in android	Android content protection framework	Enables content sources to express security policies
2010	Shabtai et al. [130]	Automated static code analysis for classifying android applications using machine learning	Android application characteristics	Apply the machine learning technique to differentiate the characteristics of applications
2009	Fuchs et al. [67]	SCanDroid: Automated security certification of Android applications	Android security certification	Perform information flow analysis on applications
2009	Ongtang et al. [112]	Saint: Semantically rich application-centric security in android	Android security architecture	Define install-time permission granting policies and runtime policies

Table 2.3: Studies Focusing on Malware Behavioral Analysis

Year	Authors	Publication Title	Focus Area	Analysis Technique
2015	Ibrahim et al. [83]	Analysis and setection of the Zeus botnet crimeware	Computer-based behavioral analysis	Based on the life cycle of Zeus botnet, its attack behavior, topology and technology
2014	Tajalizadehkhoob et al. [137]	Why them? Extracting intelligence about target selection from Zeus financial malware	Computer-based behavioral analysis	Based on the instructions sent to machines infected with Zeus malware
2013	Jin-Hyuk Jung et al. [90]	Repackaging attack on android banking applications and its countermeasures	Android behavioral analysis	Based on Android banking apps in Korea (Android market or the third-party market)
2012	Zhou et al. [164]	Dissecting android malware: characterization and evolution	Android behavioral analysis	Based on various aspects: unique characteristics, installation methods, activation mechanisms, malicious payloads
2010	Riccardi et al. [119]	A framework for financial botnet analysis	Computer-based behavioral analysis	Based on a novel architecture that has been validated by one of the biggest savings banks in Spain
2008	Bose et al. [41]	Behavioral detection of malware on mobile handsets	Android behavioral analysis	Based on a key observation of the logical ordering of an application's actions

Android Financial Malware. Today, generic Android malware has quickly evolved, becoming more and more focused on extracting profits, which has led to Android financial malware. Although this has already become a real concern for industry [82, 57, 103, 143, 77, 71, 128], there exist only a few studies of Android financial malware on the academic side. In this section, we look into all categories related to Android financial malware i.e., adware, banking malware, ransomware, scareware, and SMS malware.

Adware. The first study of adware was conducted by Erturk [56], where he analyzed the behavior and intent of the types of privacy-invasive Android adware. He also reviewed the Android mobile operating system security and addressed the broader issue as to the pros and cons of an open source operating system in terms of security and privacy. This was followed by another study [84] that proposed a system based on machine learning algorithms for detecting adware in mobile devices. The system employed static and dynamic analysis with the real-time classification approach. A detection rate of 97% was achieved and verified by third party evaluators.

Banking malware. The first work related to banking malware was presented by Jung et al. [90], which tested some of the major Android-based banking apps to verify whether a money transfer could be made to an unintended recipient through a repackaging attack. The experimental results showed that this repackaging attack is possible without having to illegally obtain any of the senders' personal information, such as the senders' public key certificate, the password to their bank account, or their security card. Another work [117] has analyzed the behavior of Android banking malware family called *BadAccents*. The authors described in detail the techniques this malware family uses and countered them with current state-of-the-art static and dynamic code-analysis techniques for Android applications.

Ransomware. First, Visaggio et al. [104] employed a model checking approach to identify the malicious payload in Android ransomware. They analyzed Java Byte Code to construct Formal Models and formulated logic rules to detect real world dataset ran-

somware samples. The authors showed that Hidden Markov Model (HMM) and Structural Entropy can obtain up to 96% precision in ransomware detection with the LADTree classification algorithm. Next, Yang et al. [154] introduced a theoretical static and dynamic based solution. They employed static features such as permissions and API sequences for the first layer and a dynamic behaviour monitoring for the second layer. Another work exploring the Android ransomware detection is HelDroid [33], where it utilized a text classifier that applies linguistic features to detect the threatening text and locking and encryption capabilities of ransomware. However, the solution to detect encryption is limited to the Android well-defined API and a malware author could easily evade the system by using native code. Moreover, R-PackDroid [102] demonstrated the possibility of detecting ransomware by only extracting information from system API packages. Results show that R-PackDroid characterized applications without requiring specific knowledge of the content of the ransomware samples with high precision. Another approach introduced by Song et al. [136] dynamically monitors read and write accesses to the file systems. This technique can detect and stop ransomware having abnormal CPU and I/O usage. This method can also detect modified patterns of ransomware without obtaining information about specific ransomware families.

Scareware. One of the earliest work on scareware was a study published by Shahzad and Lavesson [132] that presented a detection framework based on the application of machine learning algorithms. The framework focused on opcode sequences derived from instruction sequences of binary files. Based on their results, the Random Forest algorithm outperformed other algorithms with 97% accuracy score. A year later, they presented another paper [133], a veto-based malware detection that was able to detect malware better than majority voting. The paper analyzed a series of experiments with n-gram datasets on the recent threat. The experimental results indicated that the veto classifier has better recall than majority voting, i.e., the veto classifier reduces the number of misclassified scareware. Another work [129] presented a robust image-based detection method to iden-

tify web-based scareware attacks by capturing the screenshots of web pages to classify similar elements. In order to evaluate the robustness of the framework, they reordered the icons and changed the hue of the web pages and evaluated the method on a large-scale dataset that resulted in an equal error rate of 0.018%. The closest work to our solution was presented by Andronio et al. [33] in 2015, where they summarized the existing mobile ransomware families, describing their common characteristics and presenting an automated approach that recognizes known and unknown scareware and ransomware samples from goodware. The authors tested the method on a large dataset comprising hundreds of thousands of APKs including goodware, malware, scareware, and ransomware. The results exhibited nearly zero false positives and the capability of recognizing unknown ransomware samples. In contrast with our work, we defined scareware as a specific category of malware and not part of ransomware. We distinguish two types of scareware: fake software and fake service. Recently in 2017, Hamid et al. [76] presented an Android malware classification approach based on K-means clustering algorithm (i.e., Random Forest) using the nine features presented by Andronio et al. [33]: Lock Detected, Text Detected, Text Score, Encryption Detected, Threat, Porn, Law, Copyright and Moneypak Android malware in detecting scareware. The clustering algorithm demonstrated good results with 98.12% accuracy and 1.88% mean absolute error. In this thesis, we adopt this approach and evaluate if the proposed nine features can also be used in detecting other categories of financial malware.

SMS malware. The first study on the proof of concept of an SMS botnet was conducted by Hua et al. [78] who presented a design of a botnet using Short Message Service (SMS) as its Command and Control (C&C) medium. The authors covered all aspects of the botnet design including stealthiness protection, topology selection, and botnet monitoring. The authors demonstrated that the proposed mobile botnet is indeed a serious threat on the security of the mobile computing environment. In addition, they explored several effective defense strategies against such botnets. In a similar approach, a work

by Hamandi et al. [75] studied the messaging design decisions which resulted in a set of vulnerabilities in the Android operating system and they demonstrated how a malware application can be built to abuse the vulnerabilities. On the other hand, Al-Zahrani and Ghorbani [29] proposed a SMS botnet detection framework that uses multi-agent technology based on observations of SMS and Android smartphone features. An adaptive hybrid model of SMS botnet detectors was developed by using a combination of signature-based and anomaly-based methods. The detection performance of the anomaly-based detection module has an average accuracy of 95% and an average false negative rate of 3.95% on applied datasets.

Apart from that, there exist several studies on computer-based financial malware, which are related to our work. Riccardi et al. [119] presented work-in-progress research aimed at creating a system for mitigating financial botnets. The introduced architecture promoted information sharing among law enforcement authorities, ISPs and financial institutions. A recent work by Samaneh et al. [137] explored the incentives and strategies of attackers by analyzing the instructions sent to machines infected with Zeus malware between 2009 and 2013. The authors highlighted that on average, code similarity is well over 90% across all Zeus versions. This suggests heavy code reuse, selling, or perhaps stealing among hackers. Another study looked at the life cycle of this Zeus botnet, its attack behavior, topology, and technology based on two versions 1.2.7.19 and 2.0.8.9 [83].

Malware on Different Platforms. In this research, we do not discuss Windows Mobile, Apple iOS, or BlackBerry malware. Although the majority of the malware targets Android, the attacks and the defense mechanisms are applicable to all mobile platforms. This has been proven true by Mylonas et al. [107] in their study on the feasibilty of malware attacks on smartphone platforms. The study showed that all examined platforms (Windows Mobile, Blackberry, Apple iOS, and Android) could be used by average developers as privacy attack vectors. For example, harvesting data from the device without the user's knowledge. Moreover, another paper in 2011 [63] investigated the incentives behind 46

pieces of iOS, Android, and Symbian malware that spread from 2009 to 2011. The investigation highlighted that mobile malware is motivated primarily by a desire to send premium-rate SMS messages and sell information, which is more targeted than desktop malware. The best example of a malware family that has targeted various mobile platforms was *Zitmo*. Its functionality consists of the ability to upload all incoming SMS messages to a remote web server, the ability to forward SMS messages from a particular phone number, as well as the ability to change the command and control (C&C) server.

2.3.1 Android Malware Dataset

As mentioned in the previous chapter, a key to building an effective solution for Android financial malware detection is to have a comprehensive and up-to-date dataset. Thus, in this section, we review several publicly available Android malware datasets spanning from 2012 to 2017.

2012 Android Malware Genome Project Dataset [164]. The Genome project was the first to create, analyze, and publicly share an Android malware dataset to the research community. In this project, Zhou et al. collected 1260 malware samples from 2010 until 2011 from Android malware vendors. They used static analysis methods to evaluate malware behavior features such as *Installation, Activation,* and *Payload* by scanning the components of the malicious source code, tracking API calls, and studying permission lists. They also installed their proposed dataset on real smartphones (Nexus One phone running Android version 2.3.7.) to examine the effectiveness of existing mobile anti-virus software. Since Android malware is spreading fast among smartphones, this dataset needs to be upgraded with new malware samples. Also, the authors did not define specific activation scenarios to activate the different malware families and monitor their abnormal behavior.

2014 The Drebin Dataset[34]. The Drebin was another approach in a related domain. The study collected and analyzed 5560 malware and 123,453 benign samples from 2010 to 2012 with 179 malware families. Arp et al. trained the Drebin dataset with machine learning classification methods through static analysis. They used the malicious source code and manifest file to extract some features such as permission lists and API calls. Similar to Genome, this dataset is outdated and needs to be ugraded with new types of malware.

2014 Korea Internet Security Agency (KISA) [88]. On the other hand, Jang et al. released their dataset with 906 malware samples composed of 13 malware families and 1,776 benign samples collected from March to December 2014. They tracked suspicious API sequences from bytecode with memory dump techniques in order to extract API call patterns for certain malicious functionality through their dataset. In order to improve the previous work conducted in Genome and Drebin, Jang et al. focused on a dynamic analysis of malware detection by using an emulator. However, the new sophisticated malware can bypass this approach with an anti-emulator capability.

2016 Korea University Dataset [87]. In another study, Woo et al. combined malware-centric attributes with intent-based features for malware detection and classification. They adopted the serial number of a certificate, suspicious API sequences, permission distribution, intent, and the usage of system commands for executing forged files in a feature vector to extract the behaviour pattern of malware through 906 malware samples and 1776 benign samples. However, many malware cannot be activated comprehensively due to the anti-emulator capability. This reveals the need to use smartphones instead of emulators.

2017 Android Malware Dataset (AMD) [149]. Recently, another study by Wei et al. presented a dataset that includes the profile information of malware. The AMD contains 24,553 samples, categorized in 135 varieties among 71 malware families ranging from

2010 to 2016. The dataset provides an up-to-date picture of the current landscape of Android malware. They gathered 405 Android malware samples of four categories within 71 families. Their systematic methodology uses a reverse-engineering methodology. They evaluated malware behaviour with analysis of malicious components based on their priorities. Since they only conducted a static approach, they did not capture any dynamic features for this proposed dataset.

2018 University College London (UCL) [139]. The latest dataset by Suarez-Tangil and Stringhini consists of 1.2 million malware samples with 1.2k families. In their analysis, they extracted the rider code (malicious component) of the malware samples by analyzing common methods of API calls for detecting the malicious behavior. However, this static approach is not as effective as dynamic ones.

The aforementioned datasets help researchers in facilitating analysis in the Android malware detection domain. They have, however, still struggled with serious drawbacks of the contents of the analysis. In general, malware does not express their actual malicious behaviour, unless they are installed on real devices. This was proven by Alzaylaee et al. [31] in their analysis of emulator vs smartphones. Most of the malware applications are sensitive to activation trigger events and need user-interaction scenarios to be triggered. Moreover, although static analysis is faster, it cannot cope with advanced malware that employs sophisticated techniques. The dynamic analysis approach is particularly effective against sophisticated techniques employed by malware.

To summarize, all of above datasets face a shortage of real-phone installation, user-interaction scenarios, and diverse categories of malware types and families. Also, their analyses focus more on static rather than dynamic approaches. To overcome this problem, we generate our own dataset with different types of malware (adware, banking malware, ransomware, scareware, and SMS malware) with a variety of families. A detailed explanation of our dataset is presented in Chapter 4.

2.3.2 Android Malware Detection

As mentioned in previous section, analyzing an Android app can be performed in two ways: static or dynamic (see Figure 2.5). Static analysis refers to any techniques that are performed without executing the apps, neither on real devices, nor in emulators or sandboxes. Thus, static analysis can be performed faster than dynamic analysis, as the latter requires an appropriate execution environment (i.e., runtime while the apps are executed) in order to extract the behavior. The following explains these techniques in detail.

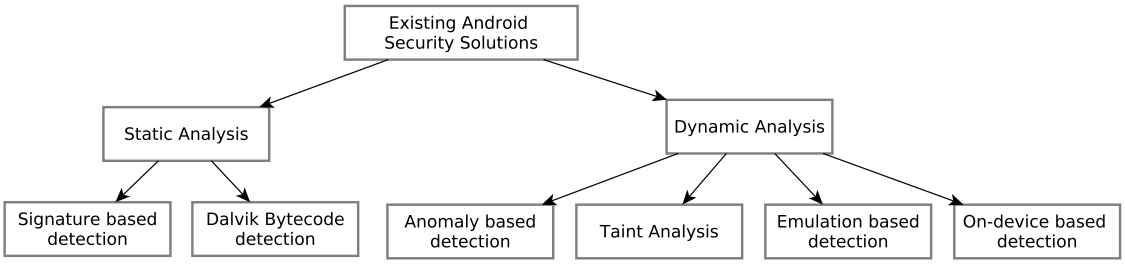

Figure 2.5: Static and dynamic analyzers

Static Analysis. This approach can be performed by disassembling its source code without execution where several features are collected from the application itself such as the code executables (string, bytecode, opcode, API) and manifest file properties (permission, intent filter, device and application components). This approach consists of three types:

1. Signature-based: this method is commonly used by anti-virus products where it extracts the semantic patterns and defines a unique signature of malware. Although this detection method is very efficient for known malware, it cannot detect unknown malware types and obfuscated and advanced malware. What is more, most malware remains undetected because of the limited signature database.

2. Dalvik Bytecode: this method helps in analyzing the app's behavior. For example, control and data flow analysis detect dangerous functionalities performed by malicious apps. Android apps are developed in Java, compiled to Java bytecode and

then translated to Dalvik byte code. In Android, Dalvik is a register-based VM that interprets the Dalvik Executable (DEX) byte code format. In this method, analysis is performed at instruction level, which consumes more resources such as power and storage space.

Dynamic Analysis. In contrast with static analysis, dynamic analysis aims to find any malicious behavior of an Android app while it is running on any platforms including emulator, sandbox, and smartphone. There are four methods of dynamic analysis:

1. Anomaly-based: this method relies on machine learning algorithms in detecting the malicious behaviors of Android apps. In this case, features extracted from known malware are used to train the model for predicting an unknown malware.

2. Taint Analysis: this method is typically used for data flow analysis and leakage detection, where it automatically labels the data, keeps track of the data, and records the data label.

3. Emulation-based: this method executes the apps in a sandbox, where it typically uses the Monkey program [6] to generate pseudo-random streams of user events such as clicks, touches, or gestures in analyzing malicious behavior of an app.

4. On-device: this method runs apps on any devices such as computers, smartphones, and tablets.

The majority of studies that employ static analysis have focused on the manifest file properties (AndroidManifest.xml), which hold the application's metadata. For instance, the application permissions contained in AndroidManifest.xml have been explored by several studies, including DroidRanger [165] and Drebin [34]. In addition, static features extracted from code executables often require additional pre-processing, and are commonly used in studies in the form of n-grams, including DroidMOSS [163] and DroidKin [73].

[6] https://developer.android.com/studio/test/monkey

Table 2.4: Limitation of static and dynamic approaches

Type	Method	Limitation	Related Work
Static	Signature-based	Cannot detect unknown malware types	[64, 61, 161, 125, 79, 122, 135]
	Dalvik bytecode	More power and memory consumption	[97, 151, 163, 37, 46]
Dynamic	Anomaly-based	Unreliable if a benign app shows same behaviors and can consume more battery and memory	[51, 42, 131, 158]
	Taint analysis	Not suitable for a real-time analysis as it slowdown system (reduce performance)	[53, 98, 124, 33]
	Emulation-based	More resource consumption	[153, 40, 123, 138, 19]
	On-device	More power and memory consumption	[106, 31, 164, 42, 85]

In contrast to our approach, we chose to leverage the string as our static feature. This is due to the hyphothesis by Richard et al. [116], where they revealed that unreferenced strings typically carry hidden information embedded in Android apps. This is proven by the example of the *GoldDream* Trojan app analyzed in their work. *GoldDream* uploaded stolen information to a remote server with the URL of *lebar.gicp.net*, which became visible only through analysis of unreferenced strings. They evaluated their framework on more than 5,000 apps from 14 different malware families and were able to classify samples with over 99% accuracy.

Similar to desktop malware, network traffic is one of the dynamic features that is useful for detecting Android malware. However, due to the lack of a large-scale malware repository and a systematic analysis of network traffic features, existing research mostly focuses on static analysis. For that reason, we propose an automated dynamic analyzer *ND-droid* for analyzing malware through network traffic analysis. Instead of using an emulator, we run *ND-droid* on smartphones in order to cope with an advanced malware evasion technique (i.e., anti-emulator).

2.3.3 Android Sandbox and Virtual Environment

One of the contributions of this thesis is the automated network dynamic analyzer called *ND-droid*. We develop *ND-droid* as a systematic approach to designing an app analysis tool in order to overcome the problems faced by researchers and to address the need to use smartphones instead of emulators.

Installation of malware apps on mobile platforms is challenging. This is due to the time-consuming installation processes within smartphones such as sluggish interaction with the malware and the limited guidelines for event processes. For that reason, most dynamic analyses are installed on an emulator or sandbox instead of a smartphone. Sophisticated and modern malware designed with mutation characteristics, namely polymorphism and metamorphism, can, however, bypass this approach. As a consequence, most of the current detection systems suffer from inaccurate evaluation and comparison due to bad and incomplete datasets captured for the analysis. Before developing *ND-droid*, we conducted a comparison study of virtual environments deployed from 2010 until 2017 to find the gap between these tools.

The *sandboxing* term was first introduced by Wahbe et al. [147] in 1993 but in a different context, i.e., software-based fault isolation. Later, Goldberg et al. in 1996 [72] used the term *sandboxing* to describe the concept of confining a helper application to a restricted virtual environment for security purposes.

There are many sandboxes that have been developed for Android applications; however, the publicly available free sandboxes only offer basic information. Table 2.5 shows an analysis of Android sandboxes including SandDroid[7], Copperdroid[8], AMAT[9], Koodous[10], DroidBox[11], and DNA-Droid[12]. Furthermore, some of the authors presented their proposed sandbox but unfortunately they did not release the sandbox for public use. This includes the Mobile Apps Assessment and Analysis System also known as MAS [141] and DroidInjector [58], a process injection-based dynamic tracking system for runtime behaviors of Android applications.

Sandboxes can be configured for a customized environment. For instance, a highly controlled environment may be seen as a specific example of virtualization.

[7]http://sanddroid.xjtu.edu.cn/
[8]http://copperdroid.isg.rhul.ac.uk/copperdroid/
[9]http://dunkelheit.com.br/amat/analysis/index en.php
[10]https://koodous.com/
[11]https://github.com/pjlantz/droidbox
[12]http://iscxm02.cs.unb.ca/

Table 2.5: Comparison of popular existing Android malware sandboxes (2010-2017)

Year	Name	Type	Static Features				Dynamic Features				User interaction	User profile
			Permission	API calls	Metadata	String	Network	Memory	API calls	System calls		
2010	AASandbox [40]	Hybrid	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
2012	Andrubis [100]*	Hybrid	✗	✓	✗	✗	✓	✗	✓	✗	✓	✗
2012	DroidBox [50]	Hybrid	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗
2012	MADAM [123]	Hybrid	✗	✓	✓	✗	✗	✗	✓	✓	✓	✗
2012	SmartDroid [160]	Hybrid	✗	✓	✗	✗	✗	✗	✓	✓	✓	✗
2012	ProfileDroid [150]	Hybrid	✓	✗	✗	✗	✓	✗	✗	✓	✓	✗
2013	AMAT [19]*	Dynamic	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗
2013	AppsPlayground [118]	Dynamic	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
2014	ForeSafe [23]*	Dynamic	✗	✗	✗	✗	✓	✗	✓	✓	✓	✗
2014	SandDroid [140]*	Dynamic	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
2015	Copperdroid [138]*	Dynamic	✓	✓	✓	✗	✗	✗	✓	✗	✓	✗
2015	Droidanalyst [59]*	Hybrid	✓	✗	✗	✗	✓	✗	✓	✓	✓	✗
2015	Koodous [21]*	Hybrid	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
2017	DroidInjector [58]	Hybrid	✓	✓	✓	✗	✗	✗	✓	✗	✗	✗
2017	DNA-Droid [32]*	Hybrid	✗	✓	✓	✗	✗	✗	✓	✗	✓	✓
2017	MAS [141]	Hybrid	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
2017	Malton [152]	Dynamic	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗

*indicates that the sandbox is a web-based analysis

Typically, by using the *sandboxing* technique, the potentially malicious activities are executed and their behaviors are recorded for further analysis. Thus, this sandbox or emulator-based dynamic analysis has been widely deployed in Android application stores. While it has been proven effective in vetting applications on a large scale, there is a doubt that *sandboxing* can execute malware behaviors effectively; emulator-based analysis can be detected and evaded by sophisticated Android malware that carry detection heuristics. By using such heuristics, the malware can check for the presence or contents of certain artifacts and infer the presence of emulators. As a result, the malware programs attempt to hide their malicious activities when being analyzed on an emulator. For this reason, countermeasures against anti-emulation are becoming increasingly important in Android malware detection.

Several techniques for anti-emulation (or anti-virtualization) have been discussed in previous work. According to Vidas & Nicolas (2014) [145], some malware applications detect the emulator through the use of Android APIs. For example, if the Telephony Manager API method *TelephonyManager.getDeviceId()* returns 000000000000000, it indicates that the run-time environment belongs to emulator rather than real phone. This is because no real phone returns 0s as the device identifier. This can be supported by an example of *Pincer* family of Android malware [31].

Moreover, Jing et al. (2014) [89] claimed that the emulator can also be detected via the networking environment which exhibits different behavior from a real phone. They exposed more than 10,000 detection heuristics based on artifacts that can detect the run-time environments and can be used in malware samples to hide the malicious activities accordingly.

To overcome this issue, several works on anti anti-emulator techniques have been presented. DroidAnalyst [59] is one of the few implementing the anti anti-analysis techniques to detect advanced and targeted malware. The dynamic analysis module of DroidAnalyst is developed to be scalable, whereby it can be configured to support the latest version of Android OS supported applications with higher API versions for which the APK are developed by third party developers. Similarly, Dynalog [14] attempts to address the problem by changing properties of the environment to emulate a real phone as much as possible and to incorporate several behaviours mimicing a real phone. Malware authors employ sophisticated fingerprinting techniques to detect a virtualized environment from the real machine. According to Amirhossein and Ghorbani (2017), human interaction is the most difficult type of fingerprinting to be mimicked. For instance, malware needs to check the mouse speed, the number of mouse clicks, or waits until the user scrolls through a document [32].

All these methods, while useful, have been shown to be insufficient to completely tackle anti-emulation [21], [22], [31]. For this reason, we proposed to conduct dynamic analysis on real smartphones instead of emulators to overcome this problem. Based on the results in Table 2.5, most of the sandboxes focus on a hybrid method and employ the common features including permission, API calls, and metadata. Nevertheless, we see some gaps in several features including string, network, memory, and system calls. Since the user interaction plays a crucial role in triggering the malicious activities of the application, we seek to investigate the user interaction and user profile that can be employed on a smartphone automatically.

2.4 Summary

In summary, there are only several work on the technical of the financial malware, and in-depth explorations of mobile financial malware especially in academia are limited. None of the previous studies offer a comprehensive understanding of Android financial malware necessary for building effective defences against it. In our study we fill this gap and propose to design and develop a framework for Android financial malware detection via machine learning that is capable of analyzing and detecting Android applications in a comprehensive manner. In contrast to the previous studies, we propose a novel combination of both static and dynamic analyses based specifically on features derived from the app's strings and network traffic. In addition, we design a systematic and automated apps analysis tool in order to address the need to use smartphones instead of emulator.

Chapter 3

Proposed Taxonomy

Research is creating new knowledge

— Neil Armstrong

3.1 Overview

A comprehensive understanding of the existing Android financial malware attacks supported by a unified terminology is necessarily required for the deployment of reliable defence mechanisms against these attacks. This chapter presents a taxonomy of Android financial malware attacks. By devising the proposed taxonomy, we intend to give researchers a better understanding of these attacks and provide a foundation for organizing research efforts within this specific field.

3.2 Motivating Example

Without knowing what constitutes mobile financial malware, detection systems are not capable of providing an accurate recognition of an advanced and sophisticated mobile financial malware. A simple illustration of that is the labelling of malware family *Zitmo*.

Table 3.1: Example of VirusTotal analysis for Banking Malware *Zitmo* (md5 hash value: 048c4a526c999539a122e39a95b7f0a1)

No	Anti-virus	Result
1	AVG	Android/Deng.FVQ
2	Ad-Aware	Android.Trojan.Zitmo.E
3	AhnLab-V3	Android-Spyware/Rehai16.d55d
4	Alibaba	A.H.Pri.Dvci
5	Antiy-AVL	Trojan[Spy:HEUR]/AndroidOS.Mekir.2
6	Arcabit	Android.Trojan.Zitmo.E
7	Avast	Android:Morcut-G [Trj]
8	Avira (no cloud)	ANDROID/Agent.EW.Gen
9	Baidu-International	Trojan.AndroidOS.Mekir.b
10	BitDefender	Android.Trojan.Zitmo.E
11	CAT-QuickHeal	Android.Mekir.A
12	Cyren	AndroidOS/GenBl.048C4A52!Olympus
13	DrWeb	Android.Spy.176.origin
14	ESET-NOD32	a variant of Android/Morcut.A
15	Emsisoft	Android.Trojan.Zitmo.E (B)
16	F-Secure	Trojan:Android/Fakeinst.NG

No	Anti-virus	Result
17	Fortinet	Android/Mekir.D!tr
18	GData	Android.Trojan.Zitmo.E
19	Ikarus	Trojan.AndroidOS.Morcut
20	Jiangmin	TrojanSpy.AndroidOS.lbq
21	K7GW	Trojan (004c7e8c1)
22	Kaspersky	HEUR:Trojan-Spy.AndroidOS.Mekir.b
23	McAfee	Artemis!048C4A526C99
24	McAfee-GW-Edition	Artemis!Trojan
25	eScan	Android.Trojan.Zitmo.E
26	NANO-Antivirus	Trojan.Android.Mekir.dubdof
27	Qihoo-360	Trojan.Android.Gen
28	Rising	APK:Trojan.Generic(AndrCity)!7.1762 [F]
29	Sophos	Andr/Spy-AEC
30	VIPRE	Trojan.AndroidOS.Generic.A
31	Zillya	Trojan.Morcut..57
32	Zoner	Trojan.AndroidOS.Morcut

Zitmo was initially analyzed by Zhou & Jiang [164] in 2008 and labelled as a banking malware. However, a quick scan of one of its family samples by the VirusTotal platform shows the disparity in labelling (Table 3.1). Based on the VirusTotal results, none of the labels indicate a banking malware category, even though some anti-viruses even use a technical malware naming convention such as *McAfee/Artemis!048C4A526C99*.

The detection results show that most of the current detection systems (20 out of 32) emphasize the Android malware threats from a high-level perspective by analyzing malware types and providing a general label (e.g., Trojan). The problem is that the existing systems are not accurately detecting financial malware. The existing signatures are only suitable for recognition of generic malware types rather than indicating its capabilities. The inconsistent labelling between different anti-virus products leads to confusion, hence affecting the mitigation and response strategy of mobile malware, which in turn leads to an increased recovery cost.

The following sections describe the proposed taxonomy in detail. The description is based on two sources: the compilation of malware reports (e.g., Fortinet [9], Kaspersky [7], Avast [8], security blogs [10, 11]) and the experimental study to investigate the malware behaviors (see the experimental results in Section 6.2).

3.3 Android Financial Malware Definition

We define an Android financial malware as *a specialized malicious software (malware) designed to direct financial profit to the fraudsters with or without the user's knowledge and consent*. This includes any reselling of victim's data, or direct transactions between the victim and the cybercriminal.

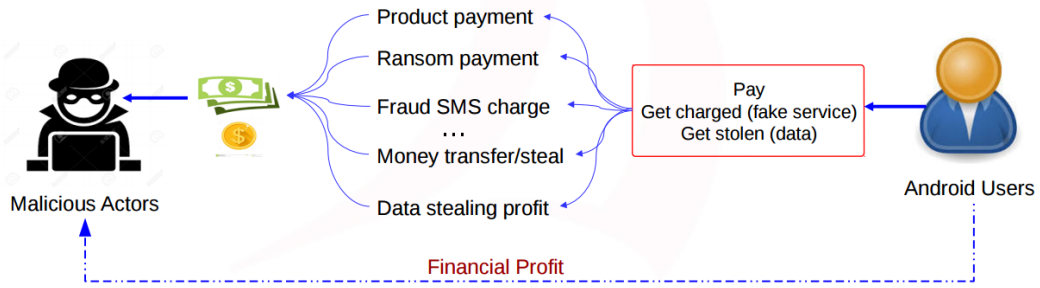

Figure 3.1: Android financial malware

As visualized in Figure 3.1, typically, the Android financial malware uses one of the following avenues to gain financial profit:

- Product payment: to persuade a user to buy fake apps or fake services.
- Ransom payment: to regain control over the mobile devices by locking the mobile screen or encrypting the personal data stored on the mobile devices.
- Fraud SMS charge: to exploit the mobile service (phone billing system) to subscribe a user to a premium-rate SMS service without the user's consent.
- Money transfer: to steal the login credentials for online banking and credit card information by replacing the authentication fields of Android apps (e.g., mobile banking apps) on the infected mobile devices.
- Data theft: to gather sensitive information by stealing personal data (banking information, social insurance number).

3.4 Android Financial Malware Classification

The defined taxonomy and classification (see Figure 3.2) are based on the smartphone's functionality and techniques the cybercriminals are using to gain their financial profit.

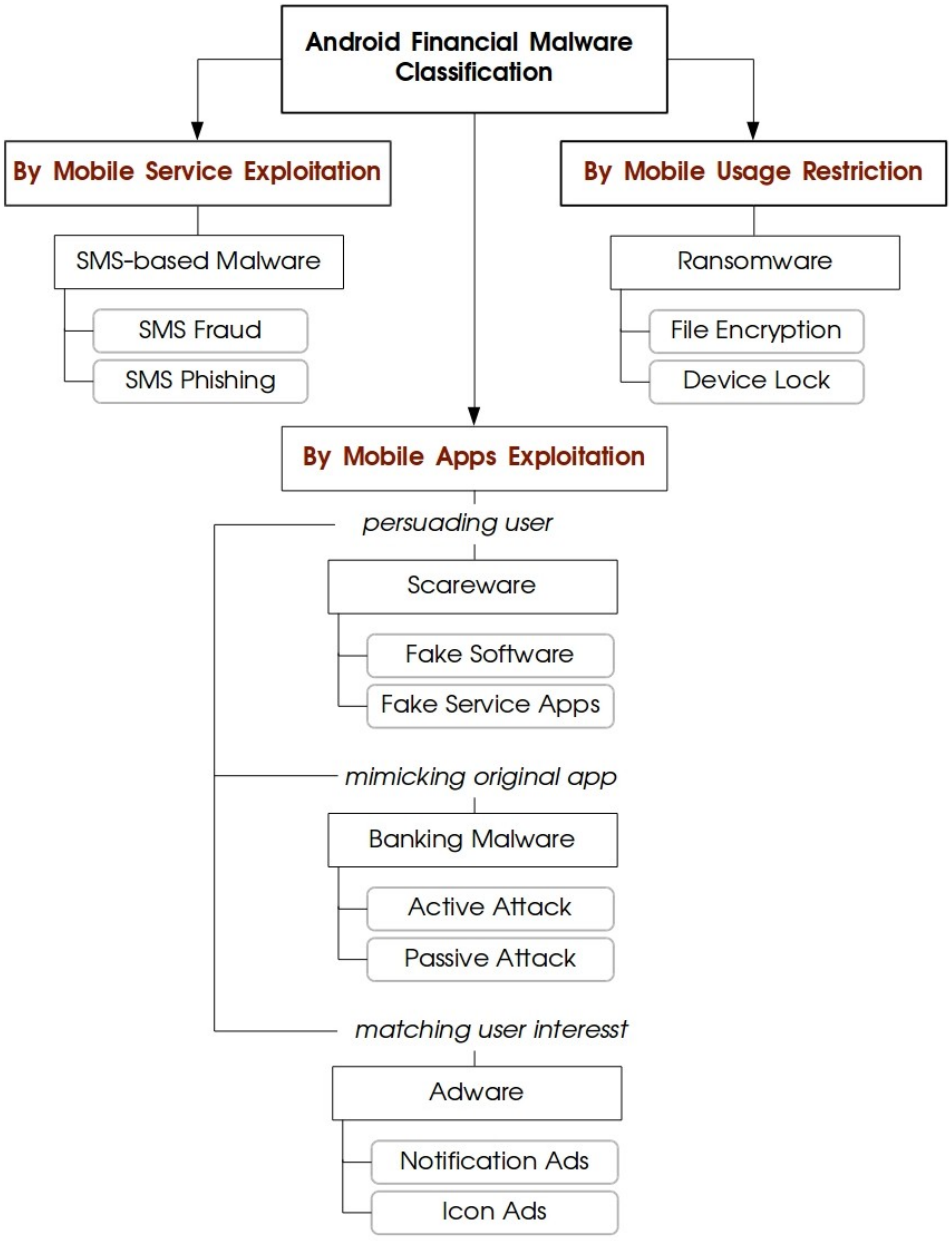

Figure 3.2: Proposed taxonomy of Android financial malware attack types

There are three elements of mobile devices which can be exploited by cybercriminals:

1. Mobile service - mobile services such as SMS, MMS, and Bluetooth can be exploited as a platform to spread malware attacks. Due to its popular monetization, we focus only on SMS and refer to this type of exploitation as *SMS malware*.

