Android Malware Detection System using Genetic Programming

Norliza Binti Abdullah

Doctor of Philosophy

University of York

Computer Science

March 2019

This thesis is dedicated to

my family especially for my late father and mother

❤ *both of you till Jannah Insya Allah*

Abstract

Nowadays, smartphones and other mobile devices are playing a significant role in the way people engage in entertainment, communicate, network, work, and bank and shop online. As the number of mobile phones sold has increased dramatically worldwide, so have the security risks faced by the users, to a degree most do not realise. One of the risks is the threat from mobile malware. In this research, we investigate how supervised learning with evolutionary computation can be used to synthesise a system to detect Android mobile phone attacks. The attacks include malware, ransomware and mobile botnets. The datasets used in this research are publicly downloadable, available for use with appropriate acknowledgement. The primary source is Drebin. We also used ransomware and mobile botnet datasets from other Android mobile phone researchers. The research in this thesis uses Genetic Programming (GP) to evolve programs to distinguish malicious and non-malicious applications in Android mobile datasets. It also demonstrates the use of GP and Multi-Objective Evolutionary Algorithms (MOEAs) together to explore functional (detection rate) and non-functional (execution time and power consumption) trade-offs. Our results show that malicious and non-malicious applications can be distinguished effectively using only the permissions held by applications recorded in the application's Android Package (APK). Such a minimalist source of features can serve as the basis for highly efficient Android malware detection. Non-functional tradeoffs are also highlight.

Contents

List of Figures

List of Tables

Abbreviations

SMS	Short Message Services
MMS	Multimedia Message Services
PDA	Personal Digital Assistant
LTE	Long Term Evolution
GPS	Global Positioning System
Wi-Fi	Wireless Fidelity
GSM	Global Standards for Mobile
DDoS	Distributed Denial of Service
IDS	Intrusion Detection Systems
IPS	Intrusion Prevention Systems
AI	Artificial Intelligence
GP	Genetic Programming
SVM	Support Vector Machine
AODV	Ad hoc On-Demand Distance Vector
ANN	Artificial Neural Network
GA	Genetic Algorithm
EC	Evolutionary Computation
EA	Evolutionary Algorithm
MOEC	Multi-Objective Evolutionary Computation
MOEA	Multi-Objective Evolutionary Algorithm
OS	Operating System
PC	Personal Computer
DoS	Denial-Of-Service
URL	Uniform Resource Locator
IP	Internet Protocol
MANET	Mobile Ad hoc Network
APK	Android Application Package

Acknowledgement

Alhamdulillah.
Praise to Allah, for His blessing. I managed to complete my thesis through blood sweat and tears for the past four years of my life dedicated to this research. Finally, this baby was born.

Firstly, I would like to thank my former supervisor, Professor John A. Clark, for all his support, assistance, and advice throughout this lonely journey. I would also like to thank Dr Daniel Kudenko for his help and valuable feedback while he was my internal examiner before stepping up to be my supervisor. I would like to show my gratitude and appreciation towards Professor Susan Stepney as my latest supervisor, your support I needed the most.

I would also like to acknowledge my internal and external examiners, Dr Vasileios Vasilakis and Professor Siraj Shaikh, respectively, for their valuable and insightful comments, which inspired me to broaden my research from various perspectives.

Thanks to all RCH 231 residents during ups and downs during four years of struggling to reach the end of our journey. To all my Malaysian friends, thanks a lot for the moral support through thick and thin. I would like to thank the Ministry of Education Malaysia who has supported this study and, I would like to show my gratefulness to the Department of Polytechnic & Community College Education, Malaysia for allowing me to take the study leave.

For both my late parent this is for both of you, even you are not here I know you to be happy to see this finally completed. To my siblings, nieces and nephews, I am so grateful to be born in our family, and I love all of you. Also, to Q, I know there must be a reason why ALLAH put you in my path and me into your path at the end of my journey, so I am not so lonely.

Declaration

I declare that this thesis is a presentation of original work, and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Contributions from this thesis have been present and published in the following conference and proceedings:

- Norliza Abdullah, John A Clark and Daniel Kudenko. Mobile Network Security. Poster presentation in Malaysian Students Conference and Research Showcase United Kingdom (MySECON 2017), May 2017.
  (Based on research described in Chapter 3 of this thesis.)

- Norliza Abdullah and Nursakinah Md. Salleh. Permission-based Android Malware Detection System Using Genetic Programming. Presented in *5th Technology and Innovation International Conference (TECHON 2019)*. The paper published in *Diges PMU (Technology and Innovation International Conference) TECHON 2019*, pages 240 – 248, October 2019. [1]
  (Based on research described in Chapter 4 of this thesis. The online version can be accessed at https://www.pmu.edu.my/techon2019/)

CHAPTER 1

Introduction

This chapter discusses the problems faced by mobile phones users. These problems motivate the research in this thesis, and it gives the proposed research hypothesis. Lastly, a brief overview of all remaining chapters is outlined.

1.1 Motivation

1.1.1 Mobile Phones Technologies

The speed and reach of communications have seen many significant advances over several millennia. Early examples would be the use of drums, smoke signals in 200BC and carrier pigeons in the 12th century. The telegraph was the first use of electric signals for communication and was followed by landlines, the dial-up Internet, SMS, broadband Internet, MMS, Skype, Facebook, Smartphone, and more. In the 21st century, high bandwidth communication is demanded by everyone, and the emergence of high-speed broadband networks has led to a myriad of applications and communication-intensive services.

Communication has become more accessible across the globe. Audiences have information at their fingertips. Today the communication devices used are not limited to telephone and fax; modern communication uses a range of devices, e.g. laptops, PDAs, mobile phones, and smartphones. Shifts within the nature of interactions are reflected in the continuously quoted 'generation' concept. Mobile services have grown from the first generation (1G) for voice traffic to LTE and the fourth generation (4G) [2]. Now, 5G [3] becomes trending for mobile phones developer to win the heart of the users, even not all mobile service provider provides the service. In [4] Fowler is claimed the 5G line is not stable yet and not suggests users in the US upgrade their mobile phones and line to 5G.