2. Mobile usage - mobile usage can be restricted by blocking the user from accessing the mobile device or some files stored in the mobile device; the user has to pay ransom in order to regain control over the mobile device. This exploitation is known as *ransomware*, which can be divided into two types: encryption-based and device-locking.

3. Mobile apps - in contrast with mobile service exploitation, mobile apps typically serve as a vehicle for rapid distribution of malware and also for the exploitation of an app's functionalities. There are three different ways of exploiting apps:

    - *by persuading the user to download infected apps*: when cybercriminals make money from malicious apps by threatening victims to download the apps or convincing them to pay some amount of money for the fake service. We refer to this technique as *scareware*.

    - *by mimicking original apps*: typically popular apps such as Uber, WhatsApp, Facebook including banking apps are used for attacks. In this research, we only focus on the financial apps, i.e. banking apps. We name this category *banking malware*.

    - *by matching user interest with the apps*: when the cybercriminals exploit the app advertisement by posing a threat to user personal information and interfere in user activity. The malware displays the related ads based on the user's interest. We call this category *adware*.

In the remainder of this section we provide details on each of the categories in the given classification, which cover various forms of financial malware including SMS malware, ransomware, banking malware, scareware, and adware.

3.4.1 Mobile service exploitation (SMS malware)

SMS Malware is financial malware that uses the SMS service as its medium of operation to intercept SMS payloads for conducting attacks. Depending on how cybercriminals gain profit through SMS service abuse, we distinguish two types of SMS malware: SMS fraud and SMS Phishing.

1. **SMS fraud.** SMS fraud refers to the exploitation of phone billing service which is called mobile premium service (sms premium-rate). This service is favoured by many legitimate service providers due to its ease of use as a mobile payment mechanism. For instance, users can order a variety of mobile content (e.g., ringtones, wallpapers, donation), receive the ordered content, and the fee will be charged to the phone bill directly. Once the transaction is completed, the aggregator (middleman) who maintains the technical service pays the service fee to the service providers. To subscribe or receive an advertised content, a user typically has to send an SMS to a given number.

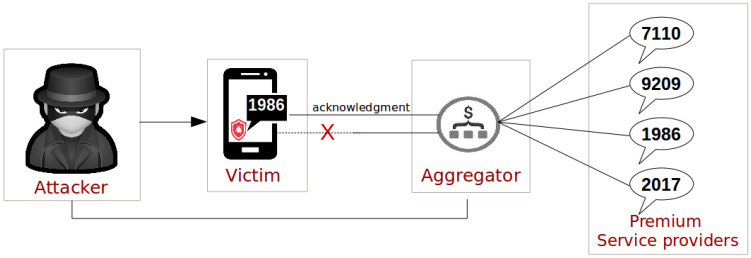

Figure 3.3: Premium-rate SMS fraud attack

Figure 3.3 illustrates how an attacker exploits the premium-rate service by subscribing a victim to a vast numbers of premium service providers silently.

In this SMS fraud, the attacker sets up its own service provider and receives money from the aggregator based on the number of subscriptions. Depending on country, some services require an acknowledgement from a user before the charge is processed, as part of the service providers procedure. However, the attackers have exploited this mechanism by intercepting the acknowledgement messages from an infected mobile device without the user's consent. As a result, the fraud is continuous and rarely caught after the first occurrence [68].

2. **SMS Phishing**. Traditionally, SMS Phishing or SMiShing refers to a form of phishing that use the social engineering technique as a method of information retrieval to acquire a user's sensitive information. For instance, a fraudster sends a victim an SMS message asking for sensitive information including credentials via a Web link or a telephone number. In the context of Android financial malware, we refer to SMS Phishing as a method of malware distribution that spreads through SMS messages and persuades a user to perform several actions, as depicted in Figure 3.4. There are two ways of distributing malware via SMS messages: spoofing and malicious apps.

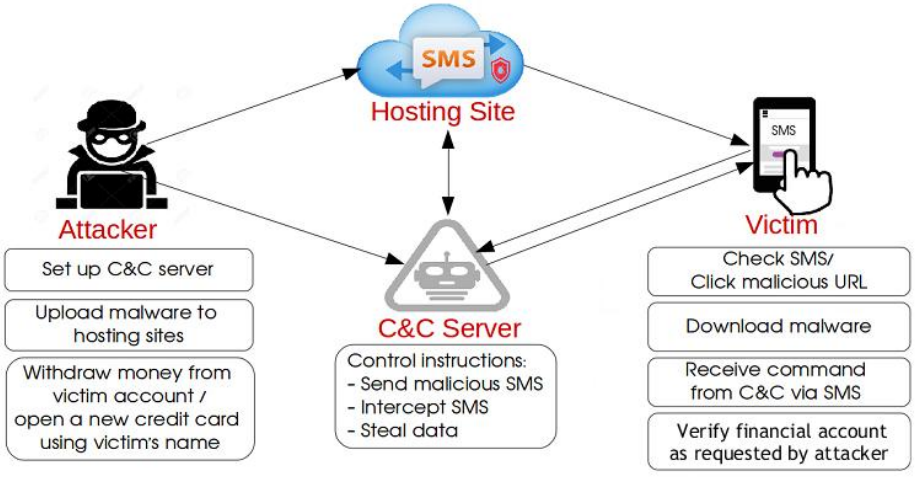

Figure 3.4: SMS Phishing scam

SMS Phishing via spoofing refers to the manipulation of the sender's information by changing the originating mobile number with different international codes or networks for the purpose of impersonating another person, company, and product. This technique leverages free SMS services that allows the attacker to freely spoof SMS messages. Such services are created mainly for users that do not own a mobile phone but need to send an SMS from a number that they have provided to the receiver in advance. However, the attackers are making use of this service as a medium of propagating malware. *MozarBot* is an example of the SMS spoofing technique employed by the Android botnet. This SMS malware is impersonating a legitimate organization in Denmark, i.e., Post Denmark. By clicking on the shortened URL in the spoof message, a victim downloads the infected Android installation application file (.apk) for *MozarBot*.

SMS Phishing via malicious apps consists of several processes: the attackers first upload malware to their hosting sites to be linked with the SMS. They use the C&C server for controlling their attack instructions, i.e., to send a malicious SMS, intercept a SMS, and steal data. Once the victim has received the SMS and visited the malicious URL, the malware is installed on the victim's device without the victim's knowledge [1]. After installation, the malware shows typical phishing behavior requesting device administrator privileges and remaining in the background to perform a series of malicious actions. Typically, these actions include the following: (1) intercept and capture all incoming and outgoing SMS messages, (2) receive command and control (C&C) commands via SMS, e.g., sending an SMS text message to every contact in the victim's phone book, and (3) steal sensitive information (e.g financial data) by intercepting SMS messages based on pre-defined keywords, such as *Pay, Check, Bank, Balance, Validation*. All obtained information is relayed to a remote C&C server. As a result, the attacker is able to withdraw money from the victim's account or open a new credit card by using the victim's personal information.

Nickyspy is an example of malware that spreads through SMS Phishing via drive-by download without the victim's knowledge. The SMS message contains the request link for an important update allegedly sent by the user's service provider. Once clicked, the link downloads the malware and executes the loader, which crashes the device and installs the actual malware components while rebooting [1].

3.4.2 Mobile usage restriction (Ransomware)

Android ransomware is inspired by desktop ransomware, which restricts usage of the infected device and demands a ransom from the infected user in order to regain control over the device or personal data. There are two variants of Android ransomware that are common today:

1. **Encryption-based ransomware** - this type of ransomware employs an encryption technique to encrypt documents and to secure the communication between the malware and its C&C server. Also, this ransomware holds a key necessary to decrypt data to the original unencrypted form. Since the security restrictions built into the Android OS prevent the malware from encrypting files stored on the device's internal memory, it encrypts data stored on external SD memory cards that typically contain personal data such as text files, pictures, and videos. The cybercriminals combine both symmetric and asymmetric encryption. Since symmetric encryption is efficient in terms of performance, the victim's files are encrypted using symmetric encryption and a session key. This session key is also encrypted by public key. Such use of asymmetric encryption is convenient as it enables the malware operator to protect only one private key that is needed for the decryption regardless of the number of victims. The malware has different ways of storing the decryption keys. Some malware families fetch the decryption key online from a C&C server once the ransom is paid. An older malware stores a key inside the malware code (e.g., SimpleLocker).

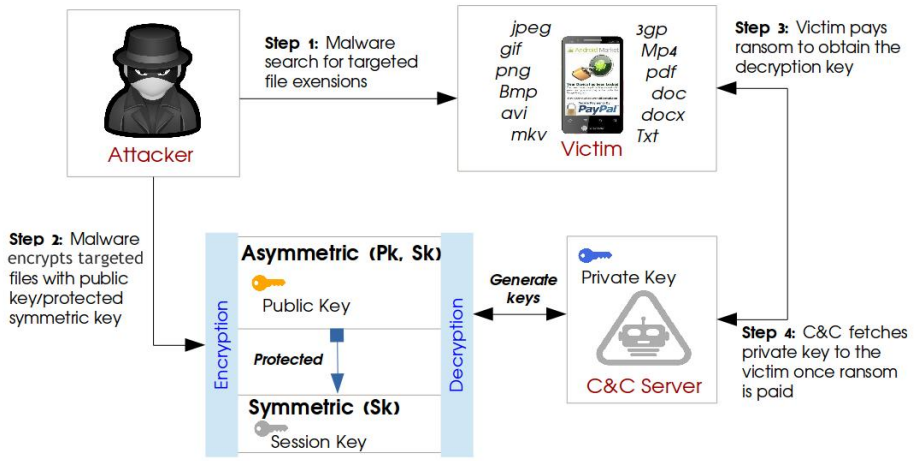

Figure 3.5: Encryption-based ransomware with dual encryption

Most frequently, the ransomware's private key is embedded in the malware or fetched from the C&C server (e.g., RansomBO). After the encryption, the symmetric key is often stored on the affected device. Figure 3.5 presents an example of an encryption-based ransomware with dual encryption that combines assymmetric and symmetric encryption. There are four steps of encryption-based ransomware: (1) File search: the malware looks for specific file extensions such as jpeg, jpg, png, bmp, gif, pdf, doc, docx, txt, avi, mkv, 3gp, mp4. (2) File encryption: the malware encrypts the targeted files via asymmetric and symmetric encryption methods. (3) Ransom Payment: In order to decrypt the files, the victim has to pay for the ransom. Once the ransom is paid, the victim receives the decryption key online via the C&C server. (4) File decryption: C&C server fetches the private key to the victim once ransom is paid.

2. **Device-locking ransomware** - this ransomware aims to block the access to the compromised device by locking the device's screen. Starting with Android 4.2, Android's lock screen supports a variety of different unlock methods as well as widgets. There are five different options of lock mechanism that have been developed by Android: *slide, face unlock, pattern, PIN, password.*

Any user selected lock mechanism will be replaced by a random PIN number set by malware. This type of ransomware is irreversible; even if a user pays the ransom, the device cannot be unlocked because the attackers do not keep track of these random PIN numbers. Without device administrator privileges or without some other form of security management solution installed, users have no effective way of regaining access to their device. The only practical way to unlock is to reset to factory defaults which would delete all the data. The attackers maintain the communication although the user has no ability to access the device (see Figure 3.6).

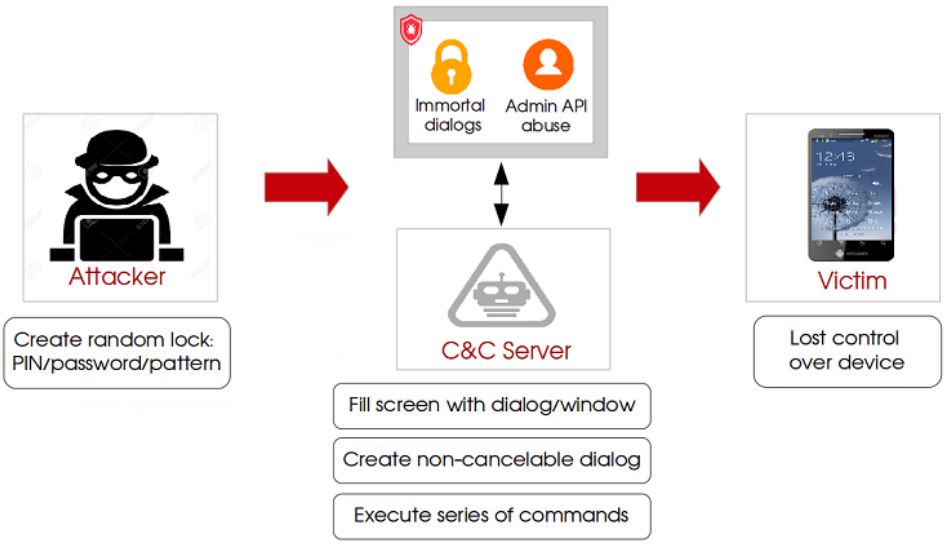

Figure 3.6: Device-locking ransomware

Typically, the attackers maintain communication with an infected locked device through the C&C server. If a C&C communication channel is established, the malware can execute commands and take over control of the infected device. Some examples of commands in ransomware, include: a) send an SMS message to phone contacts, b) steal received SMS messages and harvest contacts, c) enable or disable mobile data and Wi-Fi, and d) track a user's GPS location.

3.4.3 Mobile apps exploitation (Banking Malware, Scareware, Adware)

Banking Malware. Android banking malware refers to the specialized malware designed to gain access to the user's online banking accounts by mimicking the original banking applications or banking web interface. Based on its behavior, the Android banking malware can be categorized into two groups:

1. **Active banking malware** is designed to steal account credentials by removing the two-factor authentication system. A popular approach involves Transaction Authentication Number (TAN) theft. TAN is used by online banking services as a form of single use one-time passwords to authorize financial transactions. When the bank receives a request from the user (either via mobile or desktop), it generates the TAN and sends it via SMS to the bank customer's device. This process is intercepted by the banking Trojan malware that extracts the TAN and sends it back to the bank to gain access to the bank account to complete the one time illegal banking transaction (e.g., funds withdrawal). The users waiting for the TAN typically think that their request is not delivered and therefore request another TAN number. The visual representation of the process is shown in Figure 3.7.

2. **Passive banking malware**. In contrast to the active banking malware, the passive malware is designed to monitor the use of mobile banking apps. This type of banking Trojan disguises itself as legitimate apps (i.e. Google Play Store apps) and once installed, it will run as a service in the background to monitor events on the host device. This enables it to capture incoming SMS, monitor installed apps, and communicate with a remote server. The malware then searches for the existence of any targeted banking apps on the victim's mobile. If any results are found, it will remove and download a malicious version to replace the original apps. This malicious version displays a fake user interface asking the user to input credential information.

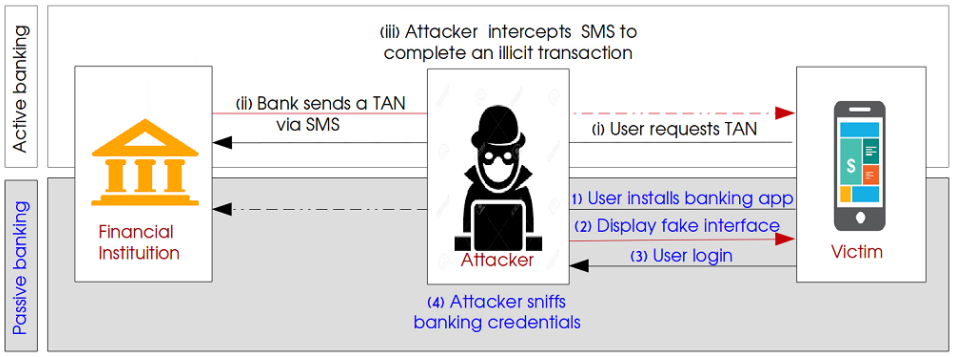

Figure 3.7: Active vs Passive banking malware attacks

The attackers then can sniff the banking credentials for illegal banking transactions. They can also capture other useful data that generate revenue for them (e.g., credit card number). Figure 3.7 shows the difference between active and passive banking malware attacks.

Scareware. Android scareware is a malicious software that poses as legitimate apps and falsely claims to detect a variety of threats on the affected mobile device (i.e. battery issues, malware threats). Similar to ransomware, the scareware exploits human emotions (panic, shock, anxiety) to manipulate users. Instead of getting money as a ransom, in a scareware attack, the cybercriminals receive money as a product payment for the malicious apps. Some scareware masks itself as a legitimate app, which makes it look legitimate and it is difficult for the detection system such as Google Bouncer[1] to identify it as scareware. Figure 3.8 shows how the attack of typical scareware works. Once the victim downloads the malware, some form of notification continues to pop-up on the device, falsely alerting the victim that the mobile device is infected and requires a security solution and protection. The damaging effects of scareware usually fall into one of two categories:

1. **Fake Service**: to deceive victims into paying for fake services. Typically, users are first offered to download apps or to buy the fake services, which are claimed to be from a trusted source (e.g., live wallpapers, photo editors, radio apps, gaming apps).

[1]http://googlemobile.blogspot.ca/2012/02/android-and-security.html

For instance, paying for fake battery upgrade or cleanup of non-existent infections on the device. The scanning process to find the false threats is free, but the cleanup process is not. If victims pay for the cleanup service, the app remains indefinitely to perform fake updates by using a Java-based pseudo-random-number generator (PRNG) that consequently leaves the device vulnerable to real malware threats.

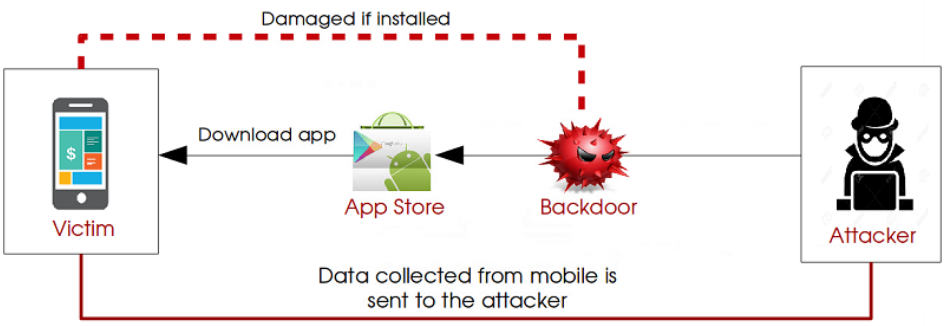

Figure 3.8: Android scareware attacks

2. **Fake Software**: to deceive victims into downloading and installing malware. For example, downloading and installing fake anti-virus (fake AV) to protect the device. Usually, after downloading and installing fake AV, the victim receives a fake progress bar with the infection result of a range of different malware randomly (e.g., Malware Tapsnake). The scareware also opens a backdoor to give attackers remote access to the device. This access remains on the device even after the malware app is removed, which leads to more attacks: signing up the victim to a premium-rate SMS, installing an advanced Mobile Remote Access Trojan (mRAT) to steal the victim's personal data (banking credentials and other sensitive information), taking pictures and relaying them to the attacker's server. The attack process becomes more attacker-friendly as the victim believes that the malware is an AV app, which causes victims to grant the malicious apps all the permissions and access that it requests.

Adware. Android adware refers to the advertising material (i.e., ads) that typically hides inside the legitimate apps. e.g., *Candy Crush, Google, Facebook, Twitter* which have been

infected by malware (available on the third-party market). Similar to scareware, adware also prompts the victim to install another app in order to launch its attack. However, once installed, the adware continuously pops up ads through a third-party library even if the victim tries to force-close the apps. This is because the ad library used by the malware repeats a series of steps to keep the ads running: (a) runs code with a number of processes, (b) creates a file then locks it (each process creates another file), (c) monitors the lock status of files (if any file is unlocked, another process is generated again).

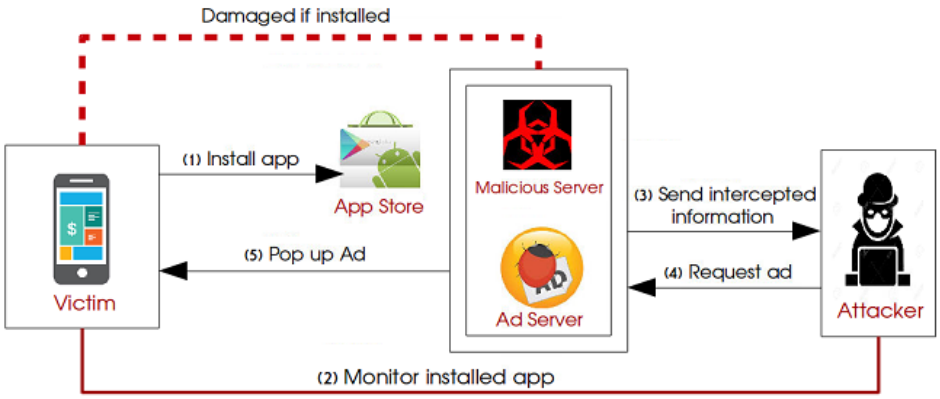

Figure 3.9: Android adware attacks

The damage effects of adware typically fall into one of three categories (see Figure 3.9):

- to display advertising content based on the user's interest (which links to a malicious site). This behavior is considered unwanted if the victim is unaware of the presence of the module of the advertising materials displayed. There are two ways of displaying ads: (1) **notification ads**: the ads deliver alerts to the mobile's notification bar when the user swipes the notification bar from the top of the screen. The notification ads technology has been used by mobile marketing firms and app developers (e.g., Facebook and Groupon) to send updates and deal alerts to their customers, but this platform has been exploited by the attackers. (2) **icon ads**: the ads are inserted onto a mobile's homescreen or desktop shortcuts and usually launch a search engine or a web service which links to a malicious site when the user touches the icon (e.g., redirecting homepages to a malicious site).

- to harvest sensitive details from the device such as the International Mobile Equipment Identity (IMEI) number, location, contacts, and sensitive information (e.g., credit card and banking information) which can link to banking malware attacks.

- to root an infected device for admin privilege escalation, which makes it difficult to remove the adware. The mobile apps typically have no access to files created by other applications, but the root privilege granted by the attacker bypasses this safeguard and exposes the infected devices to fraud and identity theft.

Unlike normal adware, financial adware depends on multiple platforms to generate revenue; instead of using the JavaScript code to click on advertisements, the financial adware inserts its code into a Google-owned mobile advertising platform (i.e. Admob) to stimulate the automatic ad clicking and pop ups in other apps' download links in the Google Store to earn additional revenue.

3.5 Summary

This chapter provides a comprehensive threat landscape of the Android financial malware according to a proposed taxonomy. For each category, we provide its definition, distinctive features and representative examples derived from both industry and academia. We believe that by having a solid understanding of the financial malware attacks followed by a unified terminology can help in the deployment of reliable defence mechanisms for financial malware attacks. This is the first attempt to organize existing research efforts in this area through a taxonomy that we hope will be extended by other researchers in the future. An evaluation of this taxonomy is presented in the *Results and Discussions* chapter.

The following chapter, Chapter 4 presents our proposed framework, which consists of the combination of the static and dynamic modules.

Chapter 4

Proposed Framework

Every 4 seconds new malware is born

— Ericka Chickowski

4.1 Overview

The quote above sums up the challenges that have been faced by malware analysts in this era. With the rapid growth of malware samples, there exist many solutions and tools that can be used by malware analysts for correlating the different signs of malicious behavior. However, the output of these tools is focusing more on the malware binary detection (or family) and not directly applicable for malware type categorization. For that reason, we present an intelligent framework for malware detection, which is capable of classifying malware based on a defined taxonomy presented in the previous chapter.

A comprehensive taxonomy combined with an intelligent framework can give analysts more time to concentrate on correlating various symptoms of malicious behavior. This combination provides a systematic overview of malware capabilities which can help analysts prioritize which malware to scrutinize for further inspection in a manual analysis phase.

This chapter outlines solution to the research problems stated in Section 1: the growth of financial malware, the immaturity of mobile malware that opened up new avenues for old attacks, the lack of research into the field of Android financial malware, the problem of existing systems (not accurately detecting financial malware), and the high recovery cost in the aftermath of cybercrime.

4.2 Conceptual Framework

A detailed illustration of our conceptual framework is outlined in Figure 4.1. It depicts the major facets of this study, as follows:

- *Stage 1: Collector* - responsible for collecting the input data of Android financial malware from various sources.
- *Stage 2: Filter* - acts as a screening filter to check the similarity of the collected data and to correlate the data with external sources and the proposed taxonomy.
- *Stage 3: Analytics engine* - a hybrid of static and dynamic modules, which involves a process of correlating and transforming data.
- *Stage 4: Detector* - consists of four levels of detection: (1) Level 1: malware binary detection, which is to distinguish the malware apps from benign apps, (2) Level 2: malware category classification, which is to classify the malware apps category, (3) Level 3: malware sub-category classification, which is to classify the malware sub-category, and (4) Level 4: malware families characterization, which is to label malware families.

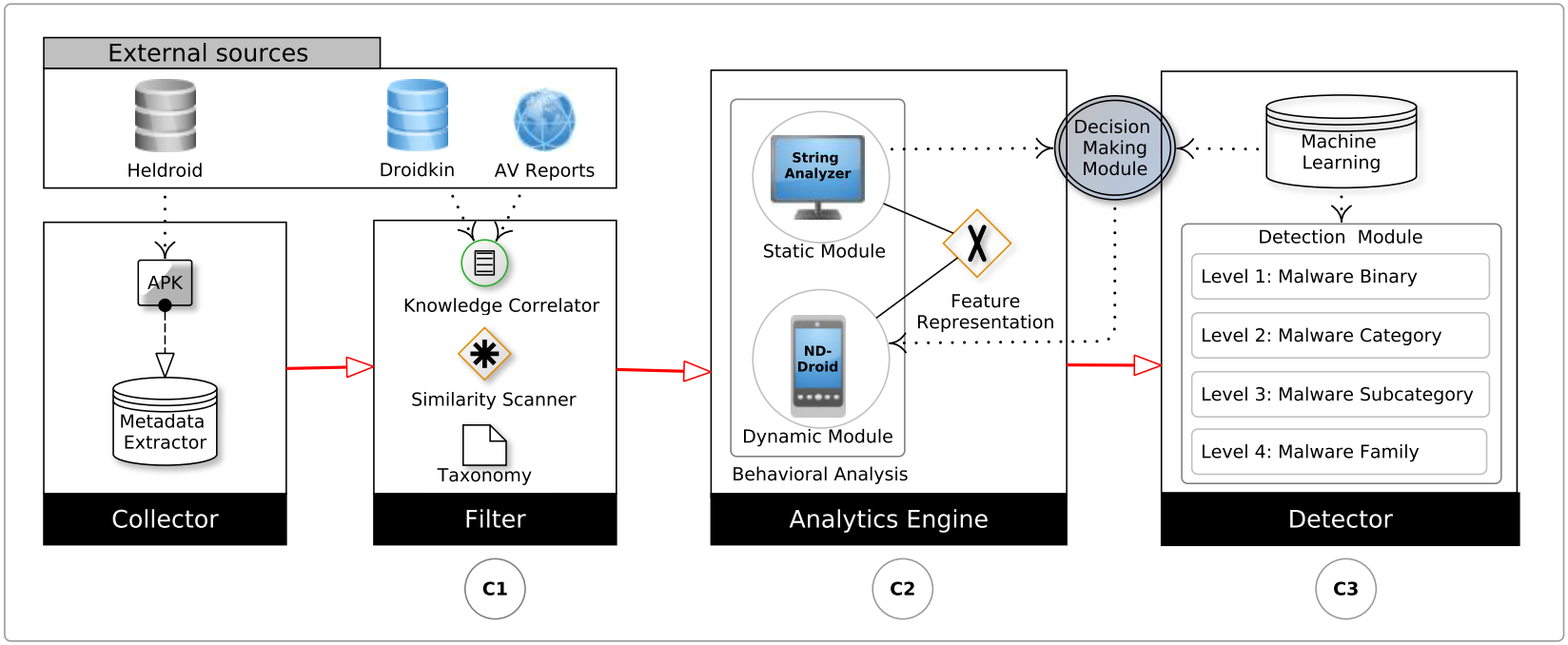

Figure 4.1: Overview of the proposed Android financial malware detection system. C1, C2, and C3 refer to the anticipated contributions of our research as described in Chapter 1 (Section 1.3)

4.2.1 Stage 1: Collector

The first component is the process of collecting samples and gathering its metadata for dataset creation. There are two types of collectors in this stage:

1. APK collector - Managing the collection of data input. The input of the framework is an Android application package i.e., .apk. This format has been used by the Android OS for distribution and installation of application software and middleware, including the malicious ones. The bash script in Appendix D.1 shows how the collector checks for the correct .apk format automatically.

2. Metadata collector - Extracting the metadata information of the collected APKs (creation date and time, version, size, etc). A tool for reverse engineering binary Android apps named *Apktool* is used in order extract all the information. The tool can decode resources to nearly original form (including resources.arsc, classes.dex, .png and XMLs) and rebuild them after making some modifications, as depicted in the command in (Appendix D.2). In addition, to extract the information of encrypted data, we employed *Heldroid*, a text classifier that applies linguistic features to detect the threatening text and locking and encryption capabilities of malware.

4.2.2 Stage 2: Filter

The second stage called ”filter” is responsible for categorizing the malware samples into several categories based on the proposed taxonomy, which is presented in Chapter 3. There are three identifiers in this stage:

- Knowledge correlator - Providing a comprehensive review of each malware variant, based on the existing reports published by the security community such as Kaspersky Lab’s cyberthreat research and reports, Symantec Internet security threat report, F-secure mobile threat report.

- Similarity scanner - Scanning the malware in order to find the relationship between the malware (twin, sibling, step sibling, cousin) by using the lightweight similarity scanner named Droidkin [73].

- Taxonomy - Classifying the malware into family and category based on the taxonomy and similarity results in the previous processes. To validate the malware grouping, a malware scanner named VirusTotal [2] is employed.

Most of the current Android malware detection systems [29, 47, 146, 110] have only categorized the malware into families during the data identification process. In our proposed framework, we extend this process and provide detailed information on financial malware behavior through specific categories such as adware, banking malware, ransomware, scareware, and sms malware. This contribution is labeled **C1** in Figure 4.1.

4.2.3 Stage 3: Analytics Engine

The "analytics engine" acts as the core component of the framework. The primary goal of this stage is to correlate all the data extracted in the previous stages. There are two components employed in this stage: behavioral analysis and feature representation.

4.2.3.1 Behavioral analysis

The first component includes a hybrid of static and dynamic modules. Static analysis (string analyzer) is a quick approach to find malicious characteristics in an application without executing them. In this phase, the APK is uncompressed and its *.XML* and *.dex* file are parsed to extract strings. On the other hand, dynamic analysis (network analyzer) involves an automated execution of an application on the physical platform, which is known as *ND-Droid*, an automated *N*etwork *D*ynamic analyzer that runs on An*droid* smartphones.

String Analyzer. The static module known as *String Analyzer* extracts all string literals and filters them to unreferenced strings, as depicted in Figure 4.2. Typically, strings can be separated into two classes: (1) All strings or referenced strings, which can be viewed as a part of functional app code. They are linked to other identifier lists such as type, prototype, field, method, and class. (2) Unreferenced strings, which can be referred as all other strings that carry textual information, e.g., string literals.