The 6G technology still under development, and as mentioned in [3], it will bloom in 2030. The evolution of mobile technology and services are presented in Figure 1 below.

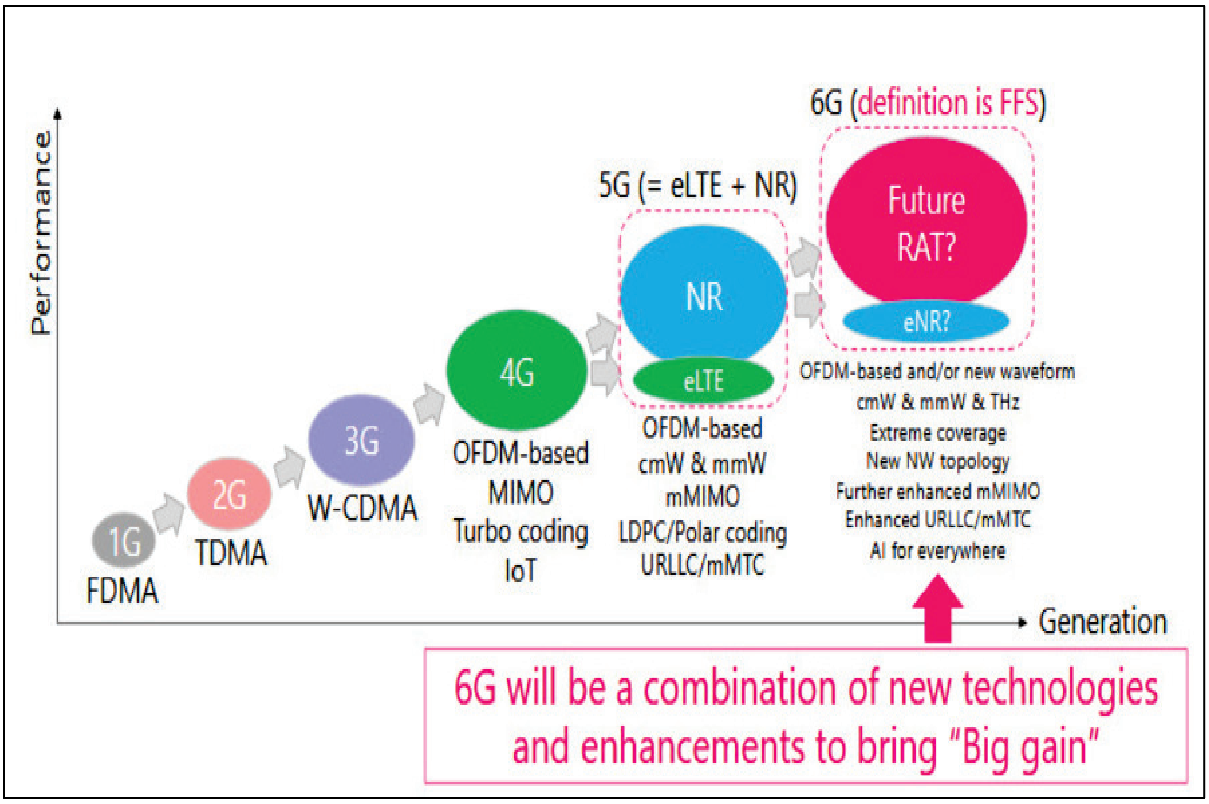


Figure 1.1 Evolution of Mobile Technology and Services [3]

As mobile phones have evolved to embrace connectivity, users have increasingly become the target of cybercriminals. The growth of online service access by mobile phone users has further increased their vulnerability.

1.1.2 Risks Faced by the Mobile Phone User

There are now many smartphone distributors: iPhone, Samsung, Blackberry, Sony, HTC, Nokia, LG, and many more. The global smartphone user base exceeded three billion in 2018 and increased to 3.2 billion in 2019 and predicted to grow to 3.5 billion and 3.8 billion at the end of 2020 and 2021 [5], [6]. In India, in 2018, the average smartphone user used 1GB data per day and spent at least 90 minutes online [7]. As the emergence of Covid-19 in early 2020, all people worldwide ordered to stay inside (lockdown policy) and on average UK phone users tend to devote 2 hours and 34 minutes online on their smartphones each day [8]. Smartphone users have the potential to be attacked because they use their mobile phone most of the time to carry out online activities such as checking emails, social media, working online, attend class online, etc.

Users often do not realise that while using the technologies for communication or surfing the Internet, they are exposing themselves to cyber and related crimes. The crimes included such as fraudulent credit card use, identity theft, hacking, financial theft, spam, phishing frauds, malware (spyware), network spoofing, and private data leakage [9]–[12]. Often users might not realise that they have been attacked since attackers may cover up their crime by masking their true identity from the victims. Attackers who conduct their criminal or malicious activities on networks, computers, and mobile phones often employ such techniques.

Cellular phone users who use their phone to access the Internet and its particular services face several risks. One of the mobile phone's benefits is that it can hold a great deal of private and personal data and so mobile phones become objects that offer significant prospects for criminals' intent on manipulating them. Mobile phones may be subject to a variety of risks: limited battery life, private data leakage, exposure to theft or other physical loss, unsecured Wi-Fi, improper session handling, vulnerability to malware, and camera-based attacks [9]–[11], [13].

Vulnerability to mobile malware (Trojans, viruses, and spyware) is also a threat to mobile phone users [14], [15]. Malware is software installed on a user's mobile phone that can perform mischievous actions. Spyware (one type of malware) is software created to collect private data without the knowledge or approval of users. Malware attacks are usually not detectable by users. A virus can be distributed via Bluetooth, and in 2004 the first widespread Bluetooth worm sample was witnessed, called Cabir [14], [15]. Malware with full control of a smartphone can use it as an eavesdropping device by turning the smartphone camera or microphone on [9]. The attacker can listen to all conversations made, record a video, or take a picture of smartphone victims. A Trojan horse can carry malicious code in an attractive package in a smartphone when users are installing apps [10]. Such a Trojan can detect a user's location by activating the phone's GPS functionality.

The camera-based attack is one example of mobile phone camera security malware [11]. Attackers could slyly take pictures and record videos by using the phone camera without the smartphone user realising it. Attackers can use spy cameras in malicious apps, with the phone camera launched automatically without the knowledge of the device owner. Captured photos and videos can be sent out to the remote intruders via Wi-Fi.

Network threats occur when the mobile phone is connected to public Wi-Fi or Bluetooth. When the mobile phone user connects to public Wi-Fi, malware may be automatically installed on the mobile device. The user also faces Wi-Fi sniffing while they are connected to the public Wi-Fi, and there is a possibility they will be subject to attack by viruses spread by other users in the network. Users might not be aware that while connected to others' Bluetooth, they might be in danger from Bluetooth attacks. The Bluetooth attacks such as Bluejacking (Bluetooth spam), Bluebugging (attackers remotely access a user's phone and use its features), and BlueSnarfing (unauthorised access to data usually happens over a link between the system pairing between the intruder and the devices).

Communication-based threats usually occur on the SMS, MMS, and GSM networks. DDoS attacks can be spread by SMS messages sent from the Internet. The attacks can cause delays when the text messages overload the network. Viruses can attack MMS by using an attachment infected with a virus, and if the MMS is opened, the phone will be infected. The infected phone can cause others' phones to be infected with a virus using the phone contact address book. The eavesdropping attack is one example of a GSM network attack [9]. The Sun [16] reported Google has admitted to its workers listening to users private conversation using the Google Assistant, which is an example of eavesdropping in an Android device.

Researchers have proposed a variety of countermeasures, e.g., firewalls, anti-virus software, anti-spyware software, IDSs, and IPSs. The prime focus of this research is to be combating malware by developing malware detection and classification algorithms.

1.1.3 Brief Overview of Intrusion Detection Systems (IDSs)

An IDS was first introduced by James Anderson in his technical report "Computer Security Threats Monitoring and Surveillance" more than three decades ago [17]. Researchers have given several definitions of IDS in their papers. An IDS is a tool, a software program that observes and protects a network and information system from malicious events such as attacks, misuse and compromise or policy abuses. An IDS has expectations about 'normal' or non-malicious activity. When it detects something 'odd' about the behaviour of a system, it can flag this as suspicious to the system administrator [18]–[21]. The ultimate aim of IDS is to catch perpetrators in the act before they do actual damage to resources.

Embracing AI with IDS is considered by many researchers as a promising approach. The purpose of incorporating AI is to obtain significantly better and more reliable results. Its use began in 1986 [22]. One promising machine learning algorithm example is GP. This was employed in 2010 for synthesising robust signature-based detectors for IDS by Sen et al. [23]. The results reveal GP can perform better than other AI techniques (SVM and Decision trees) as a lightweight method for detecting known flooding and route disruption attacks against the routing protocol AODV [23].

The backdoor detection system using an ANN and GA was proposed by Salimi and Arastouie in 2011 [24]. They developed a novel approach to reveal backdoor attacks using the system and network behaviour. Previous research on machine learning in IDS shows encouraging results and motivates us to explore its use in-depth in this research.

1.1.4 Brief Overview of Machine Learning

Machine learning is a branch of AI where systems can engage in self-learning from data, identify patterns and then make the decisions with minimal human intervention. Machine learning systems can "learn" as they gradually improve their performance on a specific task (using data) without having to be directly programmed.

Machine learning techniques can be used to develop highly efficient power detectors for sleep deprivation or battery exhaustion attacks such as Denial-of-service power attacks, Malware attacks, Spyware attacks and so forth. Narudin et al. reported that malware could be detected using a combination of a Bayes network and a random forest [25]. Other researchers used ANN, GA, decision trees, SVM, and fuzzy logic to build their IDS [26], [27].

Evolutionary Computation (EC) has achieved promising results for IDS in previous research [24], [26], [28]. The details will be discussed in Chapter 2. An EA is a form of EC, a general-purpose population-based metaheuristic optimisation algorithm. The mechanisms used in an EA have their roots in biological evolution: reproduction, mutation, recombination, and selection. Perhaps the highest-profile EC algorithm is the Genetic Algorithm (GA) GA. Genetic Programming (GP) is a particular form of GA, where the essential solution representation is a tree.

1.1.5 Mobile Phone Platform

The mobile phone OS also provides a platform for developers to create applications or 'apps' (software programs developed for smartphones that can carry out specific functions). There are 2.1 million apps available for the Android platform and 2 million at Apple's App Store in 2018 [29]. Varieties of mobile phone operating system are available, e.g. Android, Windows, iOS, and Symbian.

Android is a software bundle (operating system) for mobile phones that contains an OS, middleware, and critical applications based on Linux OSs [30], its development starting in November 2007 [31]. It permits developers to write code in a Java-like language using Google-developed Java libraries. Various Android platforms were made available under [32]the Apache free-software and open-source license since its official release in 2008: Cupcake, Donut, Éclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean 4.1 [31], Jelly Bean 4.3, KitKat, Lollipop, Marshmallow, Nougat, Oreo, Pie, Android 10, and Android 11 [33]–[36].

The Android platform was intended to offer an application environment that ensures the security of users, data, applications, the device, and the network. Android users also have the ability to utilise and control applications. The design includes the expectation that attackers would attempt to perform common attacks, i.e. social engineering attacks to persuade device users to install malware and attacks on third-party applications [32], [37], [38].

1.1.6 Summary

The modern mobile phone stores and processes highly valuable data and has become a highly attractive target for attack. A number of researchers have sought to exploit AI in providing effective IDS. However, the area is underexplored, and in particular, we might ask how AI can be used to provide a robust detection framework that targets mobile phones. The restrictions on resources provide challenges to develop malware detection for mobile phones, i.e., the actions taken by the mobile phone itself cannot be resource hungry. Inevitably, there are trade-offs to be made between the effectiveness of a detector (appropriateness of generating alarms) and its non-functional properties such as the speed of detection and consumption of other resources such as power and memory.

The Android framework has been chosen as the vehicle for experimentation since there is a very active development community, it is a widely used platform, and it is entirely open source. If research results prove promising, a natural community would be interested in further development. It is acknowledged that the fundamental research questions addressed would generally apply to other mobile phone platforms.