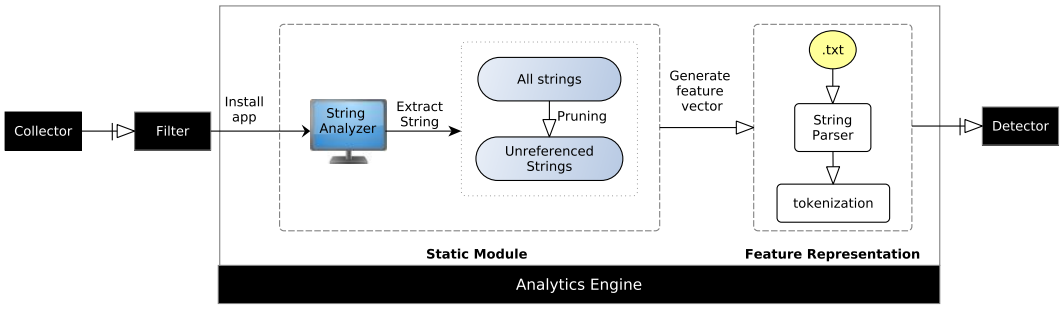

Figure 4.2: Static module: *String Analyzer*

Algorithm 1 Extracting Unreferenced Strings

procedure INPUT
  $apk \leftarrow (apk_1 ... apk_n)$ ▷ malware and benign apps
procedure OUTPUT
  $ustring(apk) \leftarrow (ustring(apk_1) ... ustring(apk_n))$ ▷ unreferenced strings
procedure METHOD
  while $apk \neq 0$ **do**
    $ustring(apk) \leftarrow getDEXstring(apk)$ ▷ get DEX strings
    foreach $DEXstring\ D_S \in apk$ **do**
      $I_{D_S} \leftarrow buildIdentifier(D_S)$ ▷ get Identifier strings
      $ustring(apk) \leftarrow ustring(apk) \bigcap I_{D_S}$ ▷ remove Identifier strings
    end for ▷ get unreferenced strings
  end do

Algorithm 1 lists the pseudocode of extracting the unreferenced strings. For each app under the analysis directory, the .dex files are extracted via a reverse engineering technique [1] to traverse all referenced strings.

[1] https://ibotpeaches.github.io/Apktool/

As mentioned in Chapter 2, we adapt the approach used by Killam et al. [116] in extracting the unreferenced strings since they claimed that the unreferenced strings typically carry hidden information embedded in Android apps. Next, a pruning process is conducted to get the unreferenced strings, by removing all strings that are referenced by the identifiers lists of the referenced string (type, prototype, field, method, and class).

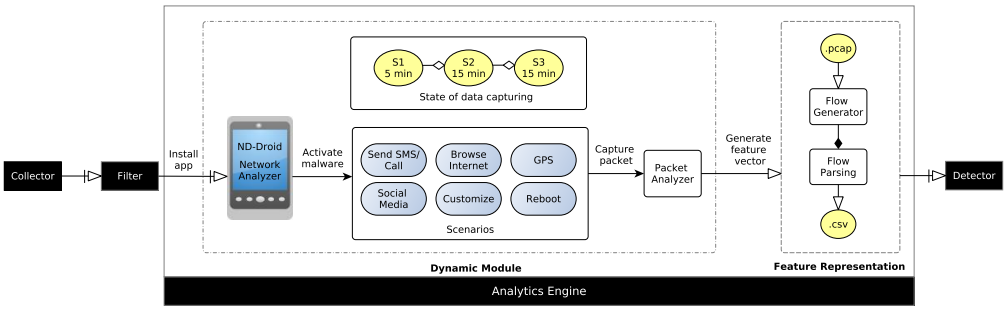

Figure 4.3: Dynamic module: *ND-Droid*

ND-Droid Network Analyzer. The dynamic module called *ND-droid* analyzes malware behavior and its properties on a real device, i.e. smartphones, automatically (see Figure 4.3). *ND-droid* includes two sub-components: state of data capturing and installation scenarios. In order to overcome the stealthiness of advanced malware, three states of data capturing are defined:

- Installation: The first state of data capturing which occurs immediately after installing malware (1-5 minutes).
- Before restart: The second state of data capturing which occurs 15 minutes before rebooting phones.
- After restart: The last state of data capturing which occurs 15 minutes after rebooting phones.

Most of advanced malware employs the evasion or transformation technique to avoid detection (i.e code permutation, register renaming, idle activation).

Thus, in this stage we consider some behaviour of malware that is only triggered over time after the restart process. Previous work in 2015 [45] claimed that more than 70% of malware generate malicious traffic in the first 5 minutes. In this framework we set the data capturing duration to 15 minutes. We believe that this duration is enough for the malware to be activated, provided that we also define a set of malware activation scenarios for each malware category. The set of malware activation scenarios consists of six elements: sending sms or calls, browsing internet, spoofing GPS, rebooting phone, activating social media, and customizing scenario option. In addition, this module also contains a user profile with an active phone number and email address.

4.2.3.2 Feature Representation

The second component focuses on a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from the raw data (i.e., strings literal and network traffic). In the static module (Figure 4.2), the tokenization operation based on words is implemented to parse the extracted unreferenced strings to the feature vector. In this phase, the stream of characters in the string literals must be broken up into distinct meaningful units or tokens before any language processing can be performed. The tokenization operation performs two types of units: token frequencies (n-gram) and token frequency-inverse document frequency (tf-idf) weights. Frequency vectors include the number of times a term (i.e., an n-gram) occurs in a given document, where in our case a "document" is the collection of strings extracted from a given app. The tf-idf assigns lower weights to terms that occur in many documents, and is calculated as follows:

$$tfidf_{i,d} = f_{i,d} * log(\frac{N}{n_i}) \tag{4.1}$$

where $f_{i,d}$ is the frequency of term i in document d, N is the total number of documents, and n_i is the number of documents that term i occurs in at least once.

Algorithm 2 Parsing Network Flow

```
1:  procedure INPUT
2:    N                                              ▷ network flows
3:  procedure OUTPUT
4:    A_N                                            ▷ aggregated network flows
5:  procedure METHOD
6:    while N ≠ 0 do
7:      A_N ← getFeatures(N)                         ▷ get features of network flows
8:      foreach Features F_N ∈ N do
9:        C_{F_N} ← buildComputation(F_N)            ▷ compute each feature
10:       A_N ← Max, Min, Mean, Stddev (C_{F_N})     ▷ define metrics for computation
11:     end for                                      ▷ get aggregated network flows
12:   end do
```

The dynamic feature representation (see Figure 4.3) is composed of two stages: flow generation and flow parsing. A flow is defined by a sequence of packets that share the following values: Source IP, Destination IP, Source Port, Destination Port, and Protocol. Algorithm 2 shows how the received flow data is parsed to generate the feature vector of network flow per sample. The netflow parser employs the aggregation modules of the four metrics: maximum (max), minimum (min), mean, and standard deviation (stddev).

4.2.4 Stage 4: Detector

The last stage, "Detector" includes two sub-components: decision making and detection module. The first sub-component determines an app to be analyzed on the run-time environment, which is based on the defined threshold score. The second sub-component includes four levels of malware detection: detecting malicious Android apps, classifying Android apps with respect to malware category and sub-category, and characterizing Android apps according to malware family. Most of the current Android malware detection systems [29, 47, 146, 110] have only detected malware with two detection levels: detecting malicious Android apps and characterizing Android apps according to malware family. In this proposed framework, we extend this layer into four levels, which contribute towards a better detection system. This contribution is labeled **C3** in Figure 4.1.

4.2.4.1 Decision-Making Module

The focus of this module is to decide which samples to analyze dynamically on run-time behavior. The module investigates a test sample by using the trained classifier from the training phase of the static classifier. The classifier labels the sample into benign, and malware. If the detection accuracy exceeds the threshold score, there is no further analysis. If the detection score is below the threshold, the static detection module forwards the sample to the dynamic module.

This module employs the average (Formula 4.2) of accuracy (based on preliminary experiments) as a performance measure for determining threshold value (more explanation is provided in Section 6.2.4).

$$Threshold_{score} = \frac{1}{n} * \sum_{i=1}^{n} X_{accuracy_i} \tag{4.2}$$

where n is the number of items (experiments) and $X_{accuracy_i}$ represents the value (accuracy) of each individual item (experiment).

4.2.4.2 Detection Module

The detection module analyzes an app in two phases. First, the *static classifier* performs a binary detection to score the maliciousness of the app between 0 (benign) and 1(malicious). To save time, the detection module captures the app's run-time behavior only for samples that fall below the threshold score of the *static classifier* (Formula 4.2).

After that, the *dynamic classifier* performs four levels of malware classification: binary, category, sub-category, and family. In this case, both static and dynamic detection employ machine learning classification techniques.

4.3 Summary

We presented an intelligent framework for malware detection, which is capable of classifying malware based on a defined taxonomy presented in Chapter 3. The detection framework is a hybrid of static and dynamic approaches and has four levels of detection capability: detecting malicious Android apps, classifying its category and sub-category, and charecterizing malware family. This combination is proposed to address the limitation of only-static and only-dynamic based techniques. The hybrid module of static and dynamic detection helped in understanding the malware phenomena and interpreting the complex behavior of malware. Importantly, a comprehensive taxonomy combined with the intelligent framework can effectively give malware analysts more time to concentrate on various symptoms of malicious behavior; this combination provides a systematic overview of malware capabilities which can help analysts prioritize which malware to scrutinize for further inspection in a manual analysis phase.

Chapter 5

Implementation

If we knew what we were doing, it would not be called research, would it?

— Albert Einstein

5.1 Overview

This chapter explains the implementation of the proposed malware detection framework, which includes details about how the dataset is built, a description of the various tools and methods used, and implementation details of the static and dynamic modules.

5.2 Dataset Description

A key to building an effective solution for Android financial malware detection is to have a comprehensive and up-to-date datasets representing the real world environment. Our detection framework employs a supervised learning algorithm for detection. The availability of a labelled dataset is one of the major requirements of any supervised learning algorithm. Hence, for the evaluation of our system, we need to have a labelled Android financial malware dataset according to our taxonomy.

Table 5.1: List of datasets used in this study

Year	Name (Android)	Types		Category		Total Apps	Description
		.apk	.pcap	Malware	Benign		
2015	D1: Botnet* [92]	✓	✗	✓	✗	1 929	The dataset consists of 14 families of botnet (2010 to 2014). that are chosen primarily due to their popularity.
2016	D2: Banking* [91]	✓	✗	✓	✓	1 073	The dataset consists of 10 families of banking malware (2010 to 2015) and benign samples (most popular banking apps) collected from Google Play.
2016	D3:AAGM* [99]	✓	✓	✓	✓	1 900	The dataset consists of 12 families of 2 malware categories: adware (250 apps), general malware (150 apps).
2017	D4: Benign*	✓	✓	✗	✓	5 330	The dataset contains legitimate apps (2015 to 2017) including the most popular apps and the most downloaded apps, which are collected from two markets: GooglePlay and Chinese market.
2017	D5: Taxonomy*	✓	✗	✓	✗	1 758	The dataset consists of 32 malware families (2010 to 2016), which consists of 5 malware categories: adware, banking malware, ransomware, scareware, and SMS malware
2017	D6: Financial Malware*	✓	✓	✓	✗	532	The dataset contains 52 malware families published from 2015 to 2016 which consists of five categories: adware, banking malware, ransomware, scareware, and SMS malware
2017	D7: Scareware*	✓	✓	✓	✓	1 650	The dataset consists of 14 families of scareware and benign samples collected from GooglePlay (2010 to 2016).
2017	D8: AMD [149]	✓	✗	✓	✗	24 650	The dataset consists of 71 malware families of several categories including adware, banking malware, scareware.
2018	D9: Network Traffic [166]	✗	✓	✓	✓	700	The dataset contains the network traffic pcap files: 100 malware samples from Drebin, 100 malware samples from Contagio blog, and 500 samples benign from Google Play.
2018	D10: MalDozer [93]	✓	✗	✓	✓	58k	The dataset consists of 32 malware families of several categories including adware, banking malware, spyware, SMS malware.
Total						97 522	

*indicates that the dataset is created by us and is available at: http://www.unb.ca/cic/datasets/index.html

For instance, a set of Android malware with the five classes (adware, banking malware, ransomware, SMS malware, scareware). However, the availability of the dataset is a major challenge for any Android malware study. In our case, there are no standard open benchmark datasets available for Android financial malware. Though many open source Android malware projects such as Android Malware Genome project [164], Maldozer [93], Contagio mobile mini-dump [142] host Android samples, the samples are not grouped by their category. Thus, we need to manually collect the Android samples belonging to different categories. Based on our dataset analysis in a previous chapter (section 2.3.1), we gathered a large collection of 97,522 Android apps from various sources, as explained in Table 5.1. After that, we conducted the pre-processing phase to filter, identify, and ensure the uniqueness of the collected samples. In this case, we discarded apps with the same hash values by using the *fdupes* tool[1]. We also analyzed the relationships of malware families in our datasets to ensure the sample's varieties via Droidkin [73], a lighweight detection of Android apps similarity.

[1]https://linux.die.net/man/1/fdupes

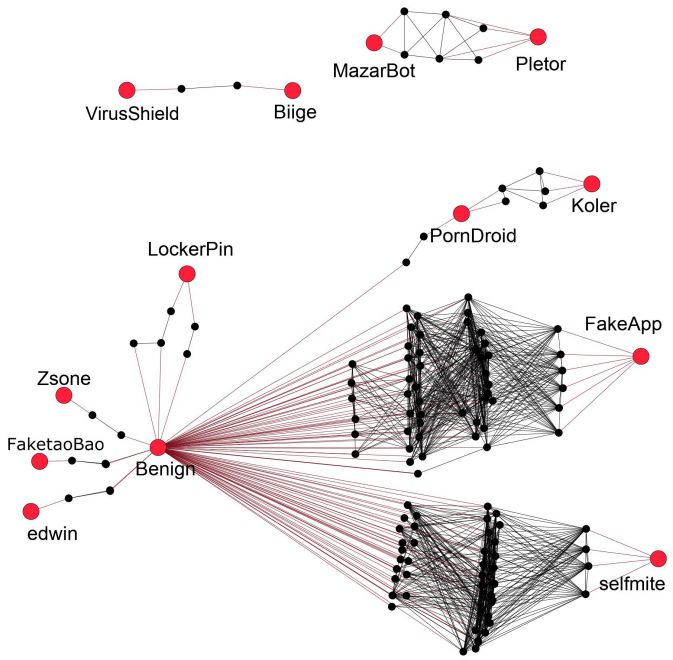

Figure 5.1: An example of Droidkin result for Android financial malware dataset. Red circles symbolize a family, black circles represent apps, and the lines indicate the relationships among the apps. If two red circles are connected, it represents the existence of a relationship.

Figure 5.1 shows an example of the Droidkin results of our Android financial malware dataset (more Droidkin results of other datasets can be found in Appendices A.1 and A.2). Droidkin depicts a weak-relationship for each family, which indicates that most of the samples are unique (with only a few similarities in a family):

1. Malware family relationships, which indicate that an app from a family maintains some similarity with an app from another family including: a) VirusShield and Biige; b) MazarBot and Pletor; c) Koler and PornDroid.

2. Benign app relationships, which indicates some similarity between malware families and benign apps: a) There are three benign apps related to Lockerpin; b) There is one app related with Zsone; c) There are two apps related with FaketaoBao; d) There are two apps related with Edwin; e) There are a few layers of apps that are tagged suspicious with Selfmite and FakeApp.

Additionaly, we also filtered the benign apps collected in our dataset. Out of 6,500 apps, 1,170 of them were removed (leaving 5, 330 apps in total) as they were flagged as suspicious or adware, by more than two antivirus (AV) products in the Virustotal web service [2]. To foster research in this area, we release the accumulated dataset (.apk sample and .pcap network traffic) to the research community [2].

5.3 System Configuration

In this section, we discuss the various tools and techniques used for the implementation of the proposed framework. We implemented our framework by using UNIX Shell scripting, Python, and the Java programming language. The static analysis implementation of the *String Analyzer* employs Apktool [3] to decode APKs and Natural Language Toolkit (NLTK) [4] to extract string literal features. In this analysis, we used various Python modules available in the scikit-learn library. Scikit-learn is a Python-based, open source library for handling various data mining and analysis tasks [114]. It provides implementations of a wide range of machine learning algorithms and functionality to generate feature vectors from string *n*-grams extracted from every APK under the analysis.

On the other hand, we used Shell scripting and the Java programming language in the dynamic analysis implementation. The *ND-Droid* network analyzer app's installation used Shell scripting to run the samples automatically. The user interaction scenario was created in the Java language and integrated via Shell scripting. Machine learning tasks including preprocessing, feature selection, training and testing phases were carried out through Scikit-learn libraries [5] and the Weka data mining tool [6]. We employed five common machine learning classifiers including k-Nearest Neighbors (kNN), Support Vector Machine (SVM), Logistic Regression (LR), Naive Bayes (NB), and Random Forest (RF).

[2]Datasets are available at http://www.unb.ca/cic/datasets/index.html
[3]https://ibotpeaches.github.io/Apktool/
[4]http://www.nltk.org/
[5]http://scikit-learn.org/
[6]https://www.cs.waikato.ac.nz/ml/weka/

5.3.1 Network Architecture

Figure 5.2 depicts the network architecture for the experimental setup and configuration of the dynamic analysis. In this analysis, we used three laptops which were connected to three smartphones. Also, these smartphones were connected to three hotspots. Table 5.2 shows the description of each devices in detail.

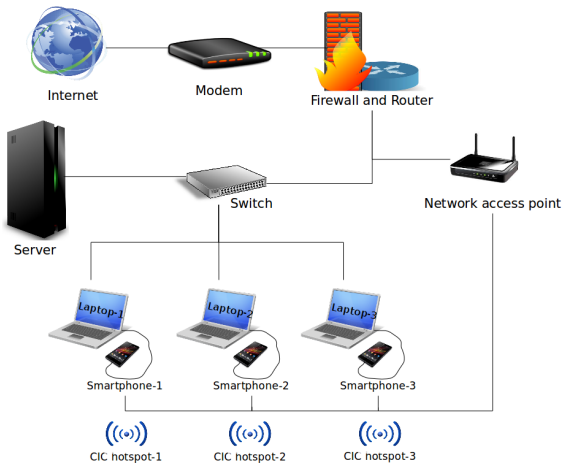

Figure 5.2: The network architecture

Table 5.2: Description of experimental devices

Device:	**Laptop-1 (Dell 14-inch)**	**Smartphone-1 (Nexus 4)**	**Laptop-2 (Dell 12-inch)**	**Smartphone-2 (Nexus 5)**	**Laptop-3 (Dell 12-inch)**	**Smartphone-3 (Nexus 5)**
OS:	64 bit ubuntu 16.04 LTS	Android 5.0 Lollipop	64 bit ubuntu 16.04 LTS	Android 6.0 Marshmallow	64 bit ubuntu 16.04 LTS	Android 6.0 Marshmallow
Memory:	8GB	2GB	16 GB	2GB	16 GB	2GB
Processor/ Phone CPU:	Intel® Core™ i5-7440HQ	1.5 GHz quad-core Krait	Intel® Core™ i7-6600U	2.26 GHz quad-core Krait 400	Intel® Core™ i7-6600U	2.26 GHz quad-core Krait 400
Graphics/ Phone GPU:	Intel® Kabylake GT2	Adreno 320	Intel® HD Graphics 520 (skylake GT2)	Adreno 330	Intel® HD Graphics 520 (skylake GT2)	Adreno 330

Figure 5.3 presents the Unified Modeling Language (UML) structural class diagram of the system's architecture. The architecture consists of four major components: collector, filter, analytics engine, and detector.

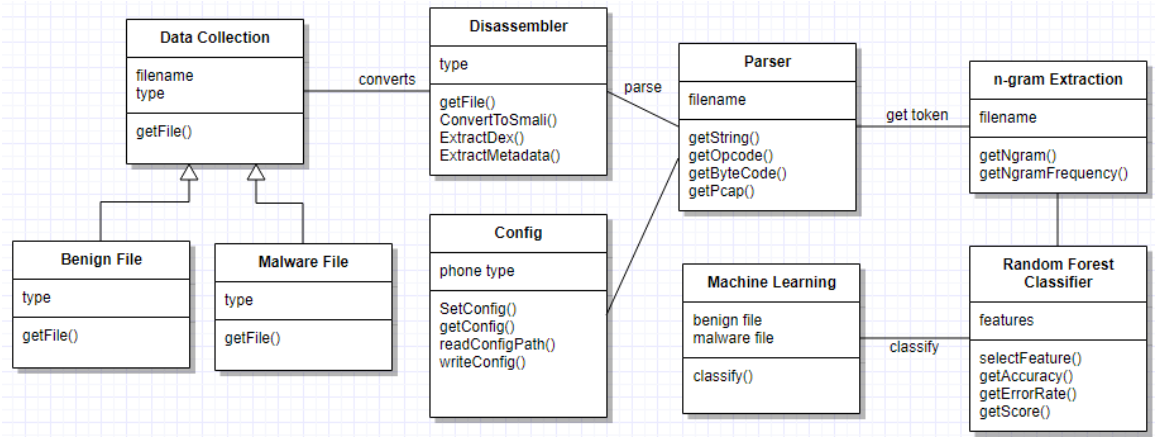

Figure 5.3: Class diagram of the framework implementation

5.3.2 Learning Parameter

The use of machine learning techniques involves careful tuning of the learning parameters. These algorithms should be calibrated with parameters that ensure maximum performance, which in our case means the maximum classification accuracy and the minimum number of false positives. To avoid over-tuning, i.e., using all samples to calibrate the system that over fit the training data, we only use 20% of the dataset for parameter tuning. We perform different rounds of experiments to decide on the optimal value for each parameter. Below is the list of parameters that need to be optimized for each classifier.

- Random Forest (RF): number of trees (set to 100 trees) and minimum number of instances per leaf (set to 1 leaf)
- Naive Bayes (NB): type of estimator (used kernel density estimator)
- k-Nearest Neighbor (kNN): number of neighbors, k (set k to 2 neighbors)
- Support Vector Machine (SVM): kernel function (used Radial basis function (RBF))

5.3.3 Evaluation methodology and metrics

In this section, we explain methodology and metrics used for evaluating the performance of the system such as *scalability*, *cross validation*, and *evaluation metrics*.

Scalability. One of the key issues in the existing malware detection techniques is the scalability. Scalability refers to the size of malware features extracted for analysis. For instance, even a simple model such as a sequence of system calls generates a number of malware features that grows proportionally to the size of malware execution traces [43]. As a result, the detection process becomes impractical as huge feature spaces make the learning process computationally intensive. In addition to the scalability of feature spaces, unacceptably high computational complexity and memory consumption are also expected to impact the practicality and efficiency of a malware detector. For example, it is reported that it took 12 to 48 hours to extract malware specifications from a network worm using a graph mining algorithm [65].

As such, practical applications of complex malware detection techniques are very limited in practice. Thus, in this thesis, we propose a novel combination of both static and dynamic analysis. In our framework, scalability is achieved through constructing feature vectors (automated analyzers and parsers). In other words, our framework can extract malware features that do not grow in proportion with the number of program samples under examination. In addition, the computation times (e.g., for training, testing, and evaluation) and memory consumption are lower as well. This makes our framework more practical and scalable.

Cross validation. Our proposed framework aims to classify an unknown APK to one of the financial malware categories from a set of categories in the given dataset. One potential approach is to divide the given dataset into two parts: training and testing.

Kalgutkar et al. [144], however, claimed that the *cross validation* approach is better than the *training and testing* approach. They highlighted that a fixed set of samples may not be effective for training the classifier, where it can produce an estimation that is biased towards the samples (if it is trained over a fixed set of samples). On the other hand, the given dataset is divided into k equal parts in the k-fold cross validation. Some of the parts are selected as a training set and others as a testing set. The instances are randomly selected to be included in folds. Although this might lead to samples of only some of the classes to be selected for training the classifier which can generate biased results, the *stratified cross validation* technique provides a solution for this problem as it preserves a proportionate amount of samples belonging to every class in each fold.

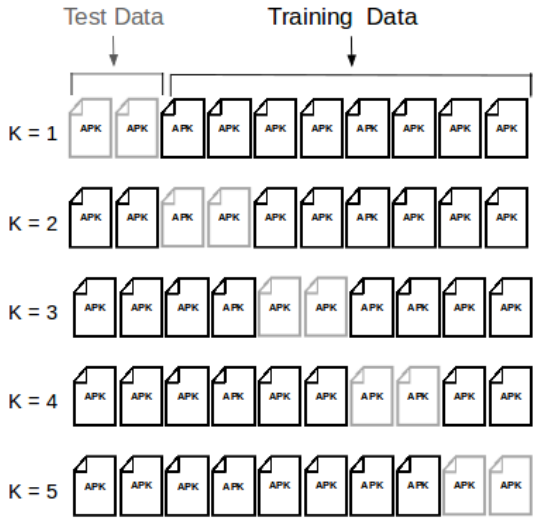

Figure 5.4: 5-fold cross validation (from [144])

Thus, we implemented a 5-times 5-fold stratified cross validation strategy for evaluating our framework. We divided the dataset into 5 parts (see Figure 5.4). We used one part for evaluating the system and the rest for training. After that, we conducted stratified cross validation experiments repeatedly for comparing performance of different kinds of strings and netflow on a single dataset. In this phase, we employed a random seed in order to ensure the repeatability of folds for each of experiments.

Evaluation metrics. Our Android malware detection framework trains a classifier over training samples. To represent output predictions of the classification model, we used a *confusion matrix*. A confusion matrix is an $N \times N$ contingency table, where N is the number of output labels. It shows the number of samples correctly and incorrectly classified by the model as compared to the actual target output values.

Table 5.3: Confusion Matrix for a binary classification problem

	Target positive	**Target negative**
System positive	True positive (t_p)	False positive (f_p)
System negative	False negative (f_n)	True negative (t_n)

Table 5.3 shows an example of the prediction results of a binary classification model with output labels as either *Positive* or *Negative* that can be represented in the form of a 2×2 *confusion matrix*. The target output categories are represented by columns; whereas, rows represent the system predictions. *True positive* refers to the number of positive samples correctly labelled as positive by the system and *true negative* represents the number of correctly predicted negative samples [144]. We calculated the following metrics (based on the confusion matrix) for evaluating the classifier performance:

- ***Accuracy***: Accuracy is calculated as the number of correctly predicted samples divided by the total number of samples. It is the proportion of correct predictions made by the classifier.

$$Accuracy = \frac{t_p + t_n}{t_p + f_p + t_n + f_n} \tag{5.1}$$

  Accuracy, however, is insufficient to evaluate the performance of the classifier as it can be misleading in the case of highly imbalanced data, where the number of samples representing a single class (*majority class*) is significantly high as compared to the other class samples (*minority class*).

Therefore, we used additional metrics to evaluate the performance of the classifier, including the *precision, recall, F_1 score, Receiver Operating Characteristic (ROC) curve,* and *False Positive Rate (FPR)*:

- ***Precision***: Precision is the number of samples correctly predicted as positive divided by the total number of samples predicted as positive in the dataset.

$$Precision = \frac{t_p}{t_p + f_p} \tag{5.2}$$

  It represents the exactness of the classifier as it represents the proportion of samples predicted as positive that are actually positive.

- ***Recall***: Recall is the number of samples correctly predicted as positive divided by the total number of actual positive samples in the dataset.

$$Recall = \frac{t_p}{t_p + f_n} \tag{5.3}$$

  It measures how accurately a classifier can predict samples belonging to a particular class. It represents the detection rate, i.e., the proportion of actual positive samples that are correctly predicted as positive.

- ***F_1 score***: F_1 score, also known as the F_1 measure, is calculated as the harmonic mean of precision and recall.

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5.4}$$

- ***FPR***: FPR is calculated as the number of false positives divided by the total number of real negative cases in the data. It measures the probability of false alarm.

$$FPR = \frac{f_p}{f_p + t_n} \tag{5.5}$$

- ***TPR***: TPR is calculated as the number of true positives divided by the total number of real positive cases in the data. It measures the probability of detection.

$$TPR = \frac{t_p}{t_p + f_n} \tag{5.6}$$

- **ROC curve**: ROC curve is defined by FPR and TPR as x and y axes, respectively, which can be used to evaluate the trade-off between true and false positive rates of classification algorithms.

5.4 Static Module Implementation

In the static module, we employed *n*-grams tokenization (sequences of words of length *n*) for text categorization. This is performed to preserve the semantics of the original strings. For example, given the sentence:

When Goodware Become Scareware.

There are three 2-gram tokens can be made:

1. *When Goodware*
2. *Goodware Become*
3. *Become Scareware.*

The tokenization used here was to split a string based on whitespace; thus in this example, the period at the end of the sentence is a part of the word *Scareware.*

5.5 Dynamic Module Implementation

To create a comprehensive scenario for data capturing on *ND-Droid*, we created a user profile for each smartphone, defined the automated user interactions, and generated the malware activation scenario:

1. *User Profile and Interaction.* Three user profiles are created for each smartphone. Each user was registered to Gmail, Facebook, Skype, and Whatsapp with a valid email address and phone number as listed in Table 5.4.

Table 5.4: User profile description

User	Account	Email/Phone No.
1 (Baly)	Gmail	cic.googl.01@gmail.com
	Facebook	cic.googl.01@gmail.com
	Skype	cic.googl.01@gmail.com
	Whatsapp	+6*******0062
2 (Ryan)	Gmail	cic.googl.02@gmail.com
	Facebook	cic.googl.02@gmail.com
	Skype	cic.googl.02@gmail.com
	Whatsapp	+6*******6315
3 (Rania)	Gmail	cic.googl.03@gmail.com
	Facebook	cic.googl.03@gmail.com
	Skype	cic.googl.03@gmail.com
	Whatsapp	+6*******2140

In addition, since some malware categories exhibit different behaviors (displaying pop-up ads), we defined the following interaction scenarios:

- Adware and Scareware: The only difference between this scenario and the other malware is that we modified the 15 minute loops slightly.

Now at the start of each loop we used a python script to grab the xml elements of the screen and search for certain keywords to click on to accept a scareware pop-up. In reality we still had to do this work manually, because many of the applications were in other languages or did not have a pop-up immediately. We also changed the loop to bring the application back to the foreground; this also re-opens the application if it was closed. These additions mean that the loop takes longer to execute and results in fewer iterations.

- SMS malware (Non-premium): The difference between this scenario and the other malware is that a different backup was used (for AV installation). The AV application was started before the malware was installed. The 15 minute loop was modified to interact with the installed AV (via a UI automator) and cause it to communicate with it's server. This was done either by initiating a scan or by attempting to update to the latest virus information. The loop was also modified to bring the malware application back to the foreground at the end of each iteration. The AV installations were as follows: AVG was installed on phone 1; Bitdefender was installed on phone 2; and Avast was installed on phone 3.

- SMS malware (premium): This is similar to the SMS malware (non-premium) scenario except it only goes through the loop once each and the script sleeps for the rest of the 15 minutes both times. The other big difference is that the phone has a SIM card in it. Because of these changes we are sending (and being charged for) 2 SMS messages per sample, assuming the malware does not block the ability of the user to send messages. We are sending these messages to the phone number of another one of our SIM cards. We also cannot make calls the same way we were doing it before because now that there is cell service we are charged for the calls even if they are to a non-existant number. Instead we call the same command without providing a phone number at all.

Similar to the Scareware scenario, this one requires user input to attempt to generate premium SMS messages.

2. *Malware Activation Scenario.* Table 5.5 shows the list of scenarios for each malware category. The scenarios aim to mimick the behavior of mobile users with the optimal user interaction events. We defined the scenario according to the taxonomy presented in Chapter 3. The following explains the scenarios in detail:

    - Sending SMS: Based on the malware behavioral analysis, we managed to gather all sensitive keywords used by the malware in order to steal personal information, as listed in Table 5.6. To automate the sending of messages on the Nexus 4, we were able to use the messaging app that came installed on the phone with the specific command (Appendix D, Listing D.3). However, with the Nexus 5 this only types out a message and starts a new line in the same message. Thus, we had to use Android Studios UI Automator to create a test apk for the messaging app that we were then able to call whenever we wanted to send a message. For consistency we used this method on the Nexus 4 as well. We only put a SIM card in the SMS malware category to activate the SMS premium-rated malware in order to minimize the damage of money loss. Thus, these messages were not actually being sent except for SMS malware. We also sent messages using WhatsApp which, unlike the previous messages, were successfully sent. Because of that, we only sent messages between our test phones. In order to send these messages, we used Android Studios UI Automator again to create a second test (see Appendix D, Figure D.1). Figure D.2 in Appendix D shows an example of the automated event or scenario (user is sending a message via Whatsapp) by one of the profiles named Baly.
    - Sending Calls: To automate phone call events we used the adb command (Appendix D, Listing D.4).

Table 5.5: User interaction scenarios

Category	Scenario	Remark
Benign	- Send Message - Make Call - Enable GPS - Browse Internet	SIM card disabled
Adware	- Send Message - Make Call - Enable GPS - Browse Internet	SIM card disabled
Banking Malware	- Send Message - Make Call - Enable GPS - Browse Internet - Click/follow popup - Save more than 10 contacts in the contact list - Save the following documents in both internal and external storage: jpeg, jpg, png, bmp, gif, pdf, doc, docx, txt, avi, mkv, 3gp, mp4 (size more than 10 KB) - Install financial apps: paypal, androidpay	SIM card enabled
Scareware	- Send Message - Make Call - Enable GPS - Browse Internet - Click/follow popup	SIM card disabled
Ransomware	- Send Message - Make Call - Enable GPS - Browse Internet - Click/follow popup - Set the four-digit PIN and lock the phone - Click/interact with any popup message - Save more than 10 contacts in the contact list - Save the following documents in both internal and external storage: jpeg, jpg, png, bmp, gif, pdf, doc, docx, txt, avi, mkv, 3gp, mp4 (size more than 10 KB)	SIM card disabled
SMS malware	- Send Message and SMS - Make Call - Enable GPS - Browse Internet - Install AV (AVG, Avast, BitDefender) - Save more than 10 contacts in the contact list	SIM card enabled

Table 5.6: Messages created based on sensitive keywords from malware

Sensitive Keyword	**SMS Text**
bank, pin, passport, validation	Adam where are you now? I want to change my bank's pin, but they ask me to bring my passport for validation
banking, balance, deposit, check	Taylor what's your banking balance? please call daddy. I want to deposit your check.
$, bank, transaction	CIMB: IBFT Transfer $300 to Rania/Maybank on 02 Jun 2017 10:02:02. Call 60362047788 if you DID NOT perform this transaction.
$, transfer, bank, TAC, expires	RHB Transfer $100 to Hong Leong Bank. TAC is 711718. Expires by Jun 07 15:20:01
TAN, expires, transaction	The mobile TAN 736592 for transaction 025787154. Expires by Sat Jun 03 15:39:14
password, banking	Hello Ryan, I have used the same password bringiton77 for my online banking, facebook, snapchat & instagram
pay, tax, HST	Hey Trump, did you pay for the tax yet? Did you have to pay the 13% HST for them?