Furthermore, since mobiles are low resource platforms, we want to evolve efficient detectors where power and execution time, for example, are important criteria — accordingly, the aim to use GP in a multi-objective context. Sen et al. stated that a Multi-Objective Evolutionary Algorithm (MOEA) could allow the combination of multi-objective optimisation and evolutionary search [23]. We hope the combination between GP and MOEA can produce detectors with excellent trade-offs between detection and resource usage.

1.2 Thesis Hypothesis

The overall thesis hypothesis is:

> Supervised learning with Evolutionary Algorithms (specifically genetic programming and the multi-objective evolutionary algorithm SPEA2) can be used to synthesise a system capable of detecting a wide range of attacks on mobile phones and do so efficiently, e.g., speedily and using limited battery power.

This research will implement GP to evolve programs to distinguish malicious and non-malicious applications in mobile datasets and demonstrate the use of GP and MOEA together to explore functional and non-functional trade-offs. Results will be compared with those of the research community (whose mobile attacks datasets we acquire and make significant use of for case study purposes). The wide range of attacks included mobile malware, ransomware and mobile botnet (datasets we acquire from the research community). This research also seeks to use only permissions as features in the GP program. Permissions in Android Package format (APK) are easily accessible. If successful, we will be able to demonstrate highly effective and efficient detection.

1.3 Brief Overview of the Thesis Chapters

The remaining chapters of this thesis are outlined below:

Chapter 2. Concepts and Related Work. This section provides an overview of mobile phones, identifies the general vulnerabilities and threats faced by Internet users and identifies specific threats to, and attacks on, mobile phone users. It also explains the IDS concept in general and details extant IDSs for mobile phones. It identifies significant issues in IDS, both general and specific to the mobile phone.

Chapter 3. **Evolution of Malware Classification and Detection in Mobile Phones**. This chapter starts with the background of EC and the method applied in this research – GP. This chapter also assesses the effectiveness of applying the GP technique. The discussion of why this method is chosen is also provided in this chapter. The particular sources of attacks targetted (i.e., malware) by the evolved detectors are explained.

Chapter 4. **Performance Evaluation of Genetic Programming on Mobile Phones Intrusion Detection System**. This chapter explores the ability of GP to synthesise detectors to identify malware and non-malware APKs. The effects of tuning the GP fitness function weight parameters are also discussed to identify whether it affects the detection performance.

Chapter 5. Improving Resource Efficiency. This chapter focuses on non-functional properties such as power consumption and execution time. Non-functional properties such as there were identified in the literature as a major problem. The combination of MOEA and GP is evaluated to explore trade-offs between a specific functional property (detection rate) and two important non-functional properties (execution time and power consumption).

Chapter 6 Further Investigation of GP Performance on Various Threats on Mobile Phones. The goal of this chapter is to investigate the GP performances on other datasets that are acquired from the research community. (These too can be downloaded with sufficient acknowledgement of the researchers.) Three datasets are investigated in this chapter. These concern malware, mobile botnets and ransomware.

Chapter 7 Conclusion. This chapter concludes the thesis. It presents a discussion of the contributions of the research and identifies future work.

1.4 Summary of Thesis Contributions

The research contributions of the thesis are made in Chapters 4, 5, and 6, as summarised below:

- Chapter 4. This Chapter establishes proof of concept that GP can be used to synthesise effective classifiers to distinguish malware and non-malware applications in Android mobile applications datasets. It also establishes the importance of weight selection in the parametrization of the fitness function used. Optimal choices of parameters may vary between target malware types.
- Chapter 5. This chapter demonstrates empirically how optimisation can be used effectively to investigate trade-offs between functional properties (detection rate) and non-functional properties such as execution time and power consumption. It shows how a specific multi-objective algorithm (SPEA2) gives the best trade-offs between three important objectives (detection rate, power consumption and execution time).
- Chapter 6. This chapter evaluates our GP based approach to three new datasets, demonstrating that the approach can generalise to further malware applications including botnets and ransomware.

CHAPTER 2

Concepts and Related Work

This chapter starts with an introduction to mobile phones, followed by an overview of operating systems in mobile phones and Android security. In order to build up security solutions that are suitable for this environment, the understanding of mobile phone vulnerabilities and the way mobile phones can be attacked are identified and explained in Section 2.3. Section 2.4 and afterwards focus on IDSs, and lastly, the significant issues in IDS are reviewed.

2.1 Introduction to Mobile

Mobile phone technologies started by the invention of the fully voice-based mobile radio system and the first cellular phone system in 1928 and 1977, respectively [39]. In 1955 the first mobile telephone was launched in Europe, and it is followed by Nokia 3210 in 1999 [39]. The 'elegant style' of Nokia 3210 design was a trigger to other mobile companies to upgrade their mobile phone designs. Verizon started the revolution of 3G in early December 2001 [39].

Desktop computers and laptops manage their hardware, software resources, and memory for running multi-tasking programs using software known as an OS. The OS is a critical part of the system software in a computer system. Modern smartphones, which are inherently highly complex mobile computing platforms, require highly sophisticated OSs to service the needs of the many functions they seek to provide. The mobile phone OS also provides a platform for developers to create applications or 'apps' (software programs developed for smartphones that can carry out specific functions).

In the mid-1990s, a few companies attempted to build and market personal data assistants (PDAs). PDAs are not considered to be mobile computing devices, but they were the forerunner to current smartphones [40]. Research in Motion (RIM) introduced the Blackberry (1999), which started as a straightforward two-way pager but expeditiously became the most widespread mobile computing device [40].

Microsoft then followed with their first OS mobile device known as Pocket PC 2000 installed with Windows CE in 2006 [40], [41]. Several hardware manufacturers (Nokia, Ericsson, Panasonic, and Samsung) decided to cooperate on a typical mobile OS known as Symbian. In January 2007, Apple revealed the iPhone, which became the first smartphone used by the general community [36], [40]. Then in September 2008, Android was initially released by Google as a rival to iPhone and its iOS [31], [36].