However our phones did not have SIM cards, thus no calls were actually made. When we tested with SIM cards we modified that line to make the call without phone number (Appendix D, Listing D.5).

- GPS spoofing: We modified an open source Android application that can fake the location of the phone.[7] We made a small modification so we could run it on API 23. Then, we created a backup so we do not need to install it every time we restore the phones.

- Reboot phone: Running a factory reset after each test was not a viable solution because it requires a number of manual steps to set up the phone again. Rather than a factory reset, we decided to use a custom recovery tool (TWRP) to create a full backup that we could restore between the tests. With our Nexus phones running Android 5.1.1 (Lollipop), we were able to create a backup, and then using only ADB we were able to successfully restore the backup and reboot the phone.

[7]https://github.com/amotzte/android-mock-location-for-development

However, on our Samsung Galaxy A5 (Model number: SM-A510FD) running Android 6.0.1 (Marshmallow), we were unable to access the device with ADB while it was in recovery mode and because of this we were unable to automate restoring the backup. On our Google Pixel phone, we used TWRP 3.1.0-RC2 and whenever we tried to restore a backup the phone was wiped as if we had performed a factory reset. The team behind TWRP knew about this issue in a previous version of Google Pixel but it was supposedly fixed in 3.1.0-RC1 and 3.1.0-RC2. As an alternative to the TWRP backup we tried using ADB's built-in backup tool. However, we found many reports of people having issues with this tool and after trying it ourselves, we determined the backup did not cover as much as we needed it to. The following applications are included in the backup that we restore to: Facebook [8], Skype [9], Allo [10], Android Messages [11], WhatsApp [12], Pulse [13].

- Internet browsing (loading urls): We are able to use the browser to navigate to a given URL using the command in Appendix D, Listing D.6). If this command is run and users have multiple browsers installed, but do not have one selected as the default, it will bring up a prompt to select one of the browsers.

On the other hand, for dynamic analysis, we extracted the network flow features by using CICFlowMeter [14], a network traffic flow generator and analyzer, to generate bidirectional flows, where the first packet determines the forward (source to destination) and backward (destination to source) directions. More than 80 statistical network traffic features such as duration, number of packets, number of bytes, and length of packets, can be measured, in both the forward and reverse directions (see Appendix C).

[8] https://play.google.com/store/apps/details?id=com.facebook.katanahl=en
[9] https://play.google.com/store/apps/details?id=com.skype.raiderhl=en
[10] https://play.google.com/store/apps/details?id=com.google.android.apps.fireballhl=en
[11] https://play.google.com/store/apps/details?id=com.google.android.apps.messaginghl=en
[12] https://play.google.com/store/apps/details?id=com.whatsapphl=en
[13] https://play.google.com/store/apps/details?id=xyz.klinker.messengerhl=en
[14] http://www.unb.ca/cic/research/applications.html

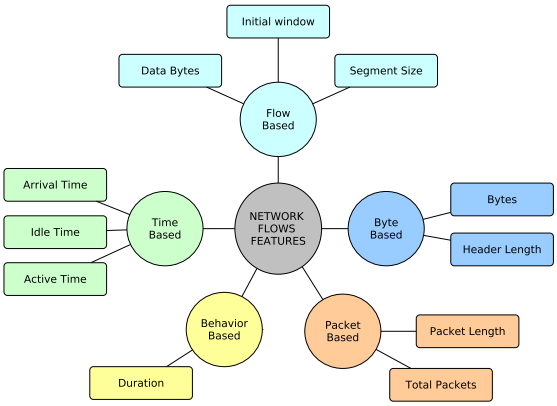

Figure 5.5: Network flow features

Feature Learning and Representation. After running the apps on *ND-Droid*, and capturing the generated traffic at the gateway level, we conducted the feature extraction and selection to find the best set of features, before the training and testing process. In this case, we further analyzed the 80 netflow features extracted from the CIC-flowmeter and categorized them into eight different categories, as depicted in Figure 5.5. Additionally, we created a Netflow-parser to automatically generate the network flow features into sets for each apk. To provide a concise representation of the extracted traffic, the Netflow-parser employed the aggregation modules such as Maximum, Minimum, Mean and standard deviation for each category.

5.6 Summary

In this chapter, we described the design and implementations of our framework, which includes the dataset description, system configuration, tools and methodology, and the static and dynamic implementation details.

The next chapter, chapter 6, presents the proposed framework evaluation and discusses the results and findings of the experiments and case studies.

Chapter 6

Result and Discussion

You've got to experiment to figure out what works

—Andrew Weil

6.1 Overview

To demonstrate the effectiveness of the proposed framework in Chapter 4, we performed a set of experiments along with the case studies. The combination of experimental design and case study is suitable for Android financial malware investigation. The experimental method involves manipulating one variable to determine if changes in one variable cause a change in another variable. A case study method provides an understanding of a complex issue and adds strength to what is already known through previous research [18]. In this chapter, we first present the analysis results of the experiments, followed by the case studies, and then evaluation results.

6.2 Experiment and Results

Experimental Flow. An experimental flow is designed as the master plan of our research that shows the main steps of how the experiment is conducted. Figure 6.1 illustrates the flow of the experiments, which involves six experiments that are tested on various set of data (as listed in Table 5.1). In light of the research questions and hypotheses presented in Chapter 1, the experiments are structured as follows:

- *Experiment 1: Data evaluation based on the proposed taxonomy* - The goal of this experiment is to understand the characteristics that uniquely identify financial malware, and to evaluate the proposed taxonomy. The experiment involves the following stages: a) dataset creation, which is to create a dataset based on the identified category in the taxonomy; b) analysis, which is to analyze the collected dataset; c) characterization, which is to characterize the behavior of each malware category; d) correlation, which is to correlate the behavior of each malware category.

  Dataset employed: *D5: Taxonomy* and *D6: Financial Malware*.

- *Experiment 2: The effectiveness of static vs. dynamic modules* - This experiment aims to investigate the performance of static and dynamic analysis. To achieve this goal, two sub-experiments are performed: a) comparison analysis of static and dynamic; b) combining static and dynamic features.

  Dataset employed: *D7: Scareware*.

- *Experiment 3: Analysis of feature engineering* - The purpose of this experiment is to identify concrete features that can be used to detect financial malware effectively. The experiment consists of two phases: a) feature selection and ranking: enumerate all possible features using several methods such as information gain and random forest feature importance; b) feature vector and representation - discover useful features from raw data including n-gram and flow parsing (e.g., .pcap to .csv).

  Dataset employed: *D7: Scareware*.

- *Experiment 4: Threshold analysis of decision-making module* - This experiment defines the threshold score of the decision-making module in the static analysis. The threshold score determines whether the app is to be analyzed in the dynamic analysis stage after completing the static analysis.

  Dataset employed: *D7: Scareware* and *D5: Taxonomy*.

- *Experiment 5: Evaluation of detection accuracy of the proposed framework* - This experiment describes the evaluation of the detection accuracy of our framework when employed on five common machine learning classifiers such as k-Nearest Neighbors (kNN), Support Vector Machine (SVM), Logistic Regression (LR), Naive Bayes (NB), and Random Forest (RF).

  Dataset employed: *D3: AAGM* and *D7: Scareware*.

- *Experiment 6: Comparative analysis of state of the art Copperdroid Emulator* - The main objective of this experiment is to conduct a comparison study between our dynamic module, *ND-Droid* and the state of the art Copperdroid emulator [1], which was developed by the Royal Holloway University of London. Over 2k samples have been submitted to *Copperdroid*, but due to some limitations (timeout, app's error, failed to allocate memory) they only managed to install 838 samples.

  Dataset employed: *D4: Benign, D6: Financial Malware, D8: AMD, D10: Maldozer*.

[1] http://copperdroid.isg.rhul.ac.uk/copperdroid/

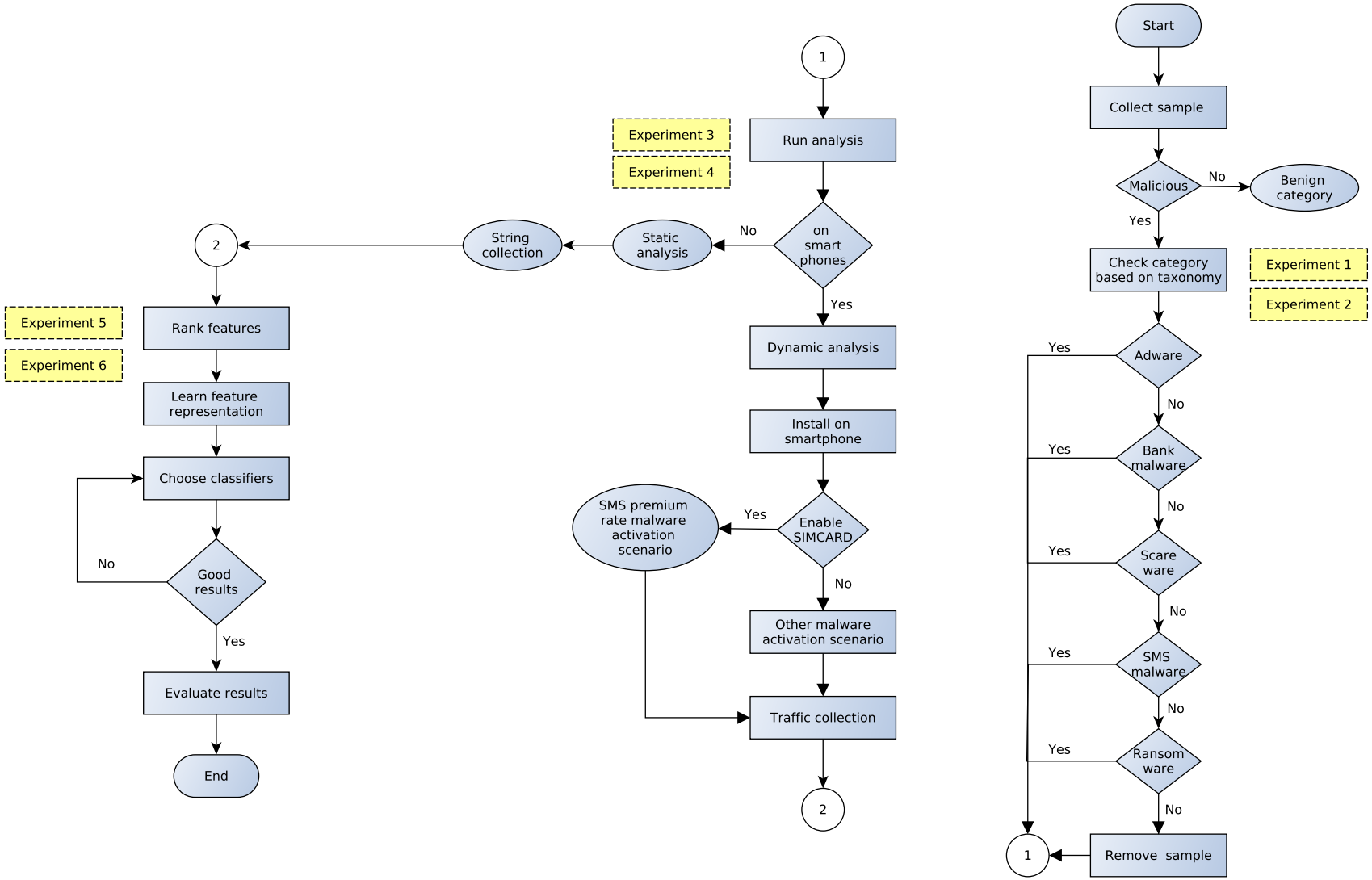

Figure 6.1: The flow of the experiments

6.2.1 Data evaluation based on proposed taxonomy

Outline. We presented the evaluation results of the Android financial malware taxonomy to answer the first and second research question, *RQ1: What constitutes financial malware* and *RQ2: Does financial malware exhibit unique characteristics that can be used to differentiate it from other malware types?* Particularly, we analyzed the malware behaviors in our dataset and classified them into: 20 types of attacks (A1-A20), 4 types of C&C communications (C1-C4), and 5 malware categories i.e., adware,banking malware, ransomware, scareware, SMS malware, as shown in Table 6.1. After that, we compiled the following main characteristics of each categories: malware installation (how does the malware spread to the Android users?), malware activation (how does the malware activate itself on the phone?), and malware attacks (what happens after the malware has reached the Android system?).

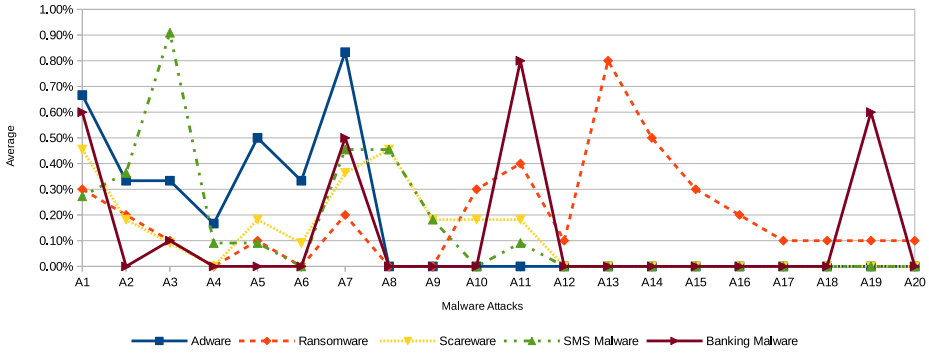

Figure 6.2: Attacks per category

Malware Behaviors. In this phase, we gathered all reports from multiple antivirus vendors such as Fortinet [9], Kaspersky [7], Avast [8], and other security blogs [10, 11] and compiled the following categories:

- Malware attacks - To further analyze the malware attacks and its characteristics, we looked deeper into the twenty categories (A1 to A20) of attack types for each malware family and category: (1) malware family: out of 52 total families, ransomware has the highest number of attacks (per family) with 75% (39 families), followed by

SMS malware with 62% (26 families), banking malware and scareware both with 50% (26 families), and adware with 37% (19 families). (2) malware category: Figure 6.2 shows the average of attacks per category. The top-5 attacks per category are as follows: about 90% of SMS malware performed an A3 attack; this is followed by adware with more than 80% using an A7 attack and almost 70% using an A1 attack; about 80% of banking malware and ransomware performed A11 and A13 attacks, respectively.

- Malware communication - We also investigated the communications between the malware and C&C servers. Overall, about 33% of malware families connected with C&C servers. We found that banking malware and ransomware families had surpassed the other categories (with 10 out of 52 families).

Overall, we found that Android financial malware uses one of the following avenues to gain financial profit:

- Data theft, which is to gather sensitive information by stealing personal data (adware, banking malware).
- Money transfer: to steal the login credentials for online banking and credit card information by replacing the authentication fields of Android apps (e.g., mobile banking apps) on the infected mobile devices (banking malware).
- Ransom payment, which is to regain control over the mobile devices by locking the mobile screen or encrypting the personal data stored on the mobile devices (ransomware).
- Product payment, which is to persuade a user to buy fake apps or fake services (scareware).
- Fraud SMS charge, which is to exploit the mobile service (phone billing system) to subscribe a user to a premium-rate SMS without the user's consent (SMS malware).

Table 6.1: Android financial malware behaviors

	Family	Year	AV Labelled	Total Collected	Total Captured	Attack																				Communication			
						A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	C1	C2	C3	C4
ADWARE	Dowgin	2016	Gdata	50	10		✓																						
	Ewind	2017	Koodous	50	10	✓		✓		✓		✓														✓	✓		
	Feiwo	2016	Fortinet	100	15	✓																							
	Gooligan	2016	Fortinet	43	14	✓				✓	✓	✓														✓		✓	
	Kemoge	2015	Lookout	35	11							✓																	
	koodous	2017	Koodous	50	10																								
	Mobidash	2015	Enet32	32	10				✓																	✓	✓	✓	
	Selfmite	2014	AntiVirus	6	4		✓	✓				✓																	
	Shuanet	2015	Lookout	24	10					✓	✓																		
	Youmi	2015	Gdata	50	10	✓						✓																	
RANSOMWARE	Charger	2017	Sophos	150	10	✓						✓			✓	✓		✓											
	Jisut	2017	ESET	150	10										✓			✓			✓		✓	✓	✓	✓			
	Koler	2015	Avast	135	10		✓	✓								✓	✓	✓											
	LockerPin	2015	ESET	150	10					✓					✓			✓	✓		✓								
	Simplocker	2015	Symantec	146	10	✓						✓						✓	✓							✓	✓	✓	
	Pletor	2014	Alibaba	112	10											✓		✓	✓	✓		✓				✓			✓
	PornDroid	2016	Ikarus	113	10		✓											✓		✓						✓			
	RansomBO	2017	Fsecure	100	10																								
	Svpeng	2014	Sophos	120	11	✓												✓	✓							✓			
	WannaLocker	2017	Avast	18	10											✓			✓	✓									
SCAREWARE	AndroidDefender	2013	Fortinet	150	17						✓	✓	✓																
	AndroidSpy.277	2016	DrWeb	150	6	✓						✓														✓			
	AV for Android	2015	Mixed AVs	83	10								✓																
	AVpass	2013	Kaspersky	150	10		✓					✓			✓														
	FakeApp	2015	Kaspersky	150	10								✓	✓		✓													
	FakeApp.AL	2015	Mixed AVs	150	11			✓					✓	✓		✓													
	FakeAV	2013	Kaspersky	150	10	✓				✓					✓														
	FakeJobOffer	2013	Fsecure	9	9	✓																							
	FakeTaoBao	2013	Mixed AVs	150	9	✓						✓																	
	Penetho	2012	Kaspersky	150	10	✓	✓			✓																✓			
	VirusShield	2014	Mixed AVs	150	10								✓																
SMS	BeanBot	2011	Sophos	85	10	✓		✓					✓													✓			
	Biige	2012	Alibaba	147	11		✓	✓		✓		✓																	
	FakeInst	2013	BkAv	150	11			✓																					
	FakeMart	2012	Alibaba	63	10			✓	✓				✓																
	FakeNotify	2012	Fsecure	150	10			✓				✓		✓															
	Jifake	2012	Fsecure	149	10		✓	✓				✓	✓																
	Mazarbot	2016	ESET	22	10	✓		✓				✓	✓	✓		✓										✓			
	Nandrobox	2012	NANO	150	11		✓	✓																					
	Plankton	2011	Ikarus	150	10		✓					✓																	
	SMSsniffer	2011	Ikarus	53	9	✓		✓																					
	Zsone	2011	Avira	150	10			✓					✓																
BANKING	Bankbot	2015	Fsecure	136	11	✓																		✓		✓			
	Binv	2014	Kaspersky	2	2	✓						✓				✓													
	Sandroid	2014	Fireeye	61	13											✓								✓		✓			
	Wroba	2014	Fireeye	152	10											✓								✓		✓			
	Fakebank	2014	Fireeye	151	8	✓						✓				✓								✓					
	SMSspy	2013	Fsecure	131	8			✓								✓								✓					
	Zertsecurity	2013	Kaspersky	4	4	✓										✓													
	Citmo	2012	Fireeye	3	3	✓						✓				✓										✓			
	Spitmo	2011	Fsecure	191	7	✓						✓												✓					
	Zitmo	2010	Mixed AVs	142	11							✓				✓										✓			
Total				4354	506	**21**	**10**	**15**	**2**	**7**	**3**	**21**	**10**	**4**	**5**	**15**	**1**	**8**	**5**	**3**	**2**	**1**	**1**	**7**	**1**	**17**	**3**	**3**	**1**

Legend:
A1: Steal information (contacts and sms messages, credit card credentials)
A2: Harvest information (text messages, bookmark history and mobile accounts)
A3: Send SPAM SMS / Phishing to the contact list
A4: Make silent calls in background
A5: Exploit root (Privilege escalation)
A6: Inject malicious code
A7: Download malicous software
A8: Premium subscription
A9: Direct user to malicious web site
A10: Prevalence of the malware (uninstall AVs, encrypt source, load dinamic code)
A11: Geographic location attack
A12: Send shortened URLs

Legend:
A13: Lock the device
A14: Encrypt the user files
A15: Modify contents,(SD card)
A16: Change the PIN number
A17: Use the Video Camera
A18: Voice,message to unlock the screen
A19: Fake login screen
A20: Change wallpaper
C1: C&C servers
C2: Start after time delay
C3: Update with new features
C4: Using TOR

Malware Installation. In order to evaluate the Android financial malware, we first need to understand how malware lands on a phone and gets activated. By inspecting the malware samples in our collection, we categorized the ways Android malware are to be installed (as shown in Table 6.3):

1. Repackaging: is a common technique used by the attackers to plagiarize or steal the legitimate applications in creating malicious ones. Due to its openness, the attackers can easily change the code and sign the applications with a self-signed certificate. To check the presence of repackaging in our dataset, we employed FSquaDRA (Fast Detection of Repackaged Applications) [159].

Table 6.2: Analysis of Android financial malware dataset

Category	Types Category	Repackaging	Update Attack	Social Engineering	Boot	SMS	Network	Call	USB	Package	Battery	System	Botnet Functionality	Information Theft	Spam/Phishing	Privilege Escalation	Functionality Exploitation	Geographic location
		Malware Installation			**Malware Activation**								**Malware Attacks**					
SMS-based	SMS Fraud	✓	✓	✓	✓	✓		✓		✓	✓		✓	✓			✓	
	SMS Phishing	✓	✓	✓	✓		✓	✓		✓		✓	✓	✓	✓			✓
Ransomware	Encryption-based	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓
	Device-locking	✓		✓	✓	✓		✓		✓		✓	✓	✓	✓		✓	✓
Scareware	Fake Software	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fake Service Apps	✓		✓	✓		✓	✓				✓	✓				✓	
Banking malware	Active Attack	✓	✓	✓	✓	✓		✓		✓			✓	✓	✓		✓	✓
	Passive Attack	✓		✓	✓								✓	✓	✓		✓	✓
Adware	Notification Ads		✓	✓	✓			✓	✓	✓		✓				✓		✓
	Icon Ads			✓	✓			✓					✓	✓	✓			✓

FSquaDRA uses a pairwise application comparison to compute similarity between apps, which is based on a variety of metrics such as Euclidean, Block, Jaccard, and Cosine, to name a few. We used the Block metric in our analysis as it gives us more results of similarity with higher scores. In total, among the 1,758 malware samples, FSquaDRA detected 56,525 pairs of similar apps. Following the defined repackaging threshold of 0.7 similarity score, we found 17,480 pairs (30.92%) are repackaged. We also calculated the average of the similarity for each category: 40.56% ransomware, 32.50% SMS malware, 26.98% Banking, 26.93% scareware, and 0% adware. We then looked deeper into the adware results and found that the highest similarity score for one of the adware families named *Shuanet* is 0.655 (with only 1 pair), which is below the threshold. This indicates that the adware samples in our dataset are unique.

2. Update Attack: similar to the repackaging method, the update attack technique also exploits the legitimate apps but specifically on the update component. For instance, instead of repackaging the whole application with a malicious payload, it includes an update component that is activated at runtime.

Once the program is installed on the user's mobile phone, it hijacks the Android update screen and notifies the user that a new update is available. To quantify the update attack technique in our samples, a dynamic analysis is required. As such, we manually inspect the malware family with previous studies [164, 91, 15]. Overall we found approximately 31% (10 out of 32 families) that reported using the update attack technique in delivering malware: 60% (3 families) from the SMS malware category (*GGTracker, Plankton, YZHCsms.*), 50% (2 families) from the adware category (*Kemoge, Shuanet*), 33% (2 families) from the scareware category (*FakeAV, AVpass*), and 30% (3 families) from the banking malware category (*Bankbot, Spitmo, Zitmo*),

3. Social Engineering: a method that requires user's participation to succeed without exploiting the browser. With social engineering, the attackers convince the user to download the malicious payloads, which may happen when checking an e-mail message, visiting a website or by clicking on a deceptive pop-up window. Social engineering can be done through various mediums such as SMS, fake application, email, and drive-by download. An example is the banking trojan called *Spitmo* family. This trojan asks the user to install a new updated app that can better protect banking activities. If the user installs the app, the trojan steals the banking credentials and sends them to a remote server. By applying a similar approach when analyzing the update attack method, we found that all malware categories are using social engineering as a medium of malware delivery: 100% (all families) of scareware (*FakeAV, FakeFlash, FakeJobOffer, FakePlayer, Penetho*), 80% (8 families) of banking malware (*Bankbot, Binv, Fakebank, Sandroid, SMSspy, Spitmo, Wroba, Zitmo*), 60% (3 families) of SMS malware (*GGTracker, Plankton, YZHCsms*), and 44% (3 families) of ransomware (*Koler, Pletor, SVpeng*).

Table 6.3: Analysis of Android financial malware installation

Category	Types	Malware Installation (%)		
		Repackaging	Update Attack	Social Engineering
SMS-based	SMS Fraud	15	40	40
	SMS Phishing	18	20	20
Ransomware	Encryption-based	12	0	14
	Device-locking	28	0	29
Scareware	Fake Software	26	33	86
	Fake Service Apps	2	0	14
Banking malware	Active Attack	25	30	60
	Passive Attack	2	0	20
Adware	Notification Ads	0	50	50
	Icon Ads	0	0	50

Malware Activation. This section further discusses the malware activation once they are installed on the phone. Android applications are typically divided into two categories: pre-installed and user-installed. Pre-installed applications include the original equipment manufacturer (OEM) or the mobile carrier-provided applications such as the calendar, email, browser and contact managers. User-installed applications refer to the applications that the user has installed (including the malicious applications) either through an app market such as Google Play or direct download or manually with adb install. Zhou & Jiang [164] claimed in their paper that Android malware can launch its payloads by registering for the system events. Thus, we further investigate the system events of Android in order to examine the malware activation in our dataset; we reverse engineered the samples and generated a shell script to check the Android system events, which is based on the set of features including system boot, phone, package, system, SMS/MMS, USB storage, battery power, and network. Table 6.4 shows the list of these features with their abbreviation and action description [2]. The results in Table 6.5 and Table 6.7 demonstrate that most of the Android financial malware is executed with the system boot (BOOT). This is not surprising as this particular event will be triggered once the system finishes its booting process.

[2]https://developer.android.com/reference/android/content/Intent.html

In our dataset, 84% (27 malware families out of 32) listened to this event to bootstrap the background service. For instance, the system boot event is triggered with the shell command *am broadcast -a android.intent.action.BOOT_COMPLETED*. This is followed by 78% (25 malware families) with the phone event (CALL): both of ransomware (6 families) and scareware (6 families) have 19% samples, and only 9% of adware samples (3 families) registered to this event. We narrowed down the percentage per category and found that scareware (fake software cattegory) has the highest number of CALL events with 83% on average, followed by SMS fraud (60%), and encryption-based ransomware (57%). The phone events indicate that the call state on the device has changed and an outgoing call is about to be placed. The package event is the third most used event, particularly in scareware. This event consists of four types where the application package can be added, removed, changed, and replaced. For instance, Penetho (scareware fake software) is a hacktool for Android devices that can be used to crack the WIFI password of the router but at the same time able to delete, destroy and steal data.

The SMS_RECEIVED event is ranked fourth with 11 malware families interested in intercepting or responding incoming SMS messages. This is reasonable as many malware intercept or respond to incoming SMS messages. For example, all samples of YZHCsms listens to the SMS_RECEIVED event and intercepts or removes all SMS messages from particular originating numbers, e.g., "12345678911". We also found that certain financial malware registers for a variety of events. For instance, Pletor registered all events except for USB storage. This is reasonable as the nature of Pletor is more sophisticated than other malware families. There are two variants of Pletor: the first uses the Tor network for communicating with its owners; the second uses more standard HTTP and SMS channels. Also, when the modifications demand encrypting money from the user, they display the victim's image using the smartphone's front camera. Moreover, fake software scareware and encryption-based ransomware employed almost all of the events except for the USB.

Table 6.4: Android system events

No	Event Name (Abbreviation)	Events	Action Description
1	System Boot (BOOT)	BOOT_COMPLETED	Sent at boot by all devices. Upon receipt of this event, the user is unlocked
2	Phone (CALL)	PHONE_STATE	Indicates that the call state on the device has changed
		NEW_OUTGOING_CALL	Indicates that an outgoing call is about to be placed
3	Package (PKG)	PACKAGE_ADDED	A new application package has been installed on the device
		PACKAGE_REMOVED	An existing application package has been removed from the device
		PACKAGE_CHANGED	An existing application package has been changed (e.g., enabled or disabled)
		PACKAGE_REPLACED	A new version of an application package has been installed, replacing an existing version that was previously installed.
4	System (SYS)	USER_PRESENT	Sent when the user is present after device wakes up (e.g when the keyguard is gone)
		INPUT_METHOD_CHANGED	An input method has been changed
		SIM_FULL	The SIM storage for SMS messages is full
5	SMS/MMS (SMS)	SMS_RECEIVED	A new text-based SMS message has been received by the device
		WAP_PUSH_RECEIVED	A new WAP PUSH message has been received by the device
6	USB storage (USB)	UMS_CONNECTED	The device has entered USB Mass Storage mode
		UMS_DISCONNECTED	The device has exited USB Mass Storage mode
7	Power Battery (BATT)	ACTION_POWER_CONNECTED	External power has been connected to the device
		ACTION_POWER_DISCONNECTED	External power has been removed from the device
		BATTERY_LOW	Indicates low battery condition on the device
		BATTERY_OKAY	Indicates the battery is now okay after being low
		BATTERY_CHANGED_ACTION	Contains the charging state, level, and other information about the battery
8	Network (NET)	CONNECTIVITY_CHANGE	A change in network connectivity has occurred. A default connection has either been established or lost.

Table 6.5: Analysis of Android financial malware activation

Category	Types	Malware Activation (%)							
		BOOT	SMS	NET	CALL	USB	PKG	BATT	SYS
SMS-based	SMS Fraud	60	40	0	60	0	20	20	0
	SMS Phishing	20	0	20	20	0	40	0	20
Ransomware	Encryption-based	43	14	14	57	0	29	14	14
	Device-locking	57	29	0	29	0	14	0	14
Scareware	Fake Software	67	50	50	83	0	67	17	67
	Fake Service Apps	17	0	17	17	0	0	0	17
Banking malware	Active Attack	70	30	0	50	0	10	0	0
	Passive Attack	20	0	0	0	0	0	0	0
Adware	Notification Ads	25	0	0	50	50	25	0	25
	Icon Ads	25	0	0	25	0	0	0	0

A work by Zhou & Jiang [164] highlighted that most of the Android malware in their dataset registered BOOT and SMS events. Our analysis of recent and more advanced mobile malware shows that in addition to BOOT and SMS, events also register to CALL and PKG. We believe the registration of a large number of events is expected to allow the malware to quickly launch the carried payloads, which indicates the characteristic of financial malware.

Table 6.6: Analysis of Android financial malware attacks

Category	Types	Malware Attacks (%)					
		Botnet	**Theft**	**Spam**	**Privilege Escalation**	**Exploit**	**Location**
SMS-based	SMS Fraud	40	60	0	0	40	0
	SMS Phishing	40	20	40	0	0	40
Ransomware	Encryption-based	14	0	29	0	29	29
	Device-locking	29	14	29	0	29	14
Scareware	Fake Software	67	33	33	67	50	50
	Fake Service Apps	17	0	0	0	17	0
Banking malware	Active Attack	60	40	40	0	60	30
	Passive Attack	20	10	10	0	20	10
Adware	Notification Ads	0	0	0	25	0	50
	Icon Ads	25	25	50	0	0	50

Table 6.7: Malware characterization based on malware installation and activation

Malware Family	**Category**	**Malware Activation**								**Malware Installation**		
		BOOT	SMS	NET	CALL	USB	PKG	BATT	SYS	Repackaging	Update Attack	Social Engineering
Kemoge	Adware				✓	✓					✓	✓
MobiDash		✓			✓							✓
Selfmite												✓
Shuanet		✓			✓	✓	✓		✓		✓	✓
Bankbot	Banking	✓			✓					✓	✓	✓
Binv			✓		✓					✓		✓
Citmo		✓	✓		✓							
FakeBank		✓								✓		✓
Sandroid		✓								✓		✓
SMSspy		✓								✓		✓
Spitmo		✓			✓		✓			✓	✓	✓
Wroba		✓								✓		✓
ZertSecurity		✓	✓	✓								
Zitmo		✓								✓	✓	✓
FakeDefender	Ransomware	✓	✓		✓				✓	✓		
Koler		✓			✓		✓		✓	✓		✓
Pletor		✓	✓	✓	✓		✓	✓	✓	✓		✓
RansomBO		✓								✓		
ScarePakage		✓	✓		✓							
SimpleLocker		✓			✓		✓			✓		
Svpeng		✓			✓					✓		✓
Avpass	Scareware	✓	✓	✓	✓		✓		✓	✓	✓	✓
FakeAV		✓	✓		✓		✓		✓	✓	✓	✓
FakeFlash				✓	✓		✓	✓	✓	✓		✓
FakeJobOffer		✓		✓	✓				✓	✓		✓
FakePlayer		✓	✓		✓				✓	✓		✓
Penetho		✓		✓	✓					✓		✓
Gazon	SMS-based			✓			✓		✓			
GGTracker		✓	✓		✓			✓		✓	✓	✓
Plankton		✓			✓		✓			✓	✓	✓
Uxipp		✓			✓					✓		
YZHCsms		✓	✓		✓		✓			✓	✓	✓
Total number of families		**27**	**11**	**6**	**24**	**2**	**12**	**3**	**9**	**24**	**10**	**24**

Malware Attacks. This section presents the type of attacks for each category in Android financial malware: adware, banking malware, ransomware, scareware, and sms malware. To investigate the attack types, we reviewed security reports from multiple sources including Fortinet [9], Kaspersky [7], Avast [8], and other security blogs [10, 11].

Based on the reviewed reports, we found the following attack types of Android financial malware in our dataset:

1. Information theft (A1): the malware is harvesting various information on the infected phones, including SMS messages, phone numbers as well as user banking accounts including the transaction authentication number (TAN). TAN is used by online banking services as a form of single use one-time password to authorize financial transactions. TANs provide additional security because they act as a form of two-factor authentication. TANs theft is a known attack targeting mobile banking services.

2. Spam and/or Phishing (A2): in mobile malware, the spam and/or Phishing scams are sent over the Short Message Service (SMS) with a shortened URLs to the phone contact list.

3. Botnet functionality (A3): a mobile bot is a type of malware that runs automatically once installed on a mobile device to gain complete access to the device and its contents as well as providing control to the botnet creator. It starts communicating with and receiving instructions from one or more command and control servers. Mobile botnets take advantage of unpatched exploits to provide hackers with root permissions over the compromised mobile device, enabling hackers to send e-mail or text messages, make phone calls, access contacts and photos, and more.

4. Privilege escalation (A4): the malware is capable of taking over the device by exploiting and preserving device administrator privileges. Once the mobile device has been taken over by the cybercriminals, typically, the malware is capable of doing the following actions: locking users out of their device, taking a photo from the device's camera, answering and dropping phone calls, and searching for banking applications on the device.

Table 6.8: Android financial malware attack types

Family	Category	Malware Attack Types												Total Attack (%)
		A1	A2	A3	A4	A5	A6(a)	A6(b)	A6(c)	A6(d)	A6(e)	A6(f)	A6(g)	
Kemoge	Adware						✓							**8**
Mobidash	Adware			✓									✓	**17**
Selfmite	Adware	✓	✓	✓			✓							**33**
Shuanet	Adware				✓							✓		**17**
Bankbot	Banking	✓		✓										**17**
Binv	Banking	✓	✓	✓		✓	✓							**42**
Citmo	Banking	✓				✓								**17**
FakeBank	Banking	✓				✓	✓				✓			**33**
Sandroid	Banking		✓	✓		✓								**25**
SMSspy	Banking		✓			✓								**17**
Spitmo	Banking	✓					✓							**17**
Wroba	Banking	✓	✓	✓		✓								**33**
ZertSecurity	Banking	✓				✓								**17**
Zitmo	Banking	✓	✓	✓		✓	✓							**42**
FakeDefender	Ransomware			✓			✓	✓	✓					**33**
Koler	Ransomware	✓	✓			✓								**25**
Pletor	Ransomware			✓		✓			✓	✓				**33**
ScarePakage	Ransomware					✓								**8**
Simplocker	Ransomware	✓		✓		✓	✓							**33**
Svpeng	Ransomware	✓		✓										**17**
AVpass	Scareware	✓		✓	✓	✓	✓	✓						**50**
FakeAV	Scareware	✓			✓	✓		✓						**33**
FakeFlash	Scareware	✓	✓		✓		✓					✓		**42**
FakeJobOffer	Scareware	✓				✓								**17**
Fake Player	Scareware		✓			✓	✓	✓						**33**
Penetho	Scareware	✓		✓	✓									**25**
Gazon	SMS Malware	✓		✓			✓							**25**
GGTracker	SMS Malware		✓			✓								**17**
Plankton	SMS Malware	✓	✓	✓			✓							**33**
Uxipp	SMS Malware	✓	✓											**17**
YZHCsms	SMS Malware	✓	✓			✓								**25**
Total number of family		**21**	**13**	**15**	**5**	**18**	**13**	**4**	**2**	**1**	**1**	**2**	**1**	

Legend:
A1: Steal and harvest information (contacts, text message, bookmark, credit card credentials)
A2: Send spam SMS/phishing/shortened URLs to the contact list
A3: Communicate with botnet
A4: Exploit root (privilege escalation)
A5: Geographic location attack
A6 (a): Download/install malicious software
A6 (b): Check and uninstall Antiviruses
A6 (c): Modify contents (SD card)
A6 (d): Use the video camera
A6 (e): Infect a connected window pc
A6 (f): Inject malicious code
A6 (g): Make silent calls in background

5. Geographic location attack (A5): the malware targets a specific users based on the geographical location and country.

6. Functionality exploitation (A6): this types of attack exploits the smartphones' functionality, as follows: (a) download/install malicous software, (b) check and uninstall AVs, (c) modify contents (SD card), (d) use the video camera, (e) infect a connected Windows PC, (f) inject malicious code, (g) make silent calls in the background.

Table 6.9: Malware attacks by geographical location

Category	Family	Target Country
Banking Malware	Binv	Brazil
	Sandroid	MiddleEast
	Wroba	Korea
	FakeBank	Iran
	SMSspy	Spain
	ZertSecurity	German
	Citmo	Russia
	Zitmo	Europe

Category	Family	Target Country
SMS Malware	YZHCsms	Asian countries
	FakePlayer	Russia, USA, China
Scareware	FakeJobOffer	India
	FakeAV	USA
Ransomware	Koler	Worldwide (30 countries)
	Pletor	Worldwide (13 countries)
	ScarePakage	USA, UK, Germany
	SimpleLocker	Ukraine, USA

To further analyze malware attacks and their characteristics, we looked deeper into the six categories (A1 to A6) of attack types for each malware category according to our taxonomy. Table 6.8 shows the summary of the attack types based on the malware category and family along with the total percentage for each type of attack. According to the result, information theft exhibits the highest percentage out of other categories (botnet functionality, spam/phishing, privilege escalation, geographic location, and functionality exploitation). About 66% of the malware family steal and harvest the information from the victim, most of them from the banking malware and scareware categories. This is followed by the geographic location attack with 56% in total. It is interesting to note that Android banking malware tends to be focused on specific geographical areas. For instance, 80% of the banking malware families (8 out of 10 families) are targeting a specific country such as Brazil, Korea, Iran, Spain, German, Russia, and others (see Table 6.9).

Overall, we noticed that ransomware targets more than 10 countries worldwide compared to other categories. Adware on the other hand is more universal and targeted all users worldwide. In addition, we also looked into the profit target behind malware infection. We categorized the types of financial charges caused by malware in our dataset into the following categories:

1. SMS charge: users are billed directly based on the fraud scheme service.i.e premium-rate sms service. The amount charged varies according to the target country as different countries have a different type of premium-rate service.

Table 6.10: Example of financial charge

Category	**Malware Family**	**Charge Amount**	**Payment Option**
Money Transfer (Ransom payment)	FakeDefender	99.98 USD	Credit card
	Koler	100-300 USD	- MoneyPak - Prepaid cards
	Pletor	- 15 Euros - 100 rubles - 5000 USD	- QIWI VISA - MoneXy
	ScarePakage	300 USD	MoneyPak
	SimpleLocker	20-200 USD	MoneXy
Product Payment	FakeJobOffer	8150 Rs	Bank Deposit
SMS Charge	YZHCsms	- 3 MYR - 2 USD (per text)	Phone Bill

2. Money transfer or steal: users are tricked to pay for the ransom with different payment options such as money pack (7eleven, Walmart, Kmart, CVS pharmacy, RiteAid pharmacy) VISA wallet or credit card. This category also includes the direct money stealing of online banking or fraud banking.

3. Product payment: users paid for fake products or services such as fake job offer service and fake anti-virus. The payment is done through several options such as credit cards, carrier billing, PayPal, and Google Play credit. Some of the malware provides a manual option through a bank deposit.

Additionally, we also checked both the charge amount and the payment options offered by the malware (see Table 6.10). We found that the amount charged varies according to its target country .e.g., from 2 USD to 5000 USD. There are five different currencies that been used by malware such as United States Dollar(USD), Euro, Russian Ruble, Indian Rupee, and Malaysian Ringgit. The payment options are also ranging from credit card, paypal, Google Play credit, MoneyPak, QIWI VISA to Bank Deposit.

Outcome. In summary, the following categories constitute the Android financial malware: adware, banking malware, ransomware, scareware, and SMS malware. Based on the analysis, Android financial malware does exhibit unique characteristics that can be used to differentiate it from general malware. A work by Zhou & Jiang [163] highlighted that most of the Android malware (general malware) in their dataset registered BOOT and SMS event. Our analysis of financial malware shows that in addition to the BOOT and SMS, system events also register to CALL, PKG, and USB. We believe the registration of a large number of events is expected to allow the malware to quickly launch the carried payloads, which indicates the characteristic of financial malware. In particular, we compared and correlated the analysis results of all malware categories in order to rank the importance of each category in our dataset. Overall, banking malware ranked first in terms of the total number of malware attacks for all categories, scoring about 80% (8 families out of 10) on both geographical location attacks, and information theft. This is followed by the scareware category, SMS malware, ransomware, and adware. Another unique characteristic of financial malware is the financial charge. We discovered that the amount charged varies according to the target country and the payment options are also ranging from credit card, paypal, Google Play credit, MoneyPak, QIWI VISA to Bank Deposit. Unique text strings (based on each category) can be used as one of the characteristics to investigate the role of strings found within an Android executable.

6.2.2 The effectiveness of static vs. dynamic modules

Outline. In this experiment, we investigated the performance of static and dynamic analysis, which includes the following sub-experiments: a) comparing static and dynamic features; b) determining the optimal threshold for the static score. The goal is to answer the fourth research question, *RQ4: What features should be extracted to identify advanced malware?*

Comparison analysis of static features. As mentioned in the previous chapter, we selected several static features to be used such as metadata, opcode, bytecode, and string. Also, based on the reported studies, we used the Support Vector Machine (SVM) classifier in conducting our experiment. There are 24 features of metadata extracted including Smali class Count, Smali Size, Apk Size, and unpacking time. In addition, we also added features of the file encyption metadata from *Heldroid*. For the opcode, bytecode, and string analysis, we tested n-gram of word token with values of n up to 3. On average, 3-grams yields the highest precision. Table 6.11 shows the evaluation results of the static analysis where strings feature outperform the other features with 97% accuracy.

Table 6.11: Evaluation of static feature analysis using SVM classifier

Feature Category	**Accuracy**	**Precision**	**Recall**
Metadata (24 features)	87.21	89.00	87.20
Strings (3-gram)	97.03	91.30	92.72
Opcode (3-gram)	92.75	79.42	84.36
Bytecode (3-gram)	83.12	69.27	67.75

Comparison analysis of dynamic features. Based on the gap discovered in the literature review (Chapter 2), we selected two dynamic features (network traffic and system calls) to be analyzed in detail. In this sub-experiment, we used the dataset (291 samples) and the *Copperdroid* emulator employed in Experiment 3. We extracted 118 features of Syscalls and 80 features of network flows of three categories: adware, banking malware, and benign. The results in Table 6.12 illustrates that by using the emulator, the system call features outperforms the network traffic. However, as highlighted in the Experiment 3, emulator has some limitations if compared to the phone. The results with the phone reveals that the network traffic feature outperforms the system call with 99% accuracy.

Table 6.12: Evaluation of dynamic feature analysis

Classifier:	DT	RF	SVM
Syscall (Emulator) accuracy (%):	90	88	92
Network Traffic (Emulator) accuracy(%):	81	85	74
Network Traffic (Phone) accuracy(%):	99	99	72

Static vs. Dynamic Analysis In this section, we presented the results obtained in detecting scareware statically and dynamically based on the outputs obtained from the previous experiments: selected 3-gram string features for static analysis and network features (on-device analyzer) for dynamic analysis. We focused on analysis of: (1) binary classification, which is the classification of a sample as either a malicious or benign, (2) Category classification, which refers to the classification of samples into 3 classes (2 of which are scareware types and 1 represent benign), (3) Family classification, where the samples are classified as one of 13 different classes, 12 of which are malware and 1 class representing benign apps.

In order to evaluate the detection performance, we split the data into 60% train-set and 40% test-set. We reported three metrics in each scenario: F-measure, accuracy, and False Positive Rate (FPR) by using the Scikit-learn machine learning library [3]. For static analysis, we tested up to 3-gram of word tokens with all classifiers. On average, 3-gram yields the highest precision. We report the result of the 3-gram for all three scenarios (see Table 6.13). KNN surpasses RF in binary detection, but RF performs the best in all scenarios: 97.38% accuracy with binary detection, 97.27% accuracy with category detection, and 97.12% accuracy with family detection with an average of 0.095 FPR. Similar to the static analysis, RF also performs the best in all scenarios for dynamic analysis: 99.30% accuracy with binary detection, 97.17% accuracy with category detection, and 92.83% accuracy with family detection with an average of 0.046 FPR (see Table 6.14).

[3]http://scikit-learn.org/stable/

Table 6.13: Scareware detection results with 3-gram word of string

Algorithm	Binary Detection (2-classes)			Category Classification (3-classes)			Family Characterization (13-classes)		
	F-Measure	Accuracy	FPR	F-Measure	Accuracy	FPR	F-Measure	Accuracy	FPR
NB	55.41	89.93	0.516	68.733	95.00	0.291	48.31	94.31	0.444
KNN	87.31	98.94	0.113	78.63	95.99	0.194	60.28	95.32	0.262
SVM	90.31	96.46	0.185	82.98	96.57	0.165	69.13	96.55	0.608
LR	91.83	97.14	0.124	84.27	97.01	0.148	71.17	97.17	0.505
RF	91.72	97.38	0.014	86.91	97.27	0.108	72.87	97.12	0.162

Table 6.14: Scareware detection result with Network flow

Algorithm	Binary Detection (2-classes)			Category Classification (3-classes)			Family Characterization (13-classes)		
	F-Measure	Accuracy	FPR	F-Measure	Accuracy	FPR	F-Measure	Accuracy	FPR
NB	98.40	98.33	0.031	94.20	93.67	0.033	92.40	92.33	0.031
KNN	98.80	98.83	0.075	95.90	95.83	0.077	92.50	92.83	0.076
SVM	85.30	90.00	0.900	85.30	90.00	0.900	85.30	90.00	0.900
LR	99.00	99.00	0.075	95.00	94.67	0.063	90.80	90.17	0.046
RF	99.30	99.30	0.045	96.50	97.17	0.047	92.60	92.83	0.046

Since both static and dynamic results (Figure 6.3) yield a high accuracy and low FPR, there is no need for us to have the integrated feature vectors between the 3-gram unreferenced strings and the 80 nominal network flow features for the purpose of increasing the accuracy. Our results demonstrate that the raw features of each analysis is adequate in detecting scareware, classifying the category, and characterizing its family accurately with a very low FPR.

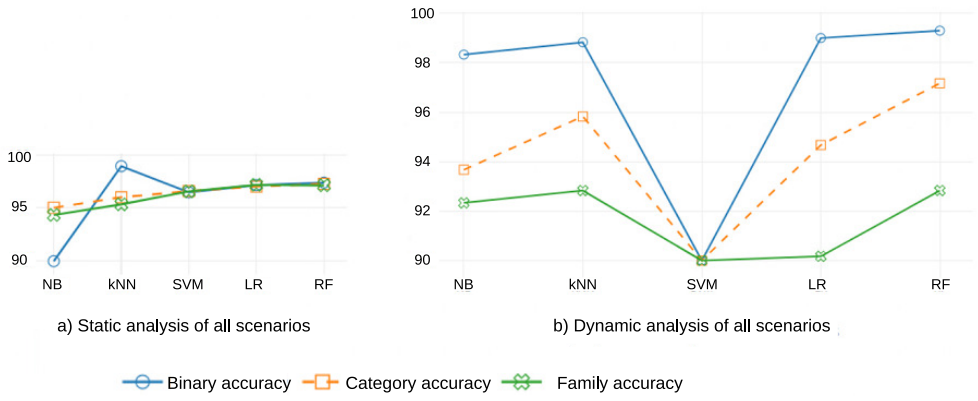

Figure 6.3: Accuracy comparison of static and dynamic analysis for all scenarios

Outcome. To answer the fourth research question, string and network traffic are the best features that can be used to identify advanced malware. The experimental results revealed that string features outperformed other features (metadata, opcode, bytecode) with 97% accuracy. Moreover, the results with the phone illustrated that the network traffic feature outperformed the system call with 99% accuracy. We evaluated these two features on the scareware dataset with three scenarios and discovered that Random Forest (RF) performed the best in all scenarios: 97.38% accuracy with binary detection, 97.27%accuracy with category detection, and 97.12% accuracy with family detection with an average of 0.095 FPR. Similar to the static analysis, RF also performed the best in all scenarios for dynamic analysis: 99.30% accuracy with binary detection, 97.17% accuracy with category detection, and 92.83% accuracy with family detection with an average of 0.046 FPR. Finally, by evaluating four other datasets with string analysis, we defined the score for the static analysis to be 97.67% accuracy. Hence, for the static analysis, a score less than 97% can be used as the threshold for the next stage, which is the dynamic analysis.

6.2.3 Analysis of feature engineering

Outline. Similar to Experiment 4, the aim of this experiment is to answer the fourth research question, *RQ4: What features should be extracted to identify advanced malware?* We identified some high level malware behaviors and selected concrete features (strings and network traffic), that may reflect those behaviors. Thus, in this experiment we conducted the following: a) enumerated important features using several methods such as information gain and random forest feature importance; b) discovered useful features from raw data and represented them as feature vectors (e.g., from .pcap to csv files).

Feature Importance. It is important that any malware detection and classification techniques are able to use a minimal number of features.

Table 6.15: Top-3 most important 3-grams string features for static analysis

Class	Rank	Token
Benign	1	<LB>this <LB>
	2	<LB>accessFlags <LB>
	3	<LB>android.intent.action.VIEW <LB>
Malware	1	<LB>/system/etc/.rild <LB>
	2	<LB>/system/etc/.dhcpcd <LB>
	3	<LB>sysName <LB>

As such, for the static analysis, we tested the proposed framework using only the 50 highest ranked 3-gram tokens, extracted from the unreferenced strings, for each category, subcategory, and family, as recommended by Killam et al. [116]. Table 6.15 shows an example of the 3 most important 3-grams when used to classify malware and benign. Note that <LB> is the line boundary token and that the whitespace between the words was removed to reduce memory consumption when training. For dynamic analysis, we compared the netflow of benign and malicious apps based on the netflow feature categories presented in previous chapter. This comparison is listed in Table 6.16. After that, we ranked the features by using the common feature selection algorithms including Information Gain, CFS Subset-evaluator, Recursive Feature Elimination (REF), and Random Forest Regressor. Table 6.17 shows the detection accuracy with the feature selection algorithms. The result indicates that we can detect malicious behavior with a minimum four feature set (fwd packet length, bwd packet length, init win bytes fwd, init win bytes bwd) via a CFS subset-evaluator with 65.33% detection accuracy, and without affecting the detection results significantly.

Feature Representation. Table 6.18 and Table 6.19 show the feature representation used in the experiment for both static and dynamic cases, respectively. Previous experiments on scareware showed that 3-grams yield a higher detection result. However, in the case of malware vs. benign, 1-gram representation outperformed other representations with 94.49% accuracy and 0.060 FPR via the RF classifier (see Figure 6.4 and Table 6.18).

Table 6.16: Extracted range values of benign and malware traffic for dynamic analysis

Category	Traffic Feature	Benign Traffic	Scareware Traffic
Behavior-based	Flow duration	1 - 119998846	-1 - 119999005
Flow-based	Flow packet	0.017 - 2000000	-2000000 - 2000000
	Inital window (forward)	-1 - 65535	-1 - 65535
	Inital window (backward)	-1 - 65535	-1 - 65535
	Segment size (forward)	0 - 2664	0 - 2587
	Segment size (backward)	0 - 1460	0 - 1460
Byte-based	Header length (forward)	-48040688260 - 12789357918	-3361587770 - 19439833982
	Header length (backward)	-94079681369 - 21997696965	-59383625813 - 36662844305
	Flow bytes	0 - 106000000	0 - 132500000
Packet-based	Packet Size	0 - 2052	0 - 1556
	Download/Upload ratio	0 - 28	0 - 15
	Total length (forward)	0 - 6515047	0 - 9043781
	Total length (backward)	0 - 18607298	0 - 150398500
	Total Packets (forward)	1 - 4734	1 - 28775
	Total Packets (backward)	0 - 12897	0 - 104309
Time-based	Arrival time (forward)	0 - 119998846	0 - 119999005
	Arrival time (backward)	0 - 119994165	0 - 119996371
	Idle time (mean)	0 - 119886447	0 - 119951391
	Active time (mean)	0 - 88071608	0 - 97184879

Table 6.17: Evaluation of feature selection with different algorithms for dynamic analysis

Feature Selection Algorithm	Selected Features (see Feature's ID in Appendix C)	Type	Accuracy (selected features)	Accuracy (all features)
Information Gain	21, 26, 3, 20, 6, 64	Filter	65.62	66.67
CFS Subset-evaluator	6, 13, 52, 53	Wrapper	65.33	
Recursive Feature Elimination	1, 3, 18, 37, 20 (Random Forest) 9, 11, 13, 41, 53 (Logistic Regression)	Wrapper	64.21 59.37	
Random Forest Regressor	1, 21, 20, 3, 18, 37, 17	Filter	64.65	

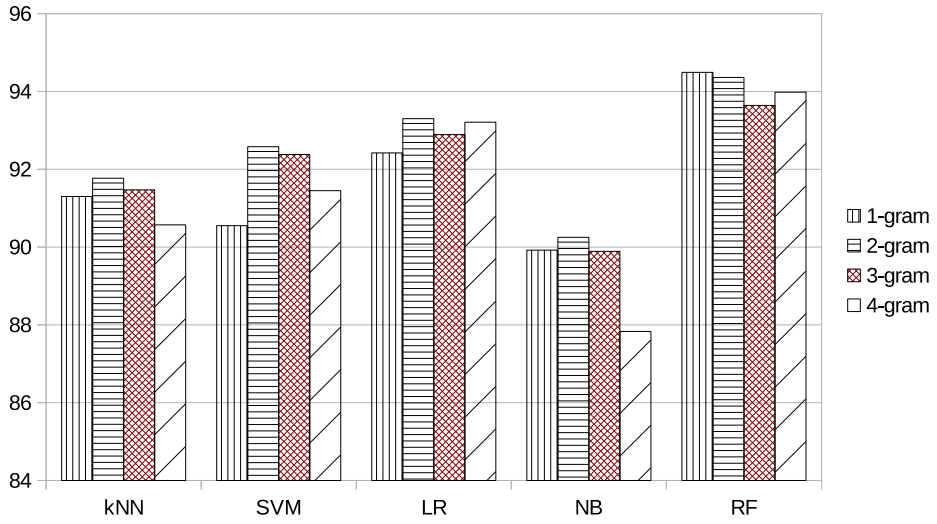

Figure 6.4: Detection accuracy of static analysis up to 4-gram

Table 6.18: Static analysis of malware binary detection (from 1-gram to 4-gram)

Malware Binary Detection							
n-gram	Classifier	Accuracy (%)	FPR (avg)	n-gram	Classifier	Accuracy (%)	FPR (avg)
1-gram	kNN	91.30	0.056	3-gram	kNN	91.47	0.075
	SVM	90.55	0.120		SVM	92.38	0.097
	LR	92.42	0.105		LR	92.89	0.095
	NB	89.92	0.071		NB	89.89	0.086
	RF	94.49	0.060		RF	93.64	0.068
2-gram	kNN	91.77	0.073	4-gram	kNN	90.57	0.086
	SVM	92.58	0.094		SVM	91.45	0.103
	LR	93.30	0.092		LR	93.21	0.089
	NB	90.25	0.083		NB	87.83	0.096
	RF	94.36	0.062		RF	93.98	0.067

Table 6.19: Evaluation of dynamic analysis on Netflow parser

Training-Testing (60-40) via CFS subset-evaluator				
Classifier	With Netflow parser (all features)	With Netflow parser (selected features)	Without Netflow parser (all features)	Without Netflow parser (selected features)
NB	87.38	93.48	55.70	49.50
SVM	49.06	49.07	59.38	59.38
K-NN	94.86	93.48	64.59	64.81
ZeroR	49.07	49.06	59.38	59.37
RF	96.50	96.73	63.78	65.33
DT	96.50	95.57	65.33	64.19

Similarly, RF outperformed other classifiers with 96.73% accuracy by using the Netflow parser, and 65.33% accuracy without using the parser (see Table 6.19).

Outcome. To support the answer in Experiment 4, we numerated important features of string and network traffic and discovered useful features from raw data and represented them into effective feature vectors. Overall, we selected the Top-50 important features for static analysis and discovered that the 1-gram performed well in malware binary detection and 3-grams performed better in scareware detection with more than 90% accuracy. Also, for dynamic analysis, we revealed that employing the Netflow parser with the features selected by a CFS subset-evaluator algorithm gives a better detection result with 97% accuracy.

6.2.4 Threshold analysis of decision making module

Outline. In this section, we conducted another experiment to determine the threshold score of the decision-making module in static analysis. The main idea is to have a hybrid approach of static and dynamic analysis where the threshold score determines whether the app should be analyzed in the dynamic analysis stage. This module can save analysis time and can avoid the feature complexity of combining different feature vectors of static and dynamic models.

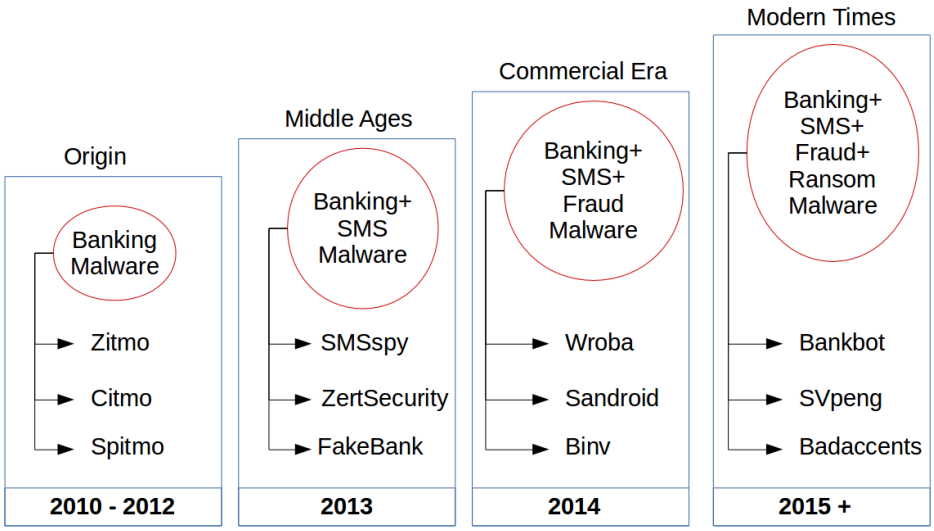

Figure 6.5: Evolution of Android financial malware (data is based on our dataset)

For a comprehensive evaluation, we selected four different datasets (Genome, taxonomy, scareware, general malware), which are based on the evolution of malware behavior (see Figure 6.5):

1. 1st stage(2010): the introduction period of Android banking malware. The evolution started primarily with the release of the traditional desktop banking malware in the mobile versions. e.g., Zitmo, Spitmo, Citmo.

2. 2nd stage (2013): these malware families feature simplistic modifications, recompiled the source code with improved infection and distribution strategies, e.g., ZertSecurity, SMSspy, Fake-Bank.

Table 6.20: Evaluation of string analysis for malware binary detection

Year (evolution phase)	Dataset	Binary detection accuracy (%)
2012 (origin)	Genome	99
2014 (middle age)	CIC-Taxonomy	98
2016 (commercial era)	CIC-Scareware	97
2017 (modern)	CIC-Malware	94
	Average	**97**

3. 3rd stage (2014): during this year we saw malware emerging with innovative techniques based on new infection strategies and payloads, e.g., Wroba, Sandroid, Binv.

4. 4th stage (2015 - onwards): this is the most advanced stage that covers the recently discovered malware such as Bankbot and Svpeng. It is capable of employing advanced techniques for infection and distribution, which are combined with the ransomware technique.

This evolution indicates the level of sophistication of malware, which gives insight for further analysis in the static module. Thus, to set a benchmark threshold score, we calculated the average accuracy of the four sets of malware evolution datasets (see Table 6.20) with the following average formula:

$$Threshold_{score} = \frac{1}{n} * \sum_{i=1}^{n} X_{accuracy_i} \tag{6.1}$$

where n represent the number of items (experiments) and $X_{accuracy_i}$ represents the value (accuracy) of each individual item (experiment).

Outcome. Our analysis shows the average accuracy of 97% (see Table 6.20), which indicates that the static analysis is able to detect the malicious app (on different level of sophistication) with consistent accuracy. This average value is used as the threshold of the decision-making module in our proposed framework.

6.2.5 Evaluation of detection accuracy of the proposed framework

Outline. To evaluate the detection accuracy of our framework, we employed the common classifiers of machine learning such as kNN, SVM, LR, NB, and RF. Based on the reviews in Chapter 2 and previous experiments, RF typically performs the best among other classifiers. Thus, in this experiment we evaluated if this statement is true for Android financial malware.

Analysis. In this experiment, we compared two different datasets: malware and scareware. Figures 6.6 and 6.7 show the result of malware analysis for both static and dynamic analysis where RF outperformed other classifiers with 94% and 97% accuracy, respectively. Similarly, Figures 6.8 and 6.9 present the result of scareware for both static and dynamic analysis where RF also outperformed other classifiers with more than 97% and 99.30% accuracy, respectively.

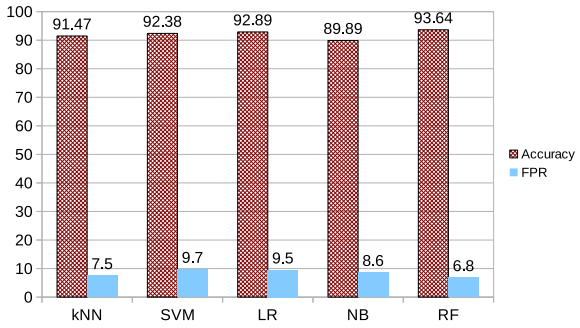

Figure 6.6: 3-gram malware detection

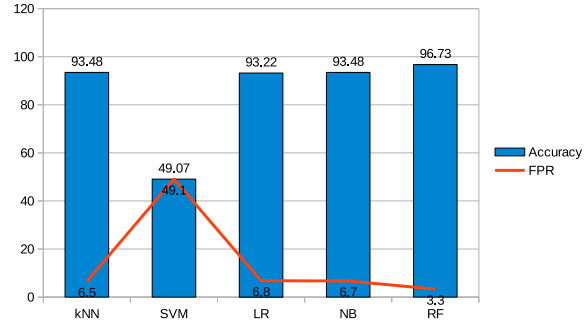

Figure 6.7: Netflow malware detection

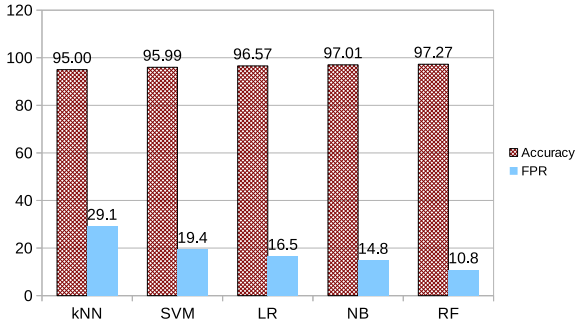

Figure 6.8: 3-gram scareware detection

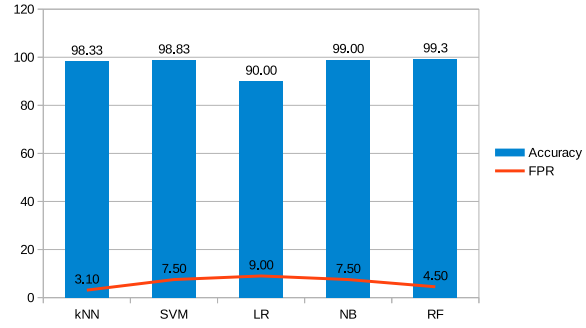

Figure 6.9: Netflow scareware detection

Random Forest Classifier. RF is a supervised learning algorithm. The "forest" it builds, is an ensemble of Decision Trees (Figure 6.10), most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result. RF is a flexible, easy to use machine learning algorithm that produces a great result most of the time (even without hyper-parameter tuning).

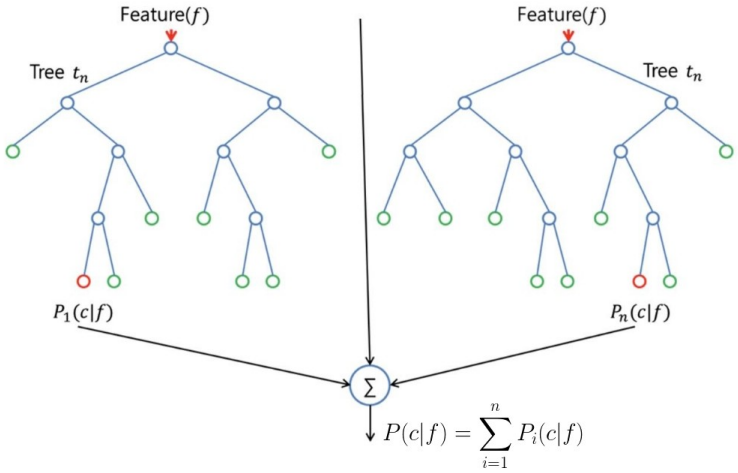

Figure 6.10: Random Forest algorithm

Algorithm 3 lists the the pseudocode of using the RF classification algorithm. For experimentation, we run tests for each of the number of tree in random forest by using the Python module available in the scikit-learn library [113]. The library consists of several options in selecting parameters. For example, the maximum features (max_features) parameter, which also refers as the number of features (n) to consider when looking for the best split, can be separated into the following types:

1. If *sqrt*, then consider max_features equal to $\sqrt{n}$
2. If log_2, then consider max_features equal to $\log_2 n$
3. If *None*, then consider max_features equal to n

Figure 6.11 visualizes the error rate based on the depth of tree up to 1,500 trees. The results reveal that the error rate is increasing after reaching 100 trees via $\log_2$ option. Thus, in our framework we set the number of trees of RF to be 100.