2.2 Mobile Operating System Security

There are four major mobile OSs: Android mobile security, iOS security, Windows mobile OS security and Symbian OS security. Nevertheless, here we only discuss the security of Android Mobile Security as in this thesis, we are investigating Android mobile threat.

2.2.1 Overview of Android Mobile Security

Android is a mobile platform created by Google and the Open Handset Alliance [31], [36], [42]. Android versions are typically represented by 'dessert-style and sweets' names starting with Cupcake, Donut and leading to the latest such as Pie [31], [36], [37], [43]. Android delivers an open-source platform and application background for cellular devices. Android platform security is divided into kernel security and application security. The kernel security for the Android platform is associated with the Linux kernel, along with a protected inter-process communication (IPC) facility to allow secure communication between applications running in various processes [44].

Android applications are able to access only a limited selection of system resources. Google included a collection of cloud-based services that are accessible to suitable Android devices with its Google Mobile Services. The security services provided by Google [44] are: Google Play - this is an accumulation of services that enable users to discover, install, and purchase applications from their Android devices; Android update - this delivers new functions and security updates to preferred Android devices; Application service - frameworks that enable Android applications to utilise cloud functionality; Verify Apps - alert or automatically prohibit the installing of harmful applications and continuously scan applications; SafetyNet - a privacy-preserving IDS to help Google monitor and mitigate known security risks as well as determining new security threats; SafetyNet Attestation - the devices are determined to be either CTS compatible or not using Third-party API (Android Package Interface); and Android Device Manager - lost and stolen devices can be located using a web app or Android app.

The Android security program contains design review, penetration testing and code review, open-source and community review, incident response, and monthly security updates. Android integrates industry-leading security features and works jointly with the developers and device implementers to ensure the Android platform and the environment is safe. A resilient security model is essential to allow a dynamic ecosystem of applications and devices constructed on and across the Android platform and sustained by cloud services [44].

Since the early days of Android development, several improvements have been made to enhance security performance. Google gave assurances throughout 2014 – 2020 that the team was dedicated to ensuring Android is a safe environment for users and developers; would preserve the security and privacy of all Android users; and would enhance the security of the platform [38], [45]–[48].

In September 2020, the Android Security Team made a promise to protect every Android user as the team introduce the final release of Android 11: which offers layered security (all parts of Android system work together to build a strong defence that runs smoothly and effectively), transparency and openness (keeping users up to date and sharing knowledge amongst Android community), and backed by Google (partnered with other experts to keep Android devices safe)[32].

2.2.2 Conclusion

Android assists the developer in writing secure applications. It offers a type-safe language (Java), an enhanced security model, a successful class library, and a robust set of applicable and securable concepts for development on mobile phones. Android's framework characteristically defaults to safe behaviour unless the developer unambiguously chooses to share data between applications. Android's open design means that finding and fixing security flaws is performed by a wide range of people. The essential data protection is in place to ensure that if a device is lost, then encrypted data is not recovered.

Android is used plenty of open-source components, several of which have vulnerabilities which can be used by attackers to invade the smartphones and access all user's data and info. Linux and WebKit both have required abundant security fixes within the last few years, but this is not an issue for many application developers who seem to appreciate honesty and speedy fixes. Also, some specialists believe that making the code closed-source to avoid open inspection would be a delusion.

2.3 Threats and Attacks

2.3.1 Threats Faced by Users While Using the Internet or Networks

The 21st Century has seen an explosion in communication around the world. In the past, people needed to use letters or telegrams that took a few days or longer to reach the intended recipient. Communication in the 20th century drastically reduced communications times with the emergence of the telephone, radio, fax, mobile phone, and email. The 21st Century has seen the emergence and rise of the webcam and smartphone, placing sophisticated real-time communication in the hands of over one billion people. However, the downside of such an explosion is that there are now over one billion platforms available to attack!

We have accustomed ourselves to our communications being confidential, unmodified, and reliable. A variety of mechanisms has been developed to ensure that such properties hold, for example, 'seals' have been used for thousands of years to ensure the integrity of documents. Threats can be thought of as potential compromises of the above (and similar) properties. Thus, for example, a breach of confidentiality is a threat, unauthorised modification is a threat, as is lack of availability.

Attackers try to take advantage of each weakness in a system to fulfil their purposes [17], [49]. Intruders can realise security threats using a variety of communications methods. Attackers may use a range of tools, scripts and programs to damage our communication systems [49]. Threats are continually being discovered; effective intrusion detection remains a persistent and challenging problem and is likely to remain a significant research field for a considerable time to come.

The detection of attacks has been an essential goal of computing since the early days of computing. Many authors have sought to characterise the nature of attackers and their attacks. In 1980, Anderson produced a technical report that is cited by most IDS researchers. The report is based on audit trail analysis and divides possible attackers into four categories: external penetrators, misfeasors, clandestine users, and masqueraders [17].

Axelsson and Lunt defined external penetrators as users who gain unauthorised access to a computer of which they are not a valid user [50], [51]. For example, attackers may try to get a username and password using illegal software to get access to a computer or account subsequently.

Misfeasors, clandestine users, and masqueraders can be identified as internal penetration agents. Internal penetrations are more common than external ones. Misfeasors are legitimate or valid users of the system who misuse their privileges [50], [51]. For example, a user having access to a computer or system account but trying to manipulate his or her privileges by editing or copying private data on the data server without getting permission.

Clandestine users try to use supervisor privileges to avoid being captured and escape auditing access controls [50], [51]. Attackers may try to use someone else's account that has supervisor privileges; they then may damage or exploit private data or information on a server. They are challenging to catch because they have used another's authorised account. It is nearly impossible to detect clandestine users by standard audit trail methods [17].

Masqueraders can be either internal users or external penetrators [17], [50]. They can control computers or systems using another user's username and password [51]. They pretend to be legitimate users. Thus, if I can obtain your authentication credentials, I can log onto a system as you, i.e. I masquerade as you. Masqueraders can be detected by how their behaviour differs from that of authorised users; for example, the attacker might spend most of his time browsing directories and executing system status commands, while an authentic user might focus on editing or compiling and linking programs [22], [52].

2.3.2 Threats Faced by the Mobile Phone User

In Q2 2020, Kaspersky detected 1,245,894 malicious installers compared to Q2 2019 at 753 550 malicious installers have been blocked [53]. Kaspersky reported in 2018 that they blocked 796,806,112 attacks from online resources in 194 countries [54]. For mobiles, they detected 1,322,578 malicious packages, 18,912 installation packages for mobile banking Trojans, and 8,787 installation packages for mobile ransomware Trojans. In 2017, Kaspersky was detecting 280,00 malware file per day compared to in 2014; they detected nearly 3.5 million malware on 1 million user devices [12]. McAfee [55] reported that malware is targeting Google Play on mobile devices are enlarged in all quartile in the year 2017 than 2016 and 2015, as shown in Figure 1.1.

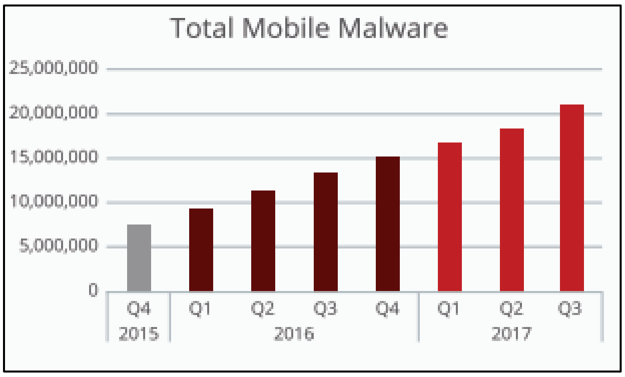


Figure 2. 1 Total Malware Samples from 2015 - 2017 on Google Play [55]

Therefore, the threats challenged by mobile phone users are discussed in this section. Consequently, reported cases had been increased from 2015 to 2018. Attackers used four tactics to infiltrate mobile phones with malware: infected applications (injecting the malware in applications release in third-party app stores); malvertising (implanting malware to valid online ad networks to aim an extensive range of end-users); scams (redirecting users to malicious websites via email, text messages or pop-up screen); and direct to the device (the hackers directly install malware when the mobile phones are left unattended) [56], [57].

Viruses, worms, Trojans, and bots are all types of malware; malware is a short form for malicious software, also acknowledged as malicious code or malcode [58]. The abilities of malware include harming, disrupting, stealing, or in general perpetrating some other "evil" or illegal activities on data, host or networks. Attacks have become more "innovative" and "wicked".

Batteries power mobile phones and smartphones. The challenge comes from limited battery life - the batteries need to be recharged when drained [59], [60]. Dhaliwal explained that infected mobile phone batteries could be drained faster than before, and the phones could start overheating [57]. A 2013 survey identified resource-draining via installed third-party insecure applications as a major problem in smartphones [60]. A smartphone may run many programs in the background that can have a significant effect on battery life. Any security solution must reflect this constraint.

A big problem for the mobile phone owners is private data leakage [11], [61]. The thieves or attackers can steal private information saved on mobile phones such as credit card or bank information, and corporate data. The mobile phone's security system can be defeated by sophisticated intruders if there is enough time [10]. For that reason, wiping or securing private data from an intruder should be considered (and indeed such functionality is available on major mobile phone platforms).

The mobile phone may be stolen from its owner or otherwise physically lost [9]. Mobile phones are portable, making them easy to drop or be stolen without the owner realising [10]. A mobile phone might contain many pieces of private data, such as pictures, account information, personal data, text messages, and contact information.

2.3.3 Attacks on the Smartphone

Companies are keen to offer ever-more functionality. Security, however, is rarely a primary development goal and as a result, many phones find themselves the target of attacks. Mobile phone users face attacks such as sniffing, spam, phishing vishing, smishing, pharming, and attacker spoofing [9], [14].

Sniffing occurs when the attacker captures and decodes packets as they pass through the network [9]. Sniffing is a form of data interception. A well-known example is packet sniffing, also known as eavesdropping [49]. Usually, the attacker will sniff or tap smartphone calls or SMS (short message services), but 3G and 4G users are also at risk. The attacker may also try to steal a username, password, content of an email and transfer files while sniffing on the same network as the mobile phone user.

The attackers can use email or MMS (multimedia message services) to spread Spam. A DoS attack can also start via MMS spam [9]. Social media users can distribute spontaneous messages and produce burgeoning traffic in social networks; this is often referred to as Social Network Spam [62]. An example of spam activity would be the direction of users to malicious commercial sites such as pornographic webs sites by a link posted by friends on a social network.

Phishing is a criminal act where an attacker, masquerading as a trusted party, steals privacy-related information, such as username, password, or credit card details [9], [14], [61]. Phishing also can be defined as "endeavouring to obtain personal information by masquerading as a trustworthy individual in electronic communication." Phishing can be circulated via email using attractive email subject and content to convince users to open the email and so become victims. In 1996, one of the first known phishing attacks was traced when hackers attempted to misappropriate America Online passwords from online users [63].

Georgia Tech first piloted phishing awareness training using the 300-member Office of Information Technology (OIT). One out of every four people clicked on the link in the phishing email message and could have had their system compromised [64].

Automated messages claiming to be from a bank were used to extricate the details of bank accounts and were dubbed 'vishing', a merging of 'voice' and 'phishing'. Vishing or "phishing voice calls" use voice calls to abuse an individual's trust in telephone services, as the prey often does not suspect that fraudsters can use methods such as caller ID spoofing and complex automated systems to initiate this type of scam [18]. Spoofing caller IDs can be accomplished by using voice over IP (VoIP) technology; attackers were able to exploit the public belief in the landline system. Vishing leverages the power and trust invested in voice communication to allow the intruder to gain access to a smartphone user's financial and other private information [9], [10].

Smishing is a form of the attacker that exploits SMS, or text, messages. Text messages can contain links to such things as web pages, email addresses, or contact numbers that when clicked, may automatically open a browser window, email message, or dial a number [14]. A smishing attack usually attacks the nationwide bank as fraudsters dispersed their SMS spam to broader mobile users with an account at one of the banks [63].