Algorithm 3 Random Forest Classification Algorithm

procedure INPUTS
  $S \leftarrow (X_1, Y_1)(X_n, ..., Y_n)$ ▷ training set
  $F \leftarrow features$ ▷ features
  $B \leftarrow trees$ ▷ number of trees in forest
procedure OUTPUT
  $RandomForest \leftarrow (S, F)$ ▷ Random Forest classifier
function RANDOMFOREST(S,F)
  $H \leftarrow 0$
  for $i \in 1, ..., B$ **do**
    $S(i) \leftarrow$ bootstrap sample from S
    $h(i) \leftarrow$ RANDOMIZEDTREELEARNS $(S(i), F)$
    $H \leftarrow H \bigcup h(i)$ ▷ randomize tree
  end for
  return H ▷ return solution
end function
function RANDOMIZEDTREELEARNS(S,F)
  for each node: **do** ▷ get node of tree
    $f \leftarrow$ very small subset of F ▷ randomize subset of feature's number
    Split on best feature in f
  end for
  return The learned tree ▷ return best feature
end function

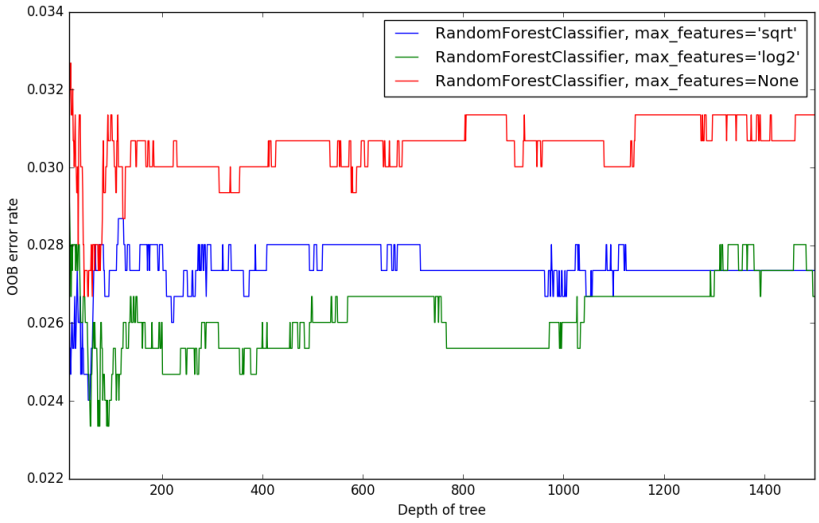

Figure 6.11: Out-of-bag error of RF vs. depth of trees

Outcome. To answer the sixth research question, our proposed framework performs better in detecting malware. Our studies on two datasets revealed that with RF, the detection rate is exceeded more than 95% accuracy with low FPR.

6.2.6 Comparative analysis of state-of-the-art Copperdroid

Outline. In this experiment we present an investigation of machine learning based malware detection using dynamic analysis on both a virtual environment and a real device. We compared the detection results of the *Copperdroid* emulator with our on-device analyzer called *ND-Droid*. As argued by Alzaylee et al. in 2017 [31], several features could be extracted more effectively from the on-device dynamic analysis compared to emulators. It was also found that approximately 24% more apps were successfully analyzed on the smartphone. They also claimed that most of the studies in machine learning based detection performed better when applied to features extracted from on-device dynamic analysis. We conducted this experiment to investigate if this statement is true for Android financial malware. Our goal is to answer our third research question, *RQ3: Does using the real smartphone for malware dynamic analysis perform better than the emulator?*

Analysis and Feature Extraction. Out of 2k samples (1500 malware and 500 benign), 787 samples (39.35%) were successfully analyzed on a real phone compared to only 291 samples (14.55%) on an emulator, as illustrated in Table 6.21.

Table 6.21: Percentage of successfully analyzed Android apps

	ND-Droid Phone	**Copperdroid Emulator**
Malware Samples	52.47%	5.067%
Benign Samples	100%	43.00%
Total	**39.35 %**	**14.55%**

As we mentioned in the previous chapter, we faced some challenges when running samples on *ND-Droid* due to the unsigned apk error.

In contrast, the *Copperdroid* highlighted the following errors when running the samples in their environment:

- failed because the sample was not a valid APK file e.g., unsigned apk.
- failed in processing e.g., bad utf8 byte, bad ascii characters, list index out of range, bad unpack parameters, bad CRC-32 for some assets file.
- failed to allocate memory.
- failed due to timeout.
- failed due to accessing properties.

Also, we compared the standard deviation (SDV) of the netflow features category, as listed in Table 6.22. SDV is a measure of how spread out the numbers are. In this case, the network traffic from *Copperdroid* has a higher standard deviation, which indicates that their data points are spread out over a wider range of values, if compared to *ND-Droid*. Additionally, we performed feature selection by using the *Information Gain* algorithm and ranked the top-10 features for both environments, as illustrated in Figure 6.12.

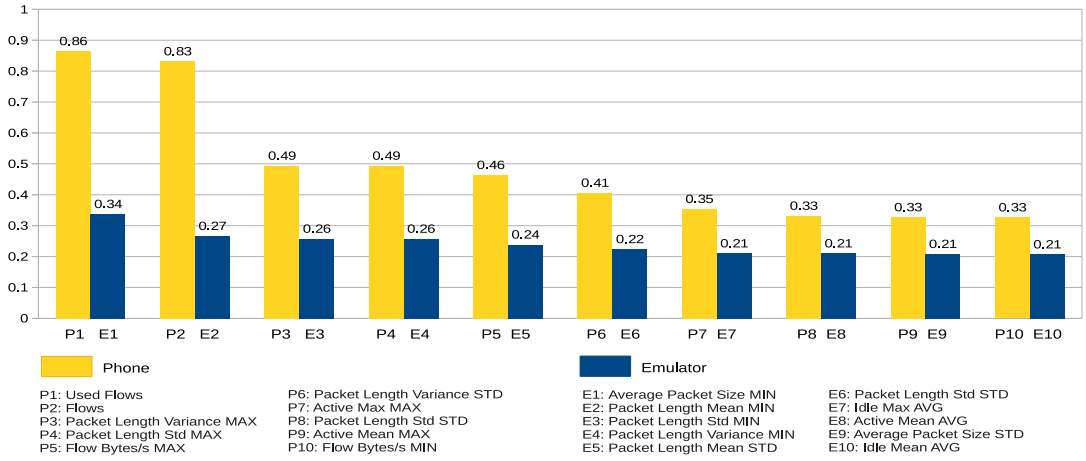

Figure 6.12: Top-10 feature selection via Information Gain

Machine learning detection comparison. Table 6.23 and Table 6.24 present the performance evaluation results of the different machine learning algorithms (for all Netflow features with 3 labels: adware, banking malware, benign). The results shown were obtained from two approaches: 10-fold cross validation and testing on 40% of the samples while 60% were used for training the model. We employed seven common classifiers namely Naive Bayes (NB), Support Vector Machine (SVM), k-Nearest Neighbor (kNN), Multi Layer Perceptron (MLP), ZeroR, Random Forest (RF), and Decision Tree (DT). With 10-fold cross validation, RF outperforms other classifiers in both environments with 99% accuracy on *ND-Droid* and 85% accuracy on *Copperdroid*. However, the train-test set approach shows that MLP yield higher detection rates with 99% accuracy on *ND-Droid* and 79% on *Copperdroid*.

Table 6.22: Standard Deviation of Netflow features

	Traffic features	**ND-Droid Phone**	**Copperdroid Emulator**
1	Flow Duration	3316856.973	16543930.062
2	Flow Bytes	756814.403	244057.757
3	Flow Packets	19039.597	39859.722
4	Packet Length	16.487	103.898
5	Down/Up Ratio	0.1	0.218
6	Packet Size	24.872	113.52
7	Active Time	221009.09	596237.842
8	Idle Time	1153176.767	3138446.267

Table 6.23: Evaluation of the machine learning algorithms with 10-fold cross validation:

Classifier	**10-fold Cross Validation**					
	ND-Droid Phone			Copperdroid Emulator		
	Accuracy	F-Measure	FPR	Accuracy	F-Measure	FPR
NB	96.91	97.00	0.017	58.42	64.00	0.123
SVM	73.88	62.80	0.739	73.88	62.80	0.739
K-NN	95.53	95.50	0.073	83.51	83.10	0.250
MLP	98.97	99.00	0.011	71.82	63.90	0.700
ZeroR	73.88	62.80	0.739	73.88	62.80	0.739
RF	98.97	99.00	0.020	85.22	84.44	0.272
DT	99.31	99.30	0.011	81.10	80.90	0.288

Table 6.24: Evaluation of the machine learning algorithms with Train/Test set

Classifier	**Training-Testing (60-40)**					
	ND-Droid Phone			Copperdroid Emulator		
	Accuracy	F-Measure	FPR	Accuracy	F-Measure	FPR
NB	96.55	96.60	0.009	64.65	68.30	0.136
SVM	71.55	59.70	0.716	71.55	59.70	0.716
K-NN	98.27	98.30	0.024	79.31	78.00	0.268
MLP	99.14	99.10	0.002	79.31	77.90	0.289
ZeroR	71.55	59.70	0.716	71.55	59.70	0.716
RF	98.28	98.30	0.005	77.59	76.50	0.346
DT	99.14	99.10	0.002	73.28	72.60	0.372

Outcome. We accepted the hypothesis that using the real smartphone for malware dynamic analysis can perform better than the emulator. The experimental results show our on-device analyzer achieved a high accuracy of 99% with a very low false positive rate of 0.2% on average. The *Copperdroid* on the other hand, achieved 79% accuracy with 29% FPR. This finding has led us to employ *ND-Droid* in our proposed framework.

6.3 Case Study and Results

Case Study Flow. Case studies are commonly used in social science fields like psychology, political science, and business in order to increase the knowledge about individuals, organizations, and related phenomena. In the context of malware analysis, a case study can help in understanding the malware phenomena in its natural context. It can also help to describe and interpret a complex behavior of malware by extending the knowledge to what is already known from previous work. In this study, we conducted the following two case studies:

1. *Scareware: The case of Android.Spy.277*. *AndroidSpy* first appeared as legitimate apps that offered services like games, wallpapers, photo editing apps. But, once installed, the app transforms into malware and becomes malicious through a backdoor.

2. *Botnet: The Dendroid Android botnet case*. Dendroid was one of the most sophisticated Android malware and was one of the first to bypass Google's Bouncer. It was available for sale in underground malware forums for $300. However, the code is now available for free at Github [4]. We analyzed the code and changed the Command & Control (C&C) server address to our server in order to create a live connections between the malware and the C&C server.

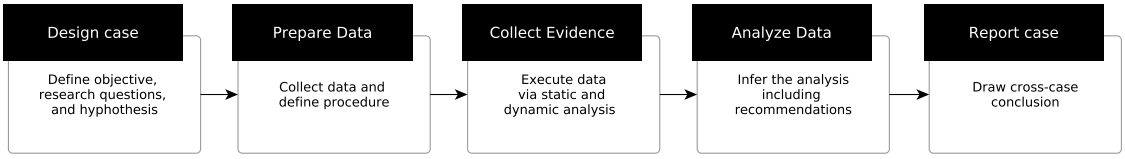

Figure 6.13: The flow of the case studies

The overall design of these case studies is illustrated in Figure 6.13. When conducting a case study, there are five major process steps to be followed [121]:

1. Design case: to define objectives, preliminary research questions, and hypotheses.
2. Prepare data: to define procedures, protocols, and measurement instruments.
3. Collect evidence: to execute the data collected, which can provide ability to address the research questions.
4. Analyze data: to infer the analysis, including recommendations for practice and further research.
5. Report case: to document all the findings, which contain conclusions, implications for practice and future research.

[4] https://github.com/nyx0/Dendroid

6.3.1 The case of *Android.Spy.277*

Overview. In order to evaluate our framework, we conducted a case study of one sophisticated malware family in our dataset (*AndroidSpy*). *AndroidSpy* first appeared as legitimate apps that offered services like games, wallpapers, and photo editing apps. But, once installed, the app transforms into malware and becomes malicious through a backdoor. These apps had been downloaded by almost 3 million users [22]. The attacker can remotely download a malicious APK called *polacin.io* to the victim's device. One infected, the Android device sends a wide array of information about the phone to command and control servers (C&C), including phone IMEI number, email address, SMS messages, and location. What's more, *AndroidSpy* performs an additional malicious act through unwanted advertisements via popups and notification bar. Victims are induced into installing fraudulent apps via fake warning of battery issues that can be solved by downloading fake utilities. By clicking on these fake alerts, for example, *AndroidSpy* brings victims to the landing pages for Android optimization applications such as Turbo Cleaner, SuperB Cleaner (Boost Clean). This behavior reveals that *AndroidSpy* has several layers of attacks to sustain their malicious behavior. To further analyze the stealthiness of this *AndroidSpy*, we analyzed all 6 samples that we have by using three malware scanners: AndroTotal [5], Joe Sandbox [6], and VirusTotal [7].

Design case: Recently, scareware has ultimately become an effective attack method for cybercriminals in getting funds; the cybercriminals make money from the malicious apps by threatening victims to download the apps or convincing them to pay some amount of money for the fake service. In scareware attacks, the actual target is the human where it aims to exploit human emotion, which can cause panic, shock, anxiety, or the perception of a threat in order to persuade users into purchasing the scareware [71].

[5] https://andrototal.org/
[6] https://www.joesandbox.com/
[7] https://www.virustotal.com

The fact that scareware appear as a legitimate program (i.e. contain an actual AntiVirus (AV)) makes it more difficult for the detection system such as Google Bouncer [8] to identify it as scareware. According to statistics, Android scareware was able to dodge detection from Google Bouncer detection for three consecutive years since 2014. For instance, before Google removed the app from the store, Android scareware with *Android.Spy.277* was hidden in more than 100 applications on Google Play and had been infected 2.8 million Android devices [22]. There are many works on malware detection, but in-depth explorations of Android scareware are limited. Thus, it remains unclear why scareware can still bypass the detection system on the app market. Hence, this case aims to tackle this problem by analyzing modern mobile scareware functionality and characteristics.

Prepare data: We collected 150 scareware samples from various sources, as mentioned in Table 5.1. However, we only managed to collect 9 samples of *Android.Spy.277*. To further investigate the relationships between each app in our dataset, we employed Droidkin [73], a lightweight detector of Android apps similarity.

Collect evidence: For this case study, we used a laptop [9] which is connected to a smartphone [10]. We implemented the same approach used in the proposed framework, which is by combining both static and dynamic analysis based specifically on features derived from the string literal (statically via reverse engineering) and network flow (dynamically on smartphones).

Analyze data: In order to evaluate the detection performance, we used the 5-fold cross validation and reported three metrics: FPR, TPR, and ROC area value (the area under the ROC curve is a measure of how well a parameter can distinguish malicious and benign groups) by using the RF classifier on Weka [11].

[8] http://googlemobile.blogspot.ca/2012/02/android-and-security.html
[9] Dell 16GB memory IntelR©CoreTMi7-6600U running Ubuntu 16.04.4
[10] Nexus 5 2.26 GHz running Android 6.0 Marshmallow
[11] https://www.cs.waikato.ac.nz/ml/weka/

Report case: The results depict that most of the AV are not able to detect this type of scareware. Out of 100 AV deployed on these three scanners (VirusTotal, Andrototal, Metadefender), only 25 of them are able to detect *AndroidSpy* as malicious apps (see Table 6.25).

Table 6.25: Comparison results of *AndroidSpy* analysis

Scanner	AV detected	Total AV
VirusTotal	20	61
Andrototal	3	8
Metadefender	2	31

With our approach, we managed to detect *AndroidSpy* accurately (detected 5 apps out of 6) in static analysis. On the other hand, the dynamic analysis shows 99.42% of ROC area value. The true positive rate surpasses 80% when the false positive rate is greater than 60%. Therefore the closer the ROC curve is to the upper left corner (true positive rate), the higher the overall accuracy of the test.

6.3.2 The *Dendroid* Android botnet case

Overview. *Dendroid* was first discovered in early 2014 by Symantec. It was one of the most sophisticated Android malware and a popular crimeware kit that was being advertised in malware forums for sale for $300. The kit helps attackers to automate the malware distribution process where it includes an APK builder and a web-based administration panel that can add the Remote Access Trojan (RAT) functionality. The builder allows the attacker to easily incorporate the *Dendroid* RAT into legitimate Android apps via a repackaging technique, which can bypass the Google's Bouncer.

Moreover, the crimeware kit also includes the following capabilities: a) accessing browser data (history/bookmarks), b) downloading and uploading of files, c) downloading any other accounts (email, social media, VPN) stored on the device, d) controlling the camera and microphone (spy on the user by taking pictures or making audio and video

recordings), e) intercepting and/or blocking SMS texts, f) harvesting call data, and contacts, g) opening installed applications, h) opening a dialogue box to ask for passwords, i) recording any ongoing calls, and j) sending SMS messages to premium-rate numbers without the user's consent. All in all, *Dendroid* was difficult to detect. However, by looking for indications of the network traffic patterns, we can find the best features that can be used in detecting this malware.

Design case: As attempts to thwart cybercrime have intensified, so have innovations in how cybercriminals provision their infrastructure to sustain their activities. Instead of attacking users with malware, cybercriminals today are offering services to automate the malware distribution process by selling the crimeware kit to get more profits. *Dendroid* kit was a popular example that includes advanced attacks capability, which can bypass the detection system (Google bouncer). Thus, this case aims to tackle this problem by scrutinizing the pattern of the network traffic (network flow).

Prepare data: We were able to get the *Dendroid* kit for free from Github [12]. The *Dendroid* crimeware kit comes with a single APK. Thus, for this case study, we focused on the behavior of the network flows. For evaluation, we conducted a comparison analysis between *Dendroid* and benign flows. We ran these two samples on three smartphones (Table 5.2) simultaneously.

Collect evidence: We implemented the same procedure that we used in the proposed framework, i.e. by using the *ND-Droid*. For this case study, we customized the malware scenario and the user interactions to be matched with the attacks capability of *Dendroid*. Additionally, we also customized the source code and changed the C&C server address to our server in order to create a live connections between the malware and the C&C server.

[12]https://github.com/nyx0/Dendroid

Analyze data: In this case study, we analyze the network flow of the *Dendroid* traffic. In order to evaluate the detection performance, we used a 10-fold cross validation and the split of 60%/40% of Test and Train set and reported four metrics: F-measure, accuracy, False Positive Rate (FPR), and ROC area value by using the Random Forest Weka machine learning tool [13].

Report case: We first compared the range values of benign and *Dendroid* traffic in order to understand their behavior, as listed in Table 6.26. The total flow of the *Dendroid* traffic (3814) is larger than the benign traffic (2157 flows), which indicates that the botnet was actively communicated with the C&C server. We then conducted the feature selection by using the Cfs-Subset Evaluator algorithm, as shown in Table 6.27.

Table 6.26: Extracted range values of benign and *Dendroid* traffic

	Traffic features	**Dendroid**	**Benign**
1	Total Flow	3814	2157
2	Flow Duration	3 - 119812209	3 - 119922577
3	Flow Bytes	0 - 20384615	0 - 7666667
4	Flow Packets	0.021 - 666667	0.028 - 666667
5	Packet Length	0 - 12312	0 - 1460
6	Down/Up Ratio	0 - 5	0 - 5
7	Packet Size	0 - 1268	0 - 975
8	Active	0 - 15716321	0 - 11417664
9	Idle	0 - 117376666	0 - 99184862

Table 6.27: Evaluation of feature selection using Cfs-Subset Evaluator algorithm

	10-fold cross validation			**60 Test-set : 40 Train-set**		
	All 84 Features	Selected 76 Features	Selected 3 Features	All 84 Features	Selected 76 Features	Selected 3 Features
Accuracy	98.28	84.15	76.38	93.13	82.47	74.75
F-measure	98.30	84.20	76.40	93.20	82.50	74.80
ROC	99.00	84.30	84.60	93.30	82.50	79.40
FPR	0.029	0.158	0.235	0.066	0.175	0.252

[13] https://www.cs.waikato.ac.nz/ml/weka/

Figure 6.14 depicts the accuracy and FPR of three scenarios of feature selection process: all features (Appendix C), 76 features (removed source IP, source port, destination IP, destination port, Timestamp, protocol, flowID), and 3 features (Fwd packet length, Fwd IAT std, Min sefment size fwd). Based on the analysis, the accuracy is proportional to the number of features where the lower the number of features, the less accurate the detection.

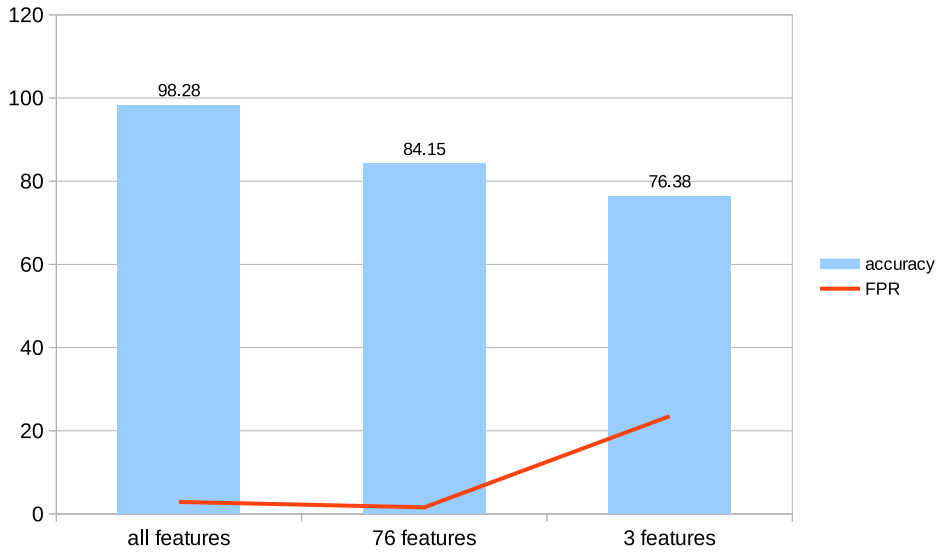

Figure 6.14: Accuracy and FPR results with different feature set (10-fold cross validation)

To further evaluate the feature selection, we implemented another algorithm (information gain) and chose the Top-5 best features according to the ranking score, as shown in Figure 6.15. With this feature-set, we managed to detect the botnet traffic accurately with 91% ROC (Table 6.28).

Table 6.28: Evaluation of feature selection using Information Gain algorithm

	10-fold cross validation	**60 Test-set : 40 Train-set**
Accuracy	84.01	83.17
F-measure	83.90	83.10
ROC	91.30	90.60
FPR	0.195	0.207

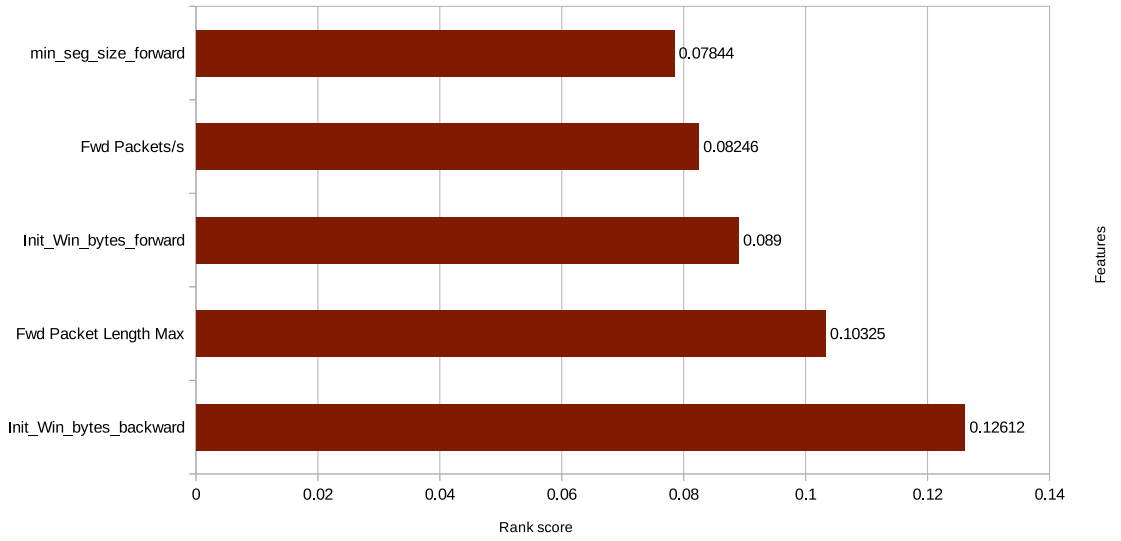

Figure 6.15: Top-5 features via information gain algorithm

6.4 Evaluation and Results

This section presents the evaluation of our proposed framework, which includes the evaluation of the hybrid modules, the comparative evaluation between our framework and other works, and the evaluation of system performance and scalability.

6.4.1 Evaluation of Android financial malware detection

Accuracy and FPR. We created a new dataset with 5,330 benign and 1,758 malware (combination of D4: Benign and D5: Taxonomy), and conducted a validation approach to evaluate our proposed framework. Out of 1.7k malware samples, 532 of them were miss-classified by the string analyzer and fell below the static threshold score of 97% accuracy and ROC. These 532 samples were then analyzed dynamically on *ND-Droid*. Table 6.29 shows the accuracy and FPR of static and dynamic modules with the four levels of detection: malware binary, category, sub-category, and family. Both static and dynamic analysis yield a high accuracy with more than 90% in detecting malicious apps from the benign.

Table 6.29: Evaluation results of the proposed framework with 5-fold cross validation

Analysis	Binary Detection (2 classes)		Category Classification (6 classes)		Sub-Category Classification (11 classes)		Family Characterization (53 classes)	
	Accuracy	FPR	Accuracy	FPR	Accuracy	FPR	Accuracy	FPR
Static	94.49	0.060	87.81	0.163	86.63	0.217	86.50	0.204
Dynamic	97.66	0.023	85.14	0.038	81.03	0.036	65.23	0.056

ROC area. For evaluating the malware classification, we reported the ROC value for each malware family in the fourth level of malware detection via the 5-fold cross validation (considering some of the total family is less than 10 samples). Table 6.30 lists all 52 families (labelled with their category, sub-category, and family) with the average of 98% ROC. The highest ROC is achieved by the *ransom_encrypt_pletor* family with 100% value.

6.4.2 Comparative evaluation of Android malware detection

We conducted a comparative evaluation between our financial malware detection framework and other related frameworks. In this case, we employed the same datasets used by Killam et al. [116] in conducting the static analysis.

The evaluation result is listed in Table 6.31. Similarly, we employed the same datasets used by Aqil et al. [166] and Copperdroid [138] in performing the dynamic analysis. The results of the evaluation are shown in Table 6.32. Overall, our proposed framework achieved better accuracy if compared to the previous works with almost 99% accuracy for both modules.

Table 6.30: Evaluation of Android financial malware detection based on ROC value (5-fold cross validation)

Sub-Category-Family	ROC Area	FPR
adware_icon_mobidash	97.70%	0.60%
adware_icon_selfmite	97.00%	0.00%
adware_notification_chinese	94.60%	0.50%
adware_notification_dowgin	97.20%	0.90%
adware_notification_ewind	98.20%	0.80%
adware_notification_feiwo	97.00%	1.50%
adware_notification_gooligan	92.50%	0.80%
adware_notification_shuanet	97.10%	0.80%
adware_notification_youmi	94.10%	0.20%
bankactive_bankbot	95.40%	1.60%
bankactive_binv	59.50%	0.00%
bankactive_citmo	97.90%	0.00%
bankactive_sandroid	98.50%	1.00%
bankactive_smsspy	98.70%	0.50%
bankactive_spitmo	98.90%	0.60%
bankactive_zertsecurity	97.10%	0.10%
bankactive_zitmo	98.70%	0.80%
bankpassive_fakebank	97.40%	0.60%
bankpassive_wroba	96.80%	0.50%
benign	99.70%	10.50%
ransomencrypt_pletor	100.00%	0.00%
ransomencrypt_ransombo	99.40%	0.60%
ransomencrypt_svpeng	97.90%	0.40%
ransomencrypt_wannalocker	98.80%	0.80%
ransomlock_charger	99.40%	0.90%
ransomlock_jisut	97.00%	0.30%
ransomlock_koler	96.70%	0.70%

Sub-Category-Family	ROC Area	FPR
ransomlock_lockerpin	81.80%	0.30%
ransomlock_porndroid	96.40%	0.90%
ransomlock_simplocker	99.50%	0.60%
scareapp_androidspy	95.30%	0.10%
scareapp_fakeappAL	98.30%	0.50%
scareapp_fakeapp	89.90%	0.40%
scareapp_fakeflash	95.60%	0.20%
scareservice_androiddefender	99.20%	0.70%
scareservice_avforandroid	96.20%	0.80%
scareservice_avpass	96.00%	1.70%
scareservice_fakeav	95.90%	1.80%
scareservice_fakejoboffer	99.30%	0.80%
scareservice_faketaobao	99.00%	0.70%
scareservice_penetho	87.70%	0.90%
scareservice_virusshield	94.00%	0.60%
smsfraud_beanbot	90.60%	0.10%
smsfraud_fakemart	91.90%	0.40%
smsfraud_jifake	89.30%	0.10%
smsfraud_mazarbot	89.30%	0.00%
smsfraud_zsone	92.70%	0.10%
smsphishing_biige	94.60%	0.80%
smsphishing_fakeinst	98.40%	0.10%
smsphishing_fakenotify	99.30%	0.70%
smsphishing_nandrobox	99.20%	0.50%
smsphishing_plankton	98.90%	0.70%
smsphishing_smssniffer	99.40%	0.50%
Average	**97.70%**	**0.56%**

Table 6.31: Comparison between our framework and other work on static analysis

	Proposed Framework	Killam (2015)
Accuracy	98.90	98.88
F-Measure	98.68	98.38
FPR	0.003	0.008

Table 6.32: Comparison between our framework and other works on dynamic analysis

	Proposed Framework	Copperdroid (2017) [138]	Aqil et al. (2018) [166]
Accuracy	98.97	85.22	98.67
F-Measure	99.00	84.44	98.70
FPR	0.020	0.272	0.033

6.4.3 Evaluation on system performance (scalability)

We measured the performance of our system by calculating the average time of the training and testing process. In this case, we categorized our dataset into five data ratio as explained in Table 6.33 and Table 6.34. Additionally, for the static analysis, we included the performance evaluation of each 4-gram, as presented in Figure 6.16 and Figure 6.17. Overall, the results of both modules indicate that our system is scalable. Although we increased the size of samples, it results in increased performance (in a manner proportional to the samples added).

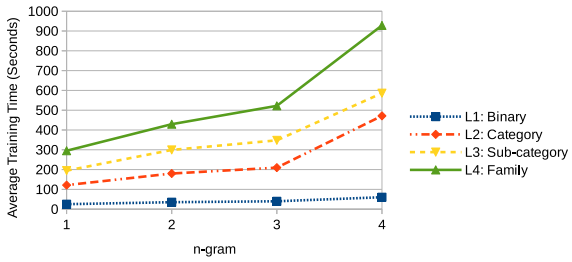

Figure 6.16: Average training time

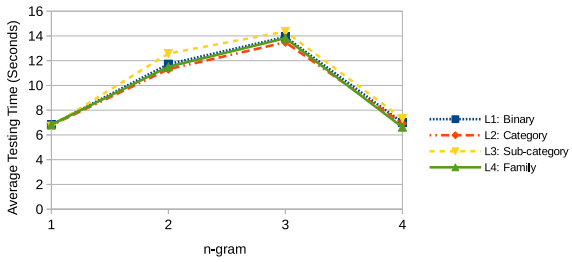

Figure 6.17: Average testing time

Table 6.33: Performance evaluation on static analysis

Sample size (apk)	Data Ratio (Benign: Malware)	Total String Length	Training Time (s)	Testing Time (s)	Accuracy (%)
1 000	50:50	1 294 371	39.90	13.92	94.49
1 500	60:40	6 826 461	80.54	37.89	97.38
3 830	70:30	6 768 468	172.55	78.56	97.81
4 970	80:20	3 623 790	101.97	40.87	98.99
5 730	90:10	3 667 753	157.73	56.76	98.90

Table 6.34: Performance evaluation on dynamic analysis

Sample size (apk)	Data Ratio (Benign: Malware)	Total Flows	Training Time (s)	Testing Time (s)	Accuracy (%)
1 000	50:50	2 147 832	0.20	0.01	97.92
1 250	60:40	2 246 772	0.23	0.01	97.20
1 700	70:30	2 642 448	0.33	0.01	98.38
2 500	80:20	3 068 584	0.55	0.02	98.80
5 000	90:10	4 948 755	1.29	0.04	98.85

6.5 Summary

We presented the results of six sets of experiments and two sets of case studies that formed our intelligent framework for malware detection. Based on the results, our framework is capable of classifying malware based on a defined taxonomy presented in Chapter 3 accurately with more than 90% accuracy and almost 98% ROC value. What is more, the comparative evaluation results show that our framework is better than previous works with almost 99% accuracy in detecting malicious apps from benign (malware binary). The detection framework has four levels of detection capability with acceptable FPR value: 2% when detecting malicious Android apps, 4% when classifying its category and sub-category, and 6% when charecterizing its family. In addition, the case study infers that our proposed framework is able to detect the advanced and sophisticated malware family such as *Android.spy scareware* and *Dendroid botnet* accurately with 99% and 91% ROC values, respectively.