In pharming, the user's web traffic is redirected to a malicious or fraudulent website [9]. Pharming is an example of an advanced phishing attack and is used for online identity stealing. It is also known as "phishing without lure" [65]. The attacker may use information gained from pharming to hone his attacks. Pharming attacks are hard to detect because the fake visited URL sites closely resemble those of the legitimate websites [65]. The attackers corrupt DNS information to redirect users to a forged website under their control.

Spoofing is a threat where attackers pretend to have a particular caller ID and act as an authorised person to get private information from mobile phone users [66]. A spoofing attack is a circumstance in which one person or program effectively masquerades as another by fabricating information and thus gaining an unauthorised advantage. Example attacks are IP address spoofing, DNS spoofing and ARP spoofing. Bellovin identified the first IP address spoofing attacks in 1989. In his article, he described a variety of IP spoofing attacks and presented defences against them using encryption, authentication and trusted system functionality [18].

2.3.3.1 Android Malware

Malware attacks increased significantly in 2019, which is 50% expand from 2018, in particularly affected Android mobile phones [67]. In November 2020, Android mobile phone users in Southeast Asia have been targeting by a new variant of Android malware called WAPDropper [68]. It just a month after Brazil being 'bombed' with a trojan called Ghimod that aiming Brazillian Bank apps [69]. This trend shows that the malware developer is working hard to create a new strain of malware in a short time. The main purpose of this cybercriminal is to steal payment data, confidential information, login credentials, phone number or email address, contact lists, device information and money from victims bank accounts [67], [69]–[71]. Furthermore, Hautala identifies four signs of malware that had been hidden in Android malware to alert Android mobile phone users [71]. The signs are; users will frequently seeing ads (irrespective any apps they are using), the app's icon will vanishes after users install an app, the mobile phone battery drain faster than before, users will see apps they not installed in their phone [71].

Kaspersky detected the first wild Android malware in 2010, known as FakePlayer[3] [15]. Fakeplayer[3] and WAPDropperis allowed Android devices to send SMS messages to premium numbers [15], [68]. Others malware families that have been used in this investigation are details explaining in Chapter 4 and Chapter 6. Some other popular varieties of mobile malware are adware, banker malware, ransomware, rooting malware, SMS malware, spyware and Trojans capabilities as explained below [54]–[57], [72], [73]:

Adware – displays continual ads to a user in the shape of pop-ups, sometimes causing the unintentional forwarding of users to websites or applications.

Banker malware – happenings to steal users' bank authorisations without their knowledge.

Ransomware – requests money from users to restore the data or the functionality of the devices being locked.

Rooting malware – 'roots' the device, basically unlocking the OS and gaining extended privileges.

SMS malware – controls devices to send and intercept text messages resulting in SMS charges.

Spyware – observes and records info about users' activities on the devices without their knowledge.

Worms — can reproduce themselves repeatedly and execute without user interaction.

Trojans —hidden malicious functionality within legitimate software. Once activated, it can compromise phone data and operation.

Android mobile users can take some steps to stop mobile malware; keep update the software updated, uninstall apps that users think malicious, install antivirus apps, and not install Android apps from third-party app stores [71].

2.4 Intrusion Detection Systems (IDSs)

Internet access is essential, not only for computers but also for the population of mobile phone users. McKinley reported that 3.9 billion people were unable to connect to the Internet and benefit from its services. This included 19.4 million people in rural America, even in 2018 [74]. Protecting the private data and information associated with these services is now vital, according to Dixon, Gordon and Marceux. The Internet has now become a necessity, and access to it is a human right because of the capabilities of the Internet can offer to the users [75], [76]. Researchers and antivirus companies have sought to find better countermeasures for the attacks at hand as the users of the Internet have grown in number, and their vulnerability to attack increased.

The detection of attacks is a crucial component of providing adequate security. An IDS comprises devices or applications capable of detecting abnormal events, potential security accidents, and intrusions in the computers or network systems and sends warnings or alerts to the administrator to counter the identified intrusion [18], [20]. An IDS can be viewed as a second protective layer besides firewall and antivirus, and the fundamental goal of IDSs is to detect intrusion.

Kemmerer and Vigna denied that IDS could detect an *intruder*, preferably it can detect only the sign of *intrusions*, either when they happen or afterwards [77]. After more than a decade, KhorasaniZadeh et al. issued a new interpretation of the goal of an IDS: the IDS is to offer and improve the entire security and robustness of computer structures [21].

Intrusion detection can be classified based on audit data as either host-based (HID) or network-based (NID). HIDs operate on a personal computer or device in the network and NIDs observe network traffic to/from the network [78]. Anderson discussed the significance of auditing and audit trails in host-based intrusion detection [17]. Network intrusion detection uses packet sniffers to read and analyse packet exchanges between hosts, usually deployed by broadcast type networks (TCP/IP). Host-based intrusion detection monitors the hosts themselves and responds to attacks on them. HIDs usually combat internal threats, i.e. intrusions by dishonest employees.

2.4.1 Intrusion Detection Systems Approaches

In the early stages, two Intrusion Detection approaches were commonly discussed by researchers: anomaly-based detection and misuse based detection [79]. Each approach has strengths and weakness. The evolution of systems and attacks on them gave rise around 1999 to the third type of approach: specification-based detection [50], [78]–[80]. Specification-based detection derived strengths from both anomaly-based detection and misuse-based detection. We discuss these below.

Anomaly-based detection refers to the identification of patterns of behaviour that deviate from some 'norm' or expected activity [81], [82]. Any events that violate normal behaviour are considered suspicious. Profiles of normal behaviour may draw on a variety of data, such as normal login time, duration of the login session, CPU usage, disk usage, and favourite editor [79]. The IDS observes current user activities and characterises them as anomalous (suspicious) or otherwise using the profiles. Deviations from profile norms will be flagged as potential intrusions. For example, a normally passive public website suddenly attempting too many open connections may be infected by a worm. There are four methods used in anomaly-based detection: statistical, predictive pattern generation, neural networks, and sequence matching and learning [21]. Anomaly-based detection suffers from accuracy problems. It is hard to pin down "normality"; often, anomalous behaviour is found on further analysis to be entirely respectable. At some level, every user session can be thought of as unique; it is a question at which level of granularity attackers chose to observe information. A user's day is rarely (never) "exactly" the same as the last. Finding an optimum level at which to summarise (profile) information and relate it genuine suspiciousness is hard.