Chapter 7

Conclusion And Future Work

The measure of greatness in a scientific idea is the extent to which it stimulates thought and opens up new lines of research.

—Paul A.M. Dirac

The cybersecurity researchers today have turned their attention to the anomaly-based detection of Android malware: to automatically learn data trends and alert anomalies with machine learning. This approach suffers from an inaccurate evaluation and comparison due to the lack of adequate datasets, which resulted in unreliable outputs for the real-world deployment. Due to the time-consuming installation processes within smartphones, such as the sluggish interaction with malware and the limited guidelines for the event processes, most of the available datasets are crafted mainly for static analysis; and those created for dynamic analysis are installed on an emulator or sandbox.

Nevertheless, sophisticated malware can bypass these approaches as malware authors have employed obfuscation and polymorphism techniques to thwart detection. This also includes a wide range of anti-emulator techniques, where the malware programs attempt to hide their malicious activities by detecting the emulator. These deficiencies are the major reasons why a perfect dataset is yet to exist. Thus, researchers must resort to datasets that are often sub-optimal.

In this thesis, a systematic approach to generate the required datasets is presented to address the need of using smartphones instead of emulators. In this regard, an automated dynamic analysis system running on smartphones is developed to generate the desirable dataset in a testbed environment. Recognizing the difficulty of analyzing Android malware, we sought to understand trends and relationships between Android malware families. We investigated a large volumes of Android malware samples collected from various sources. Among other outcomes, we defined new knowledge in Android financial malware by developing a taxonomy of Android financial malware attack.

Due to the increased number of financial-related malware, the security community today has turned their attention to Android financial malware. What constitutes Android financial malware is still ambiguous. A comprehensive understanding of the existing Android financial malware attacks supported by a unified terminology is necessarily required for the deployment of reliable defence mechanisms against these attacks. In this thesis, we addressed this issue and devised a taxonomy of Android financial malware attacks. By devising the proposed taxonomy, we intend to give researchers a better understanding of these attacks, explore Android financial malware characteristics, and provide a foundation for organizing research efforts within this specific field. The study also identified a number of challenges related to Android financial malware, which can create opportunity for future research. To correlate the generated dataset and the proposed taxonomy, a hybrid framework for malware detection is presented. We proposed a novel combination of both static and dynamic analysis based specifically on features derived from text strings (statically via reverse engineering) and network flow (dynamically on smartphones). We conducted six sets of experiments and two sets of case studies to construct the best model for identifying Android financial malware. Specifically, a dataset containing 5,000 samples is used to verify the performance of the proposed method. The experimental results show that the proposed method with a Random Forest classifier achieves high accuracy of over 90% for all three scenarios with a very low false positive rate of 0.046 on average.

As cybercrime is growing in both frequency and sophistication, so do the security challenges faced by researchers. This never-ending battle between cybersecurity organizations and cybercriminals has always been like a game of cat and mouse. To facilitate research, the thesis introduced three innovations in the detection of mobile malware. First, a new architecture of dynamic analyzer called *ND-Droid* is designed to perform an automated dynamic analysis on smartphones instead of an emulator. The automated system is vital for the effectiveness of the dataset in terms of realism, total capture, feature completeness, and evaluation capabilities. The second innovation, the comprehensive taxonomy of Android financial malware, provides a guideline to better understand malware attacks, which can save the time of malware analysts in the malware-triage process. Last but not least, an intelligent framework for malware detection based on a defined taxonomy is presented, which is capable of performing three levels of detection scenarios: detecting malicious Android apps, classifying Android apps with respect to malware category and sub-category, and characterizing Android apps according to malware family. A comprehensive taxonomy combined with the intelligent framework can effectively save the time of malware analysts to correlate various symptoms of malicious behavior; this combination provides a systematic overview of malware capabilities which can help analysts to prioritize which malware to be scrutinized for further inspection.

7.1 Findings Summary

Referring back to Chapter 1, the following Table 7.1 shows the findings and justification of the research objectives and the research questions. In summary, all of these findings answered the research questions which indicate that the objectives of this research have been achieved.

Table 7.1: Research question and justification

Experiment/Research Question	Result
E1: Data evaluation based on taxonomy RQ1: What constitutes Android financial malware?	Based on the behavioral analysis, we classified the behaviors of Android financial malware into specific types of attacks and developed a taxonomy as a guide for other researchers. We defined Android financial malware and classified it into 5 categories: adware, banking malware, ransomware, scareware, SMS malware.
E1: Data evaluation based on taxonomy RQ2: Does financial malware exhibit unique characteristics that can be used to differentiate it from other malware types?	True. Android financial malware especially banking malware is more targeted than the general malware. e.g., Banking malware ranked first in terms of the total number of malware attacks, scoring about 80% for both geographical location attacks, and information theft. This is followed by scareware category, SMS malware, ransomware, adware. Our evaluation shows that in addition to the BOOT and SMS (general malware), system events also register to CALL and PKG (financial malware).
E2: The effectiveness of static vs. dynamic modules E3: Analysis of feature engineering E4: Threshold analysis of decision-making module RQ4: What features should be extracted to identify advanced malware?	String and network traffic are the best features that can be used to identify advanced malware. The experimental results revealed that string feature outperformed other features (metadata, opcode, bytecode) with 97% accuracy. Moreover, the results with the ND-Droid phone illustrated that the network traffic feature outperformed the system call with 99% accuracy. By evaluating four other datasets using string analysis, we defined the score for the static analysis to be 97.67% accuracy. Hence, for the static analysis, a score less than 97% can be used as the threshold for the next stage, which is the dynamic analysis.
E5: Evaluation of detection accuracy of the proposed framework RQ5: Which classifiers perform well in classifying malware?	Random forest classifier performs best. The experimental results show that the proposed method with Random Forest classifier achieves high accuracy of over 90% for all three scenarios with a very low false positive rate of 0.046 on average.
E6: Comparative analysis of state of the art Copperdroid Emulator RQ3: Does using the real smartphone for malware dynamic analysis perform better than the emulator?	True. We accepted the hypothesis that using the real smartphone is better than the emulator. The experimental results show our on-device analyzer, ND-Droid with MLP and RF classifiers achieved high accuracy of 99% with a very low false positive rate of 0.2% on average. The Copperdroid on the other hand, achieved 79% accuracy with 29% FPR. This finding has led us to employ ND-Droid in our proposed framework.

7.2 Challenges and Limitation

The study of Android financial malware is dynamic and stimulating. As this is the first study of its type to systematically categorize Android financial malware based on a taxonomy, researchers not only have a new opportunity for research but also a number of challenges. Below are some important challenges in relation to the baseline of Android financial malware field:

1. **Lack of dataset**: The top challenge is a lack of sufficient data. Lack of data is the common problem for all researchers in academia and the study of Android financial malware is no exception. Data is the key for any successful modeling and detection system. In order to ensure that the detection algorithm is reliable, building the malware detection model requires two types of dataset: the training and testing dataset. There exist several public datasets [27, 26, 25] but they are not related to Android financial malware. To tackle this issue, we collected malware samples from various resources and manually checked if the collected samples are financial malware according to the proposed taxonomy. In addition, the naming convention of malware labelling (i.e. family name) is inconsistent among both academic and industry fields. The inconsistent labelling between different anti-virus vendors and researchers leads to confusion and is time-consuming for reorganization. Researchers have to compare the malware labelling from several anti-virus vendors and follow the majority numbers of the most frequent label of the specific malware family. This technique is completely manual and can lead to errors, which can affect the accuracy of malware labelling. Just recently, however, there is a new tool called *Euphony* [80] that can be used to harmonized the unification labeling based on VirusTotal.

2. **Lack of systematic approach for malware assessment**: There exist a number of metrics developed for malware threat assessment such as vulnerability rating, risk assessment, and incident and impact metrics.

There are no standard metrics for assessing malware behavior and evaluating its complexity. Researchers need these metrics to overcome the weaknesses of the current detection system. As the capability of malware is dynamic and becoming more sophisticated, having a standard metrics can help researchers to catch up with the malware trends automatically by defining a formula (i.e. malware complexity scoring formula) which can help researchers with further analysis. In 2016, Maasberg et al. [101] also highlighted this issue giving a set of measures to quantify malware threats systematically using weights and ratings by focusing on four elements of malware: propagation, characteristics, attribution, impact, and associated dimensions. Although the proposed solution is not fully automated and limited to the zero-day malware (unknown and unidentified malware), it provides a valuable insight into potential capabilities of measuring malware. Significantly, understanding to what degree of sophistication (weight, rating, score) the Android financial malware exhibits can help researchers provide an accurate and cost-effective mitigation option and strategy.

3. **Hybrid of malware**: The diversity of mobile platforms force malware writers to boost their chances of infection by targeting several attack vectors. The recent samples show that the majority of malware is now hybrid with the banking malware category. For instance, modern mobile banking malware is not only capable of stealing banking information, but can also capture SMS messages, record videos of the victim's screen and upload the videos, and even lock the mobile devices and encrypt documents. Moreover, adware and scareware can also be used to compliment banking malware to gain information associated with financial transactions [57]. In that case, it is not easy to classify the malware into a specific category according to the proposed taxonomy. Researchers have to do malware triage and analysis in order to measure the most dominant category between the mix of malware categories.

Importantly, the presence of metrics can help to prioritize the most dominant category in an automated way. An accurate dominant malware category acts as an aid to more accurate detection and can help to facilitate the strategy for the mitigation system.

4. **Use of obfuscation**: Today, the use of obfuscation is prevalent in mobile malware. Obfuscation aims to disguise the malware code to make reverse engineering more challenging. Obfuscation refers to the code masking strategy of changing the content of the application (Dalvik Executable .dex files and/or AndroidManifest .xml files) but preserving its original functionality. The obfuscation techniques employed by Android malware typically fall into one of the following categories: loading native libraries, hiding exploits in package assets, using encryption, truncating URLs, injecting malicious bytecode, manipulating the DEX file format to hide methods, and customizing the output of encryption to hide an APK. As such, obfuscation affects the accuracy of malware detection approaches significantly, which also slows down the discovery and detection by security products. Researchers have to consider an effective way to evaluate the resilience of malware detection technique with regard to the obfuscation behavior in order to combat this issue.

5. **Lack of collaboration with third-party organizations**: In contrast to non-financially related malware, financial malware requires an interaction with both the victims and the third-party organizations (i.e.financial institutions, SMS centre). For instance, in order to successfully launch their attacks, banking malware needs to monitor the communication between the banking system and the victim. Even after successfully infecting the victim's phone, the cybercriminals still have to collect the victim's banking information (i.e. TAN information) in order to be able to intercept any transactions with the banks. Due to the TAN expiry duration, this theft happens in real time according to the victim's transaction request. As such, in order to combat financial malware, researchers have to understand how the mobile banking attacks

are launched and how the malware is communicated with the bank in real time. This method is significant for the malware response and mitigation strategy. Due to policy and privacy issues, a collaboration with this third-party is difficult to establish. As researchers do not have sufficient information and access to banking systems, the research scope on malware incident response and mitigation strategy is limited. This situation leads researchers to focus more on the analysis and detection part instead of the incident response and mitigation part.

6. **Exploitation of human emotion**: One of the major differences between malware and financial malware is the exploitation of human emotion. Most of the financial malware in accordance with our taxonomy is associated to a psychological game. Psychology plays an important role in almost all aspects of financial malware particularly in ransomware and scareware. From the moment an attack is launched, threatening the victims, to the moment the victims pays, or refuses to pay. The malware authors use psychological tactics designed to create a sense of urgency and to exploit human emotions especially anxiety, panic and fear by assigning a warning and deadline. Koler ransomware, for example, presents fake FBI warnings accusing users of viewing of pornography and demands a ransom payment within 48 hours threatening that the recovery keys would be unavailable after that. As a result, if victims got infected with this type of malware, they are willing to pay because they afraid that their activities would be put under a microscope and their reputation would be ruined. In order to increase the capability of malware detection and mitigation, this attribute of human emotion should be taken into account when developing techniques for Android financial malware detection. Study of malware behavior towards human emotion can be conducted in order to measure the malware metrics in accordance with human behavior. This is important to improve understanding and instill awareness of cybersecurity, which can save many people from becoming cyber victims.

7. **Speed of malware evolution**: Android financial malware is evolving quite rapidly. The research pace, however, is not accelerating (as much as malware); researchers are still working on developing an intelligent system and methods to tackle this issue. Within a short period of time (i.e. 5 years) the evolution shows that advanced malware capabilities are increasing in the number of unique variants. Researchers can track the speed of evolution with the following four stages:

   (a) 1st stage(2010): the introduction period of Android banking malware. The evolution started primarily with the release of the traditional desktop banking malware in the mobile versions. e.g., Zitmo, Spitmo, Citmo.

   (b) 2nd stage (2013): these malware families feature simplistic modifications, re-compiled the source code with improved infection and distribution strategies, e.g., ZertSecurity, SMSspy, Fake-Bank.

   (c) 3rd stage (2014): during this year we saw malware emerging with innovative techniques based on new infection strategies and payloads, e.g., Wroba, Sandroid, Binv.

   (d) 4th stage (2015 - onwards): this is the most advanced stage that covers the recently discovered malware such as Bankbot and Svpeng. It is capable of employing advanced techniques for infection and distribution, which are combined with the ransomware technique.

7.3 Future Work

The findings of this research can be applied to the mitigation domain of Android financial malware. For future work, we aim to work on both sides: researchers and users. We aim to assist research work in academia and industry, and also to help protect the users.

This section presents the different alternatives that can be conducted for future research:

- One area of future work is applying a broader range of features for dynamic analysis including the memory, system calls, API calls, and phone statistics, as illustrated in Figure 7.1. These features need to be analyzed simultaneously on *ND-Droid*.

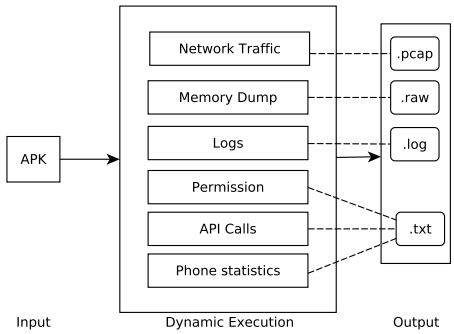

Figure 7.1: Features to be added on *ND-Droid*

- One of the areas that benefits researchers are the web-based Graphical User Interface (GUI) of *ND-Droid*. The web-based service that can link with our on-device analyzer can help researchers to run a dynamic analysis effectively. The web-based service can be configured with user interaction and user profile packages.

- Another interesting area that can be investigated in the future is an on-device mitigation system. Based on our findings, we can develop an Android app that can monitor the user's smartphone. This app acts as a defender for the user, which can be used as alternative to antivirus software.

References

[1] *The rise of android drive-by downloads*, http://0x4d31.blogspot.ca/2012/05/rise-of-android-drive-by-downloads.html, accessed April 1, 2018.

[2] *Virus total*, https://www.virustotal.com/en/, accessed August 1, 2017.

[3] *Android botnet targets middle east banks*, http://krebsonsecurity.com/2014/04/android-botnet-targets-middle-east-banks/, accessed August 13, 2015.

[4] *A timeline of mobile botnets*, https://www.virusbtn.com/virusbulletin, accessed August 13, 2015.

[5] *Securelist: Zeus-in-the-mobile for android*, http://securelist.com/blog/virus-watch/29258/zeus-in-the-mobile-for-android-10/, accessed August, 2015.

[6] *Net losses:estimating the global cost of cybercrime*, http://www.mcafee.com/ca/resources/reports/rp-economic-impact-cybercrime2.pdf, accessed Jan, 2016.

[7] *The first mobile encryptor trojan*, https://securelist.com/blog/mobile/63767/the-first-mobile-encryptor-trojan/, accessed Jan, 2017.

[8] *Mobile crypto-ransomware*, https://blog.avast.com/2015/02/10/mobile-crypto-ransomware-simplocker-now-on-steroids/, accessed Jan, 2017.

[9] *Mobile ransomware: Status quo*, https://blog.fortinet.com/2014/06/25/mobile-ransomware-status-quo, accessed Jan, 2017.

[10] *Scarepakage android ransomware pretends to be fbi porn warning*, https://www.theguardian.com/technology/2014/jul/17/scarepakage-android-ransomware-porn-fbi, accessed Jan, 2017.

[11] *Sms trojan yzhcsms found in android market and third party stores*, http://forums.juniper.net/t5/Security-Now/SMS-Trojan-YZHCSMS-Found-in-Android-Market-and-Third-Party/ba-p/132963, accessed Jan, 2017.

[12] *Android.BankBot in virus library*, http://vms.drweb.com/search/?q=Android.BankBot&lng=en/, accessed July 11, 2015.

[13] *Current android malware*, http://forensics.spreitzenbarth.de/android-malware/, accessed July 11, 2015.

[14] *Security threat trends 2015*, https://www.sophos.com/en-us/threat-center/medialibrary/PDFs/other/sophos-trends-and-predictions-2015.pdf, accessed July 11, 2015.

[15] *Current android malware*, https://forensics.spreitzenbarth.de/android-malware/, accessed July 11, 2017.

[16] *IT threat evolution in q3 2014*, https://securelist.com, accessed July 13, 2015.

[17] *Security affair 2015: 11 percent of mobile banking apps includes harmful code*, http://securityaffairs.co/wordpress/33212/malware/mobile-banking-apps-suspect.html, accessed June 12, 2015.

[18] *The case study as research methodology*, `http://www.ischool.utexas.edu/~ssoy/usesusers/l391d1b.htm`, accessed Mar, 2017.

[19] *Android malware toolkit*, http://dunkelheit.com.br/amat/analysis/index en.php, accessed Mar, 2018.

[20] *Google bouncer: Protecting the google play market*, `https://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/`, accessed Mar, 2018.

[21] *Koodous apk repository*, https://koodous.com/, accessed Mar, 2018.

[22] *Scareware app downloaded over a million times from google play*, http://researchcenter.paloaltonetworks.com/2015/01/scareware-app-downloaded-million-times-google-play/, accessed Mar 30, 2017.

[23] *Foresafe dynamic analysis*, `http://www.foresafe.com/`, accessed May 9, 2018.

[24] *Financial malware attacks*, https://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/smartphone-security-1/top-ten-risks/financial-malware-attacks, accessed Nov 7, 2015.

[25] *Android malware*, http://amd.arguslab.org/sharing, accessed September 7, 2017.

[26] *Android malware dataset*, https://github.com/ashishb/android-malware, accessed September 7, 2017.

[27] *Malware sample sources for researchers*, https://zeltser.com/malware-sample-sources/, accessed September 7, 2017.

[28] Mansour Ahmadi, *Modeling neglected functions of android applications to effectively detect malware*, (2017).

[29] Abdullah J Alzahrani and Ali A Ghorbani, *Sms mobile botnet detection using a multi-agent system: research in progress*, Proceedings of the 1st International Workshop on Agents and CyberSecurity, ACM, 2014, p. 2.

[30] Abdullah J Alzahrani, Natalia Stakhanova, Hugo Gonzalez, and Ali A Ghorbani, *Characterizing evaluation practices of intrusion detection methods for smartphones*, Journal of Cyber Security and Mobility (2014).

[31] Mohammed K Alzaylaee, Suleiman Y Yerima, and Sakir Sezer, *Emulator vs real phone: Android malware detection using machine learning*, Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics, ACM, 2017, pp. 65–72.

[32] Gharib Amirhossein, *Dna-droid: A real-time android ransomware detection framework*, 2017.

[33] Nicoló Andronio, Stefano Zanero, and Federico Maggi, *Heldroid: Dissecting and detecting mobile ransomware*, International Workshop on Recent Advances in Intrusion Detection, Springer, 2015, pp. 382–404.

[34] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens, *Drebin: Effective and explainable detection of android malware in your pocket.*, Ndss, vol. 14, 2014, pp. 23–26.

[35] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel, *Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps*, Acm Sigplan Notices **49** (2014), no. 6, 259–269.

[36] N Asokan, Lucas Davi, Alexandra Dmitrienko, Stephan Heuser, Kari Kostiainen, Elena Reshetova, and Ahmad-Reza Sadeghi, *Mobile platform security*, Synthesis Lectures on Information Security, Privacy, & Trust **4** (2014), no. 3, 1–108.

[37] Alexandre Bartel, Jacques Klein, Yves Le Traon, and Martin Monperrus, *Dexpler: converting android dalvik bytecode to jimple for static analysis with soot*, Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis, ACM, 2012, pp. 27–38.

[38] Michael Becher, Felix C Freiling, Johannes Hoffmann, Thorsten Holz, Sebastian Uellenbeck, and Christopher Wolf, *Mobile security catching up? revealing the nuts and bolts of the security of mobile devices*, Security and Privacy (SP), 2011 IEEE Symposium on, IEEE, 2011, pp. 96–111.

[39] Alastair R Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan, *Mockdroid: trading privacy for application functionality on smartphones*, Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, ACM, 2011, pp. 49–54.

[40] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak, *An android application sandbox system for suspicious software detection*, Malicious and unwanted software (MALWARE), 2010 5th international conference on, IEEE, 2010, pp. 55–62.

[41] Abhijit Bose, Xin Hu, Kang G Shin, and Taejoon Park, *Behavioral detection of malware on mobile handsets*, Proceedings of the 6th international conference on Mobile systems, applications, and services, ACM, 2008, pp. 225–238.

[42] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani, *Crowdroid: behavior-based malware detection system for android*, Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM, 2011, pp. 15–26.

[43] Mahinthan Chandramohan, Hee Beng Kuan Tan, Lionel C Briand, Lwin Khin Shar, and Bindu Madhavi Padmanabhuni, *A scalable approach for malware detection through bounded feature space behavior modeling*, Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, IEEE, 2013, pp. 312–322.

[44] Thomas M Chen and Cyrus Peikari, *Malicious software in mobile devices*, Handbook of Research on Wireless Security **1** (2008), 1–10.

[45] Zhenxiang Chen, Hongbo Han, Qiben Yan, Bo Yang, Lizhi Peng, Lei Zhang, and Jin Li, *A first look at android malware traffic in first few minutes*, Trustcom/BigDataSE/ISPA, 2015 IEEE, vol. 1, IEEE, 2015, pp. 206–213.

[46] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner, *Analyzing inter-application communication in android*, Proceedings of the 9th international conference on Mobile systems, applications, and services, ACM, 2011, pp. 239–252.

[47] Byungha Choi, Sung-Kyo Choi, and Kyungsan Cho, *Detection of mobile botnet using vpn*, Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on, IEEE, 2013, pp. 142–148.

[48] Mauro Conti, Bruno Crispo, Earlence Fernandes, and Yury Zhauniarovich, *Crêpe: A system for enforcing fine-grained context-related policies on android*, Information Forensics and Security, IEEE Transactions on **7** (2012), no. 5, 1426–1438.

[49] Jonathan Crussell, Clint Gibler, and Hao Chen, *Attack of the clones: Detecting cloned applications on android markets*, European Symposium on Research in Computer Security, Springer, 2012, pp. 37–54.

[50] A DESNOS and P LANTZ, *Droid box: an android application sandboxfor dynamic analysis*.

[51] Gianluca Dini, Fabio Martinelli, Andrea Saracino, and Daniele Sgandurra, *Madam: A multi-level anomaly detector for android malware.*, MMM-ACNS, vol. 12, Springer, 2012, pp. 240–253.

[52] William Enck, *Defending users against smartphone apps: Techniques and future directions*, International Conference on Information Systems Security, Springer, 2011, pp. 49–70.

[53] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth, *Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones*, ACM Transactions on Computer Systems (TOCS) **32** (2014), no. 2, 5.

[54] William Enck, Damien Octeau, Patrick D McDaniel, and Swarat Chaudhuri, *A study of android application security.*, USENIX security symposium, vol. 2, 2011, p. 2.

[55] William Enck, Machigar Ongtang, and Patrick McDaniel, *Mitigating android software misuse before it happens*, (2008).

[56] Emre Erturk, *A case study in open source software security and privacy: Android adware*, Internet Security (WorldCIS), 2012 World Congress on, IEEE, 2012, pp. 189–191.

[57] Emre Erturk, *Two trends in mobile security: Financial motives and transitioning from static to dynamic analysis*, CoRR **abs/1504.06893** (2015).

[58] Wenhao Fan, Yaohui Sang, Daishuai Zhang, Ran Sun, and Yuan'an Liu, *Droidinjector: A process injection-based dynamic tracking system for runtime behaviors of android applications*, Computers & Security **70** (2017), 224–237.

[59] Parvez Faruki, Shweta Bhandari, Vijay Laxmi, Manoj Gaur, and Mauro Conti, *Droidanalyst: Synergic app framework for static and dynamic app analysis*, Recent Advances in Computational Intelligence in Defense and Security, Springer, 2016, pp. 519–552.

[60] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan, *Android security: a survey of issues, malware penetration, and defenses*, IEEE communications surveys & tutorials **17** (2015), no. 2, 998–1022.

[61] Parvez Faruki, Vijay Ganmoor, Vijay Laxmi, Manoj Singh Gaur, and Ammar Bharmal, *Androsimilar: robust statistical feature signature for android malware detection*, Proceedings of the 6th International Conference on Security of Information and Networks, ACM, 2013, pp. 152–159.

[62] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Fairuz Amalina, Ra'uf Ridzuan Ma'arof, and Shahaboddin Shamshirband, *A study of machine learning classifiers for anomaly-based mobile botnet detection*, Malaysian Journal of Computer Science **26** (2014), no. 4.

[63] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner, *A survey of mobile malware in the wild*, Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM, 2011, pp. 3–14.

[64] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken, *Apposcopy: Semantics-based detection of android malware through static analysis*, Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2014, pp. 576–587.

[65] Matt Fredrikson, Somesh Jha, Mihai Christodorescu, Reiner Sailer, and Xifeng Yan, *Synthesizing near-optimal malware specifications from suspicious behaviors*, Security and Privacy (SP), 2010 IEEE Symposium on, IEEE, 2010, pp. 45–60.

[66] Fsecure, *Another reason 99% of mobile malware targets androids*, https://safeandsavvy.f-secure.com/2017/02/15/another-reason-99-percent-of-mobile-malware-targets-androids/ (2018).

[67] Adam P Fuchs, Avik Chaudhuri, and Jeffrey S Foster, *Scandroid: Automated security certification of android*, Tech. report, 2009.

[68] P. Garner, I. Mullins, R. Edwards, and P. Coulton, *Mobile terminated sms billing:exploits and security analysis*, Third International Conference on Information Technology: New Generations (ITNG'06), April 2006, pp. 294–299.

[69] Raj Samani Gary Davis, *Mobile threat report*, `https://www.mcafee.com/ca/resources/reports/rp-mobile-threat-report-2018.pdf`, accessed April 11, 2018.

[70] Gerard D Gil, Arash H Lashkari, M Mamun, and Ali A Ghorbani, *Characterization of encrypted and vpn traffic using time-related features*, Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016), 2016, pp. 407–414.

[71] Jim Giles, *Scareware: the inside story*, New Scientist **205** (2010), no. 2753, 38–41.

[72] Ian Goldberg, David Wagner, Randi Thomas, Eric A Brewer, et al., *A secure environment for untrusted helper applications: Confining the wily hacker*, Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography, vol. 6, 1996, pp. 1–1.

[73] Hugo Gonzalez, Natalia Stakhanova, and Ali A Ghorbani, *Droidkin: Lightweight detection of android apps similarity*, International Conference on Security and Privacy in Communication Systems, Springer, 2014, pp. 436–453.

[74] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang, *Riskranker: scalable and accurate zero-day android malware detection*, Proceedings of the 10th international conference on Mobile systems, applications, and services, ACM, 2012, pp. 281–294.

[75] Khodor Hamandi, Ali Chehab, Imad H Elhajj, and Ayman Kayssi, *Android sms malware: Vulnerability and mitigation*, Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on, IEEE, 2013, pp. 1004–1009.

[76] Isredza Rahmi A Hamid, Nur Syafiqah Khalid, Nurul Azma Abdullah, Nurul Hidayah Ab Rahman, and Chuah Chai Wen, *Android malware classification using k-means clustering algorithm*, IOP Conference Series: Materials Science and Engineering, vol. 226, IOP Publishing, 2017, p. 012105.

[77] Mark Hillick, *Scareware traversing the world via a web app exploit*, Bethesda (MD): SANS Institute InfoSec Reading Room (2010).

[78] Jingyu Hua and Kouichi Sakurai, *A sms-based mobile botnet using flooding algorithm*, IFIP International Workshop on Information Security Theory and Practices, Springer, 2011, pp. 264–279.

[79] Chun-Ying Huang, Yi-Ting Tsai, and Chung-Han Hsu, *Performance evaluation on permission-based detection for android malware*, Advances in Intelligent Systems and Applications-Volume 2, Springer, 2013, pp. 111–120.

[80] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro, *Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware*, Proceedings of the 14th International Conference on Mining Software Repositories, IEEE Press, 2017, pp. 425–435.

[81] Mikko Hypponen, *Malware goes mobile*, Scientific American **295** (2006), no. 5, 70–77.

[82] IBM, *Financial malware explained*, `http://cdn.americanbanker.com/pdfs/WGW03086USEN.PDF`, 2014.

[83] Laheeb Mohammed Ibrahim and Karam H Thanon, *Analysis and detection of the zeus botnet crimeware*, International Journal of Computer Science and Information Security **13** (2015), no. 9, 121.

[84] Ianir Ideses and Assaf Neuberger, *Adware detection and privacy control in mobile devices*, Electrical & Electronics Engineers in Israel (IEEEI), 2014 IEEE 28th Convention of, IEEE, 2014, pp. 1–5.

[85] Paul Irolla and Eric Filiol, *Glassbox: Dynamic analysis platform for malware android applications on real devices*, arXiv preprint arXiv:1609.04718 (2016).

[86] Shaharudin Ismail and Zahri Hj Yunos, *Worms and trojans go mobile*, 2005.

[87] Jiyoung Woo Aziz Mohaisen Huy KangKim Jae-wook Jang, Hyunjae Kang, *Androdumpsys: Anti-malware system based on the similarity of malware creator and malware centric information*, Computers and Security, vol. 58, 2016, pp. 125–138.

[88] Jang and Huy Kang Kim, *Function-oriented mobile malware analysis as first aid*, Mobile Information Systems, Graduate School of Information Security, Korea University, Seoul 136-713, Republic of Korea, 2016.

[89] Yiming Jing, Ziming Zhao, Gail-Joon Ahn, and Hongxin Hu, *Morpheus: automatically generating heuristics to detect android emulators*, Proceedings of the 30th Annual Computer Security Applications Conference, ACM, 2014, pp. 216–225.

[90] Jin-Hyuk Jung, Ju Young Kim, Hyeong-Chan Lee, and Jeong Hyun Yi, *Repackaging attack on android banking applications and its countermeasures*, Wireless Personal Communications (2013), 1421–1437.

[91] Andi Fitriah A Kadir, Natalia Stakhanova, and Ali A Ghorbani, *An empirical analysis of android banking malware*, Protecting Mobile Networks and Devices: Challenges and Solutions (2016), 209.

[92] Andi Fitriah Abdul Kadir, Natalia Stakhanova, and Ali Akbar Ghorbani, *Android botnets: What urls are telling us*, International Conference on Network and System Security, Springer, 2015, pp. 78–91.

[93] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb, *Maldozer: Automatic framework for android malware detection using deep learning*, Digital Investigation **24** (2018), S48–S59.

[94] Kaspersky, *Interpol & kaspersky lab: 60% of android attacks use financial malware*, (2014), no. October.

[95] Kristen Kennedy, Eric Gustafson, and Hao Chen, *Quantifying the effects of removing permissions from android applications*, Workshop on Mobile Security Technologies (MoST), 2013.

[96] Hahnsang Kim, Joshua Smith, and Kang G Shin, *Detecting energy-greedy anomalies and mobile malware variants*, Proceedings of the 6th international conference on Mobile systems, applications, and services, ACM, 2008, pp. 239–252.

[97] Jinyung Kim, Yongho Yoon, Kwangkeun Yi, Junbum Shin, and SWRD Center, *Scandal: Static analyzer for detecting privacy leaks in android applications*, MoST **12** (2012).

[98] William Klieber, Lori Flynn, Amar Bhosale, Limin Jia, and Lujo Bauer, *Android taint flow analysis for app sets*, Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis, ACM, 2014, pp. 1–6.

[99] Hugo Gonzalez Kenneth Fon Mbah Ali A. Ghorban Lashkari, Andi Fitriah A.Kadir, *Towards a network-based framework for android malware detection and characterization.*, the 15th International Conference on Privacy, Security and Trust, PST, Calgary, Canada, 2017.

[100] M Lindorfer, *Andrubis: A tool for analyzing unknown android applications*, 2012.

[101] Michele Maasberg, Myung Ko, and Nicole L Beebe, *Exploring a systematic approach to malware threat assessment*, System Sciences (HICSS), 2016 49th Hawaii International Conference on, IEEE, 2016, pp. 5517–5526.

[102] Davide Maiorca, Francesco Mercaldo, Giorgio Giacinto, Corrado Aaron Visaggio, and Fabio Martinelli, *R-packdroid: Api package-based characterization and detection of mobile ransomware*, Proceedings of the Symposium on Applied Computing, ACM, 2017, pp. 1718–1723.