Misuse-based detection (also known as signature-based detection) looks for known attack patterns in the current events or activities [21], [83]. Misuse-based detection must retain attack details (signatures) that correspond to identify attacks. The signatures can be developed using a range of techniques, from hand translation of attack manifestations to automatic training or learning using labelled sensor data [84]. Detection using this approach is often accurate because the signatures stored may summarise unambiguously mischievous behaviours. For example, an HTTP request linking to the .exe file may indicate attacks to a personal computer. Approaches to misuse-based detection include expert systems, keystroke monitoring, model-based, state transition analysis, and pattern matching [21]. Misuse-based detection is unable to detect accurately novel attacks. It is considered a major weakness for misuse-based detection IDS.

Specification-based detection monitors the executing program based on programming signatures of benign behaviour, instead of an existing precise outbreak pattern [50], [85]. It is mostly used in ad hoc networks, and it is suggested as a way for different types of ad hoc routing protocols to be updated [86]. It essentially monitors for deviations from *specified* behaviour. It is a natural candidate when protocols are monitored(because they have specifications. DoS attacks are not detected by specification-based detection because these types of attacks follow specifications. Specification-based detection can produce a lower rate of false alarms than anomaly-based detection, but it cannot compete with anomaly-based detection for novel attacks detection.

2.4.2 Intrusion Detection System Performance Metrics

The performance of IDS can be calculated based on the confusion matrix. The confusion matrix contains four values such as True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). The explanation of the confusion matrix as below:
TP rate refers to the proportion of correct malware detected.
TN rate refers to the proportion of correct for non-malware detected.
FP rate refers to the proportion of incorrect predictions for non-malware detected.
FN rate refers to the proportion of incorrect predictions for malware detected.

2.4.3 Related Research on Intrusion Detection Systems

Table 2.1 summarises research on IDS; the main focus is intrusion detection for mobile phones. It aims to show IDS development and trends and indicates characteristics of the approaches described.

Table 2. 1 Summary of Research on Intrusion Detection Systems

Researches	Year	Way of Detection
Anomaly-based Detection		
IDES [22], [52]	1988	● detection by constructing a profile for a group of users that behave in the same manner by their organisational status and attempt to correlate behaviour for a particular user. ● detect not only with the past behaviour for a certain user but also with the behaviour that is recorded as "normal" for that group
NIDES [87]	1995	● persistent storage ● statistical analysis ● rule-based analysis ● batch analysis ● agend ● arpool ● archiver ● user interface ● agen ● resolver

Researches	Year	Way of Detection
Misuse-based (Signature-based Detection)		
MIDAS [88]	1988	● Attempted break-ins ● Masquerade ● Penetration ● Misuse ● Trojan horse/ virus
USTAT [89]	1993	● audit collection / pre-processing ● knowledge base ● inference engine ● decision engine
Specification-based Detection		
Haystack [90]	1988	● Attempted break-ins ●Masquerade attacks ● Leakage ● Denial of service ● Malicious use ● Penetration of the security control system
EMERALD [91], [92]	1997 – 1999	● service analysis level ● domain-wide level ● enterprise - wide level
FloGuard [93], [94]	2010 - 2011	● detectors vigorously in a cost-effective manner ● online system-wide forensic

Table 2. 2 Summary of Research on IDS using AI Approaches

Researches	Year	Data source	Methodology	Metric	Results
Anomaly-based Detection					
ISABA [95]	2010	KDD99 dataset (normal, probe, DoS,U2R, R2L)	Naïve Bayesian classifier	Detection Rate	99.82%,99.72%,99.49%,99.47, 99.35%
Misused-based Detection (Signature-based Detection)					
Artificial Neural Networks [96]	1998	RealSecure™ datasets (source address, a destination address, packet data)	Artificial neural networks	Mean Square & Correlation	0.058298, 0.069929 0.982333, 0.975569
NEDAA [97]	1999	DARPA Intrusion Detection EvaluationData	AI and Decision trees	Not reported	Produce sets of rules for compilation into the expert system.
Power-Aware IDS in MANETs [23]	2010	Simulated networks (flooding attacks, route disruption attack)	GP and MOEA	Detection Rate & False Positive Rate	98.65%, 100%,93.29% 1.23%, 0.63%, 4.56%

Researches	Year	Data source	Methodology	Metric	Results
Misused-based Detection (Signature-based Detection)					
Backdoor detection system [24]	2011	Dynamic - system behaviour and network traffic	ANN and GA	Not reported	Not reported
Specification-based Detection					
Artificial Intelligence Techniques Applied to Intrusion Detection [98]	2005	Dynamic – using SNORT	Neural networks, Fuzzy logic with network profiling and Data mining	Not reported	Not reported

2.4.4 Evolution of Intrusion Detection Systems

James Anderson introduced the IDSs concept in 1980. The technical report [17] discussed the importance of auditing and audit trails for host-based intrusion detection. From the early 1980s, SRI International started their IDS research led by Dorothy Denning. The US government supported their project. Denning's work resulted in the presentation of the Intrusion Detection Expert System (IDES) in the IEEE Symposium on Security and Privacy (1986) [22] as reported earlier. IDES used audit trail analysis based on its user profiles database; it is an implementation of an anomaly-based IDS. It used AI for encoding an expert's knowledge of known patterns of attack and system weaknesses (if-then rules) [22], [52].

In 1988, the Haystack project was developed for US Air Force [90]. It compares audit data to define patterns. The Haystack was the first DIDS (Distributed Intrusion Detection System) for client and server track that focused on the detection of insiders (legitimate users that misuse their privileges). The first Network Intrusion Detection Systems (NIDS) began in 1990 with research by University of California researchers [99]. They developed the Network Security Monitor (NSM), which contributed to Distributed Intrusion Detection System (DIDS