[103] M.Léveillé Marc-Etienne, *TorrentLocker Ransomware in a country near you*, (2014), no. December.

[104] Francesco Mercaldo, Vittoria Nardone, Antonella Santone, and Corrado Aaron Visaggio, *Ransomware steals your phone. formal methods rescue it*, International Conference on Formal Techniques for Distributed Objects, Components, and Systems, Springer, 2016, pp. 212–221.

[105] Collin Mulliner and Jean-Pierre Seifert, *Rise of the ibots: Owning a telco network*, Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on, IEEE, 2010, pp. 71–80.

[106] Simone Mutti, Yanick Fratantonio, Antonio Bianchi, Luca Invernizzi, Jacopo Corbetta, Dhilung Kirat, Christopher Kruegel, and Giovanni Vigna, *Baredroid: Large-scale analysis of android apps on real devices*, Proceedings of the 31st Annual Computer Security Applications Conference, ACM, 2015, pp. 71–80.

[107] Alexios Mylonas, Stelios Dritsas, Bill Tsoumas, and Dimitris Gritzalis, *On the feasibility of malware attacks in smartphone platforms*, E-business and telecommunications, Springer, 2011, pp. 217–232.

[108] Mohammad Nauman and Sohail Khan, *Design and implementation of a fine-grained resource usage model for the android platform.*, Int. Arab J. Inf. Technol. **8** (2011), no. 4, 440–448.

[109] Mohammad Nauman, Sohail Khan, and Xinwen Zhang, *Apex: extending android permission model and enforcement with user-defined runtime constraints*, Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ACM, 2010, pp. 328–332.

[110] Anh Nguyen and Lei Pan, *Detecting sms-based control commands in a botnet from infected android devices*, ATIS 2012: Proceedings of the 3rd Applications and Technologies in Information Security Workshop, School of Information Systems, Deakin University, 2012, pp. 23–27.

[111] Machigar Ongtang, Kevin Butler, and Patrick McDaniel, *Porscha: Policy oriented secure content handling in android*, Proceedings of the 26th Annual Computer Security Applications Conference, ACM, 2010, pp. 221–230.

[112] Machigar Ongtang, Stephen McLaughlin, William Enck, and Patrick McDaniel, *Semantically rich application-centric security in android*, Security and Communication Networks **5** (2012), no. 6, 658–673.

[113] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research **12** (2011), 2825–2830.

[114] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., *Scikit-learn: Machine learning in python*, Journal of machine learning research **12** (2011), no. Oct, 2825–2830.

[115] Ms Nigam Paridhi Pravin, *Vetdroid: Analysis using permission for vetting undesirable behaviours in android applications*, International Journal of Innovative and Emerging Research in Engineering **2** (2015), no. 3, 131–136.

[116] P. Cook R. Killam and N. Stakhanova, *Android malware classification through analysis of string literals*, Analytics for Cybersecurity and Online Safety (TACOS)), 2016.

[117] Siegfried Rasthofer, Irfan Asrar, Stephan Huber, and Eric Bodden, *How current android malware seeks to evade automated code analysis*, Information Security Theory and Practice, Springer, 2015, pp. 187–202.

[118] Vaibhav Rastogi, Yan Chen, and William Enck, *Appsplayground: automatic security analysis of smartphone applications*, Proceedings of the third ACM conference on Data and application security and privacy, ACM, 2013, pp. 209–220.

[119] Marco Riccardi, David Oro, Jesus Luna, Marco Cremonini, and Marc Vilanova, *A framework for financial botnet analysis*, eCrime Researchers Summit (eCrime), 2010, IEEE, 2010, pp. 1–7.

[120] Sanae Rosen, Zhiyun Qian, and Z Morely Mao, *Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users*, Proceedings of the third ACM conference on Data and application security and privacy, ACM, 2013, pp. 221–232.

[121] Per Runeson and Martin Höst, *Guidelines for conducting and reporting case study research in software engineering*, Empirical software engineering **14** (2009), no. 2, 131.

[122] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez, *Puma: Permission usage to detect malware in android*, International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions, Springer, 2013, pp. 289–298.

[123] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli, *Madam: Effective and efficient behavior-based android malware detection and prevention*, IEEE Transactions on Dependable and Secure Computing (2016).

[124] Golam Sarwar, Olivier Mehani, Roksana Boreli, and Mohamed Ali Kaafar, *On the effectiveness of dynamic taint analysis for protecting against private information leaks on android-based devices.*, SECRYPT, vol. 96435, 2013.

[125] Ryo Sato, Daiki Chiba, and Shigeki Goto, *Detecting android malware by analyzing manifest files*, Proceedings of the Asia-Pacific Advanced Network **36** (2013), no. 23-31, 17.

[126] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Jan Clausen, Kamer A Yuksel, Osman Kiraz, Ahmet Camtepe, and Sahin Albayrak, *Enhancing security of linux-based android devices*, in Proceedings of 15th International Linux Kongress. Lehmann, 2008.

[127] Daniel Schreckling, Joachim Posegga, and Daniel Hausknecht, *Constroid: data-centric access control for android*, Proceedings of the 27th Annual ACM Symposium on Applied Computing, ACM, 2012, pp. 1478–1485.

[128] Christian Seifert, Jack Stokes, Long Lu, David Heckerman, Christina Colcernian, Sasi Parthasarathy, and Navaneethan Santhanam, *Scareware detection*, September 8 2015, US Patent 9,130,988.

[129] Christian Seifert, Jack W Stokes, Christina Colcernian, John C Platt, and Long Lu, *Robust scareware image detection*, Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, 2013, pp. 2920–2924.

[130] Asaf Shabtai, Yuval Fledel, and Yuval Elovici, *Automated static code analysis for classifying android applications using machine learning*, Computational Intelligence and Security (CIS), 2010 International Conference on, IEEE, 2010, pp. 329–333.

[131] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss, *"andromaly": a behavioral malware detection framework for android devices*, Journal of Intelligent Information Systems **38** (2012), no. 1, 161–190.

[132] Raja Khurram Shahzad and Niklas Lavesson, *Detecting scareware by mining variable length instruction sequences*, Information Security South Africa (ISSA), 2011, IEEE, 2011, pp. 1–8.

[133] RajaKhurram Shahzad and Niklas Lavesson, *Veto-based malware detection*, Availability, Reliability and Security (ARES), 2012 Seventh International Conference on, IEEE, 2012, pp. 47–54.

[134] Yuru Shao, Qi Alfred Chen, Zhuoqing Morley Mao, Jason Ott, and Zhiyun Qian, *Kratos: Discovering inconsistent security policy enforcement in the android framework.*, NDSS, 2016.

[135] Wook Shin, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka, *Towards formal analysis of the permission-based security model for android*, Wireless

and Mobile Communications, 2009. ICWMC'09. Fifth International Conference on, IEEE, 2009, pp. 87–92.

[136] Sanggeun Song, Bongjoon Kim, and Sangjun Lee, *The effective ransomware prevention technique using process monitoring on android platform*, Mobile Information Systems **2016** (2016).

[137] ST Tajalizadehkhoob, Hadi Asghari, Carlos Gañán, and MJG Van Eeten, *Why them? extracting intelligence about target selection from zeus financial malware*, Proceedings of the 13th Annual Workshop on the Economics of Information Security, WEIS 2014, State College (USA), June 23-24, 2014, WEIS, 2014.

[138] Kimberly Tam, Salahuddin J Khan, Aristide Fattori, and Lorenzo Cavallaro, *Copperdroid: Automatic reconstruction of android malware behaviors.*, NDSS, 2015.

[139] Gianluca Stringhini Tangil, *Eight years of rider measurement in the android malware ecosystem: Evolution and lessons learned.*, 2018.

[140] Botnet Research Team et al., *Sanddroid: An apk analysis sandbox. xi'an jiaotong university*, 2014.

[141] Chia-Wei Tien, Tse-Yung Huang, Ting-Chun Huang, Wei-Ho Chung, and Sy-Yen Kuo, *Mas: Mobile-apps assessment and analysis system*, Dependable Systems and Networks Workshop (DSN-W), 2017 47th Annual IEEE/IFIP International Conference on, IEEE, 2017, pp. 145–148.

[142] Virus Total, *Contagio mobile malware mini dump*, Online: http://contagiominidump.blogspot.ca/ (2016).

[143] Bartlomiej Uscilowski, *Mobile adware and malware analysis*, Symantec Corp **1** (2013).

[144] Kalgutkar Vaibahvi, *Android authorship attribution through string analysis*, 2017.

[145] Timothy Vidas and Nicolas Christin, *Evading android runtime analysis via sandbox detection*, Proceedings of the 9th ACM symposium on Information, computer and communications security, ACM, 2014, pp. 447–458.

[146] Ickin Vural and Hein Venter, *Mobile botnet detection using network forensics*, Future Internet-FIS 2010, Springer, 2010, pp. 57–67.

[147] Robert Wahbe, Steven Lucco, Thomas E Anderson, and Susan L Graham, *Efficient software-based fault isolation*, ACM SIGOPS Operating Systems Review, vol. 27, ACM, 1994, pp. 203–216.

[148] Wei Wang, Ilona Murynets, Jeffrey Bickford, Christopher Van Wart, and Gang Xu, *What you see predicts what you get—lightweight agent-based malware detection*, Security and Communication Networks **6** (2013), no. 1, 33–48.

[149] Sankardas Roy Xinming Ou Wei, Yuping Li and Wu Zhou, *Deep ground truth analysis of current android malware.*, LNCS, volume 10327, 14th Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2017), Springer, Cham, 2017.

[150] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos, *Profiledroid: multi-layer profiling of android applications*, Proceedings of the 18th annual international conference on Mobile computing and networking, ACM, 2012, pp. 137–148.

[151] Erik Ramsgaard Wognsen, Henrik Søndberg Karlsen, Mads Chr Olesen, and René Rydhof Hansen, *Formalisation and analysis of dalvik bytecode*, Science of Computer Programming **92** (2014), 25–55.

[152] Lei Xue, Yajin Zhou, Ting Chen, Xiapu Luo, and Guofei Gu, *Malton: Towards on-device non-invasive mobile malware analysis for art*, In 26th USENIX Security Symposium (USENIX Security 17). ACM, 2017.

[153] Lok-Kwong Yan and Heng Yin, *Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis.*, USENIX security symposium, 2012, pp. 569–584.

[154] Tianda Yang, Yu Yang, Kai Qian, Dan Chia-Tien Lo, Ying Qian, and Lixin Tao, *Automated detection and analysis for android ransomware*, High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on, IEEE, 2015, pp. 1338–1343.

[155] Ilsun You and Kangbin Yim, *Malware obfuscation techniques: A brief survey*, Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on, IEEE, 2010, pp. 297–300.

[156] Yuanyuan Zeng, Kang G Shin, and Xin Hu, *Design of sms commanded-and-controlled and p2p-structured mobile botnets*, Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, ACM, 2012, pp. 137–148.

[157] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang, *Vetting undesirable behaviors in android apps with permission use analysis*, Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013, pp. 611–622.

[158] Min Zhao, Fangbin Ge, Tao Zhang, and Zhijian Yuan, *Antimaldroid: An efficient svm-based malware detection framework for android*, International Conference on Information Computing and Applications, Springer, 2011, pp. 158–166.

[159] Yury Zhauniarovich, Olga Gadyatskaya, Bruno Crispo, Francesco La Spina, and Ermanno Moser, *Fsquadra: fast detection of repackaged applications*, IFIP Annual Conference on Data and Applications Security and Privacy, Springer, 2014, pp. 130–145.

[160] Cong Zheng, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, and Wei Zou, *Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications*, Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices, ACM, 2012, pp. 93–104.

[161] Min Zheng, Mingshen Sun, and John CS Lui, *Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware*, Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on, IEEE, 2013, pp. 163–171.

[162] Wu Zhou, Xinwen Zhang, and Xuxian Jiang, *Appink: watermarking android apps for repackaging deterrence*, Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, ACM, 2013, pp. 1–12.

[163] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning, *Detecting repackaged smartphone applications in third-party android marketplaces*, Proceedings of the second ACM conference on Data and Application Security and Privacy, ACM, 2012, pp. 317–326.

[164] Yajin Zhou and Xuxian Jiang, *Dissecting android malware: Characterization and evolution*, Security and Privacy (SP), 2012 IEEE Symposium on, IEEE, 2012, pp. 95–109.

[165] 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.*, NDSS, vol. 25, 2012, pp. 50–52.

[166] Aqil Zulkifli, Isredza Rahmi A Hamid, Wahidah Md Shah, and Zubaile Abdullah, *Android malware detection based on network traffic using decision tree algorithm*, International Conference on Soft Computing and Data Mining, Springer, 2018, pp. 485–494.

Appendix A

Results of Droidkin

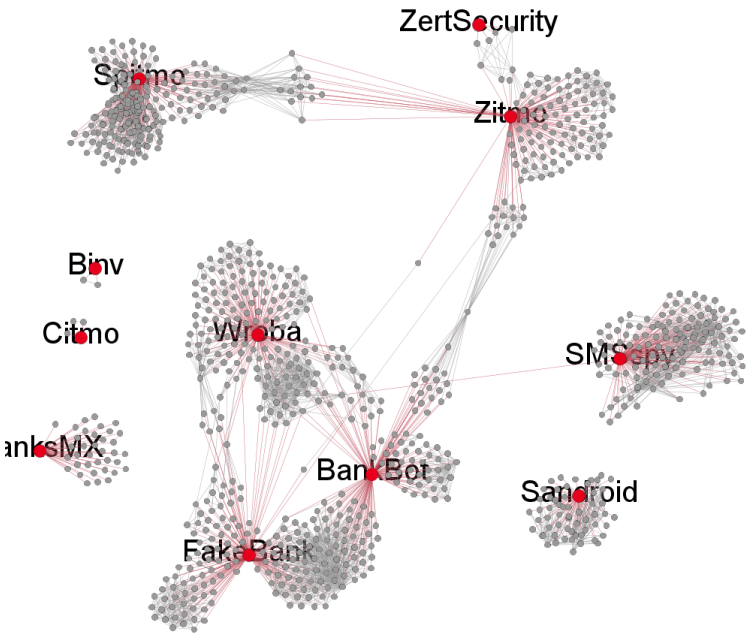

Figure A.1: Droidkin result for Android banking dataset. Red circles symbolize a family, black circles represent apps, and the lines indicate the relationships among the apps. If two red circles are connected, it represents the existence of a relationship

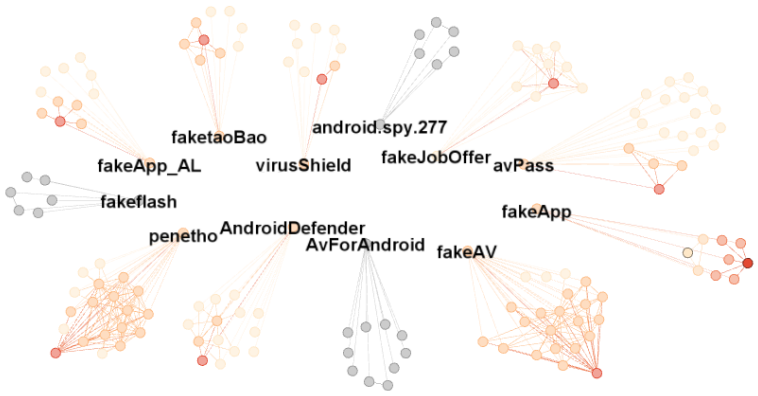

Figure A.2: Droidkin result for Android scareware dataset. Red circles symbolize a family, black circles represent apps, and the lines indicate the relationships among the apps. If two red circles are connected, it represents the existence of a relationship

Appendix B

List of Android Malware

BankBot. This malware family is a Trojan-banker type that steals the user's confidential banking information. This Trojan can imitate legitimate online banking applications to get access to clients' confidential data; by sending a special SMS message to a phone number (associated to the user's bank account). Another variant of this family can scan the system for the presence of banking applications, download its fake version as an update package, and uninstall original programs. Moreover, the malware can intercept incoming SMS messages, steal phone book information, gather data regarding the infected mobile device, and even uninstall some anti-viruses [12].

Binv. This malware is a classical banking-Trojan that targets the Brazilian users of Android devices by masquerading its malicious applications as legitimate banking applications. The Brazilian mobile banking users particularly exposed to the attack due to the lack of authentication system (i.e. two-factor authentication) for banking applications. The malware is very simple; it was designed to steal login credentials in a classic phishing scheme. This malware was created using a free development tool known as App Inventor, which does not require any particular skill to create a mobile application. The malware was originally published on the Google Play store with the name Governo Federal (Federal Government) [17]

Citmo. The cybercriminals behind Carberp began its operations in 2009 but did not actually migrate to the mobile platform until 2012, when a Carberp-in-the-mobile, or Citmo is found on Google Play masqueraded as mobile applications. Citmo was primarily targeting Russian banks and financial institutions (Sberbank and Alfa bank). However in December 2012, Citmo was also targeting several banks in the US and Canada. Citmo has joined the ranks of Zeus-in-the-mobile (Zitmo) and SpyEye-in-the-mobile (Spitmo) in targeting SMS messages sent by banks to authenticate online banking transactions. The Citmo Android Trojan works in almost the same way as ZitMo. It can hide particular SMS messages, or resend them to the attacker's command server or to other phone numbers[3]. Even though the cybercriminals associated with Carberp have been arrested by Russian and Ukranian authorities, the Carberp malware is not over as Carberp's source code was leaked in 2013. Similar to Zeus, anyone can modify Carberp. Thus, the new variants can

appear in Europe, United States, or Latin America. The malware was also published on the Google Play store[5].

FakeBank. This malware name is self-explanotary; The fake banking application which targets Iranian users. FakeBank is a Trojan horse for Android devices that opens a back door and steals information from the compromised device. Once installed on a victim's phone, the application monitors SMS activity for incoming verification messages from the Iranian bank it seeks to imitate. This malware copies the information into another message and sends it out via SMS, to the attackers. Additionally, it is able to infect a connected Windows PC and tricks the user to exchange legit banking apps against malicious ones. It was also published on the Google Play store [13].

Sandroid. This malware disguises itself as a bank dynamic token generator which targets middle east banks, including Riyad Bank, SAAB (formerly the Saudi British Bank), AlAhliOnline (National Commercial Bank), Al Rajhi Bank, and Arab National Bank. It is running another background service that monitors user's incoming SMS. Every incoming SMS is silently sent to a certain number through SMS. It also actively silently sends SMS commands from a remote website. The remote malicious user may then initiate unauthorized transaction without user's consent and may incur financial loss [3].

SMSspy. This malware is a classical banking-Trojan that targets the Spanish users of Android devices by masquerading its malicious applications as the legitimate banking applications. Once installed on a victim's phone, the application monitors SMS activity for incoming verification messages and is able to intercept and modify banking authentication codes (mTAN messages)[13].

Spitmo. This malware is also known as SpyEye-in-the-mobile and was discovered in 2011 after the SpyEye source code was leaked. Spitmo is a banking trojan that steals information from the infected smartphone. The Trojan also monitors and intercepts SMS messages from banks (mTAN messages) and uploads them to a remote server. In 2009, SpyEye surpassed Zeus and evolved into pricey, user-friendly software program. It was sold, updated and copyrighted like a legit business application. However, in January 2014, the man who created SpyEye plead guilty and he was sentenced by a US district judge.[13].

Wroba. The Korean malware Wroba spreads via alternative application stores. Once it infects a device, Wroba behaves very aggressively. It searches for mobile banking applications, removes them and uploads counterfeit versions. This malware hides its main malicious activity within a package that is encrypted and hidden within itself. Wroba malware is used to capture login credentials for bank accounts and banking information and other data to monetize the attack. The inner malicious package is present in the original package as an asset file and is decrypted using DES before it can be loaded and the malicious functions called [4].

ZertSecurity. This malware family is a Trojan banker type which steals confidential banking authentication codes (mTAN messages). Once installed on a victim's phone, the application tricks a compromised user to insert his banking account details. The Trojan also monitors and intercepts SMS messages from banks (mTAN messages) and uploads them to a remote server.[13].

Zitmo. This malware is also called ZeuS-in-the-Mobile and was discovered in the end of September 2010. Zitmo targeted various mobile platforms such as Symbian, Windows Mobile, Blackberry, and Android. Its functionality consists of; the ability to upload all incoming SMS messages (with mTAN also) to a remote web server, the ability to forward SMS messages from a particular number, as well as the ability to change the command and control (C&C). The malicious application passes itself off as a security tool from the *Trusteer* company. If a user installs the malicious application then the *Trusteer Rapport* icon will appear in the main menu[5].

Appendix C

List of Network Traffic features

Table C.1: The list of network-flow features extracted by CICflowmeter[70]

Feature name	Description
duration	Duration of the flow
total_fpackets	Total packets in the forward direction
total_bpackets	Total packets in the backward direction
total_fpktl	Total size of packet in forward direction
total_bpktl	Total size of packet in backward direction
min_fpktl	Minimum size of packet in forward direction
min_bpktl	Minimum size of packet in backward direction
max_fpktl	Maximum size of packet in forward direction
max_bpktl	Maximum size of packet in backward direction
mean_fpktl	Mean size of packet in forward direction
mean_bpktl	Mean size of packet in backward direction
std_fpktl	Standard deviation size of packet in forward direction
std_bpktl	Standard deviation size of packet in backward direction
total_fiat	Total time between two packets sent in the forward direction
total_biat	Total time between two packets sent in the backward direction
min_fiat	Minimum time between two packets sent in the forward direction
min_biat	Minimum time between two packets sent in the backward direction
max_fiat	Maximum time between two packets sent in the forward direction
max_biat	Maximum time between two packets sent in the backward direction
mean_fiat	Mean time between two packets sent in the forward direction
mean_biat	Mean time between two packets sent in the backward direction
std_fiat	Standard deviation time between two packets sent in the forward direction
std_biat	Standard deviation time between two packets sent in the backward direction
total_fhlen	Total bytes used for headers in the forward direction
total_bhlen	Total bytes used for headers in the forward direction
fPktsPerSecond	Number of forward packets per second
bPktsPerSecond	Number of backward packets per second
flowPktsPerSecond	Number of flow packets per second
flowBytesPerSecond	Number of flow bytes per second
min_flowpktl	Minimum length of a flow
max_flowpktl	Maximum length of a flow
mean_flowpktl	Mean length of a flow
std_flowpktl	Standard deviation length of a flow
min_flowiat	Minimum inter-arrival time of packet
max_flowiat	Maximum inter-arrival time of packet
mean_flowiat	Mean inter-arrival time of packet
std_flowiat	Standard deviation inter-arrival time of packet
downUpRatio	Download and upload ratio
avgPacketSize	Average size of packet
min_active	Minimum time a flow was active before becoming idle
mean_active	Mean time a flow was active before becoming idle
max_active	Maximum time a flow was active before becoming idle
std_active	Standard deviation time a flow was active before becoming idle
min_idle	Minimum time a flow was idle before becoming active
mean_idle	Mean time a flow was idle before becoming active
max_idle	Maximum time a flow was idle before becoming active
std_idle	Standard deviation time a flow was idle before becoming active

Appendix D

List of Scripts

Listing D.1: Script for checking .apk file

```
#!/bin/bash
for f in ./*; do
        type=$(file $f)
        if [[ $f == *".apk" ]]; then
                signed=$(jarsigner -verify $f)
                        echo $f "is already an apk file"
        elif [[ $type == *"Zip archive data"* ]]; then
                signed=$(jarsigner -verify $f)
                        mv $f $f.apk
                        echo $f "is now an apk file"
        elif [[ $type == *"dex"* ]]; then
                echo $f "is a dex file"
        fi
done
```

Listing D.2: Command for decoding and encoding app

```
 \$ apktool d ransomware.apk
I: Using Apktool 2.3.1 on ransomware.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: 1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
\$ apktool b ransomware
I: Using Apktool 2.3.1 on ransomware
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
```

```
import android.support.test.InstrumentationRegistry;
import android.support.test.filters.SdkSuppress;
import android.support.test.runner.AndroidJUnit4;
import android.support.test.uiautomator.UiDevice;
import android.support.test.uiautomator.UiObject;
import android.support.test.uiautomator.UiSelector;
import org.junit.Test;
import org.junit.runner.RunWith;

@RunWith(AndroidJUnit4.class)
@SdkSuppress(minSdkVersion = 18)
public class ChangeTextBehaviorTest {
    private static final String BASIC_SAMPLE_PACKAGE = "com.example.mkhoshpa.ui";
    private static final int LAUNCH_TIMEOUT = 5000;
    private static final String STRING_TO_BE_TYPED = "UiAutomator";
    private UiDevice mDevice;

    @Test
    public void startMainActivityFromHomeScreen() throws Exception {
        String[] texts = new String[]{"Adam where are you now? I want to change my bank's pin, but they ask me to bring my passport for validation", "
t to deposit your check.", "CIMB: IBFT Transfer $300 to Rania/Maybank on 02 Jun 2017 10:02:02. Call 60362047788 if you DID NOT perform this transactio
xpires by Jun 07 15:20:01", "The mobile TAN 736592 for transaction 025787154. Expires by Sat Jun 03 15:39:14", "Hello Ryan, I have used the same passw
 instagram", "Hey Trump, did you pay for the tax yet? Did you have to pay the 13% HST for them?"};
        this.mDevice = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());
        this.mDevice.pressHome();
        this.mDevice.pressDPadCenter();
        this.mDevice.findObject(new UiSelector().textContains("Pulse")).clickAndWaitForNewWindow();
        this.mDevice.findObject(new UiSelector().resourceId("xyz.klinker.messenger:id/fab")).clickAndWaitForNewWindow();
        UiObject to = this.mDevice.findObject(new UiSelector().resourceId("xyz.klinker.messenger:id/contact_entry"));
        to.click();
        to.setText("5065555532");
        this.mDevice.findObject(new UiSelector().resourceId("xyz.klinker.messenger:id/fab")).click();
        this.mDevice.findObject(new UiSelector().resourceId("xyz.klinker.messenger:id/message_entry")).setText(texts[(int) (Math.random() * 7.0d)]);
        this.mDevice.findObject(new UiSelector().resourceId("xyz.klinker.messenger:id/send")).click();
        this.mDevice.pressBack();
        this.mDevice.pressBack();
        this.mDevice.pressHome();
    }
```

Figure D.1: UI for sending message

Listing D.3: Script for sending SMS

```
#!/bin/bash
adb shell am start -a android.intent.
action.SENDTO -d sms:\<phone_number\>
--es sms_body "\<message_body\>"
--ez exit_on_sent true
adb shell input keyevent 22
adb shell input keyevent 66
```

Listing D.4: Script for sending calls (without SIM cards)

```
 adb shell am start -a android.intent.
 action.CALL %-d tel:<phone_number>.
```

Listing D.5: Script for sending calls (with SIM cards)

```
#!/bin/bash
adb shell am start -a android.intent.
 action.CALL -d tel:
```

Listing D.6: Script for browsing Internet

```
#!/bin/bash
adb shell am start -a android.intent.
action.VIEW -d url
```

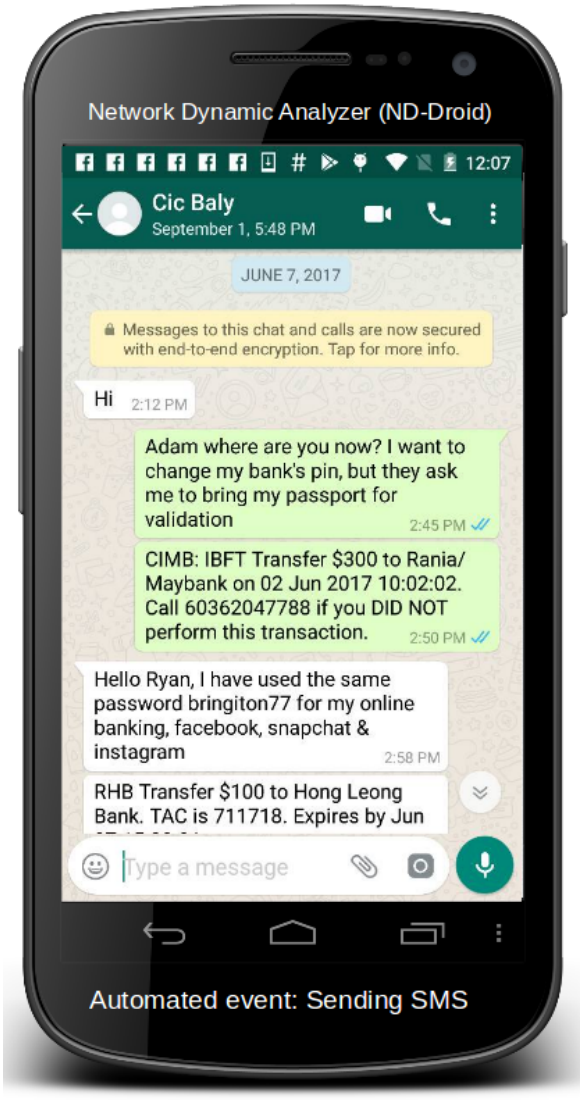

Figure D.2: ND-Droid: automated event example (sending messages)

Vita

Andi Fitriah Abdul Kadir

Education

Master of Computer Science (Network Security),
International Islamic University Malaysia (IIUM), Malaysia 2011-2013,
Thesis: Analyzing the Dynamics Behavior of Fast-Flux Domain Name System (DNS) through Visualization.

Bachelor of Computer Science (Major in Computer Networks),
International Islamic University Malaysia (IIUM), Malaysia 2005-2009.
Thesis: ROBODA SP2: FPGA-Based robot for object detection applications.

Publications

Journal:

A. Abdul Kadir, N. Stakhanova, and A. Ghorbani. Understanding Android Financial Malware Attacks: Taxonomy, Characteristics, and Challenges. Journal of Cyber Security and Mobility, 7(3):1-52, July 2018.

Referred Book Chapter:

A. Abdul Kadir, N. Stakhanova, and A. Ghorbani. An empirical analysis of Android banking malware. In W. Meng, X. Luo, J. Zhou, and S. Furnell, editors, Protecting Mobile Networks and Devices: Challenges and Solutions. CRC Press - Taylor Francis, 2016.

Conference Papers:

AH Lashkari, A. Abdul Kadir, L. Taheri, and A. Ghorbani, Towards Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification, In the proceedings of the 52nd IEEE International Carnahan Conference on Security Technology (ICCST), Montreal, Quebec, Canada, 2018.

AH Lashkari, A. Abdul Kadir, H Gonzalez, KF Mbah, and A. Ghorbani. Towards a Network-Based Framework for Android Malware Detection and Characterization, Proceeding of the 15th international conference on privacy, security and trust, 2017.

A. Abdul Kadir, N. Stakhanova, and A. Ghorbani. Android botnets: What urls are telling us. In M. Qiu, S. Xu, M. Yung, and H. Zhang, editors, International Conference on Network and System Security (NSS) , volume 9408 of Lecture Notes in Computer Science , pages 78-91, Springer International Publishing, 2016. (**Best Paper Honorable Mention**)

H. Gonzalez, A. Abdul Kadir, N. Stakhanova, A. J. Alzahrani, and A. A. Ghorbani. Exploring reverse engineering symptoms in Android apps. In Proceedings of the Eighth European Workshop on System Security, EuroSec '15, pages 7:1-7:7, New York, NY, USA, 2015. ACM.

A. Abdul Kadir, RAR Othman, NA Aziz. Behavioral analysis and visualization of fast-flux DNS. In Intelligence and Security Informatics Conference (EISIC), 2012 European (pp. 250-253). IEEE.

Posters:

A. Abdul Kadir, N. Stakhanova, and A. Ghorbani. Android financial malware detection through network flow analysis. Poster session presented at the ACM Canadian Celebration of Women in Computing (CAN-CWiC), Halifax Convention Centre, 2018.

AH Lashkari, A. Abdul Kadir, L. Taheri, and A. Ghorbani, Towards Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. Poster session presented at the Annual Research Exposition of Faculty of Computer Science, UNB, Wu Conference Centre, Auditorium, 2018.

A. Abdul Kadir, N. Stakhanova, and A. Ghorbani. A framework for detecting Android financial malware. Poster session presented at the Annual Research Exposition of Faculty of Computer Science, UNB, Wu Conference Centre, Auditorium, 2017.

H. Gonzalez, A. Abdul Kadir, N. Stakhanova, A. J. Alzahrani, and A. A. Ghorbani. Exploring reverse engineering symptoms in Android apps. Poster session presented at the Annual Research Exposition of Faculty of Computer Science, UNB, Wu Conference Centre, Auditorium, 2016.

A. Abdul Kadir, N. Stakhanova, and A. Ghorbani. An empirical analysis of Android banking malware. Poster session presented at the Annual Research Exposition of Faculty of Computer Science, UNB, Wu Conference Centre, Auditorium, 2016.

A. Abdul Kadir, N. Stakhanova, and A. Ghorbani. Android botnets: What urls are telling us. Poster session presented at the Annual Research Exposition of Faculty of Computer Science, UNB, Wu Conference Centre, Auditorium, 2015.

A. Abdul Kadir, RAR Othman, NA Aziz. Behavioral analysis and visualization of fast-flux DNS. Poster session presented at Invention and Innovation Exhibition (IRIIE), IIUM, 2012. (**Gold Medal Award**)

A. Abdul Kadir, RAR Othman, NA Aziz. Analzing The Dynamics Behaviour of Fast-Flux DNS Through Visualization. Poster session presented at Postgraduate Research Colloquium (PGSC), IIUM, 2012. (**Best Poster Award**)

Workshop:

Workshop, "Digital Forensics Workshop", Information Security Centre of Excellence (ISCX), UNB, July 7, 2016.

Workshop, "A Brief Introduction to Honeypots", CS Square workshop at Faculty of Computer Science, UNB, October 28, 2015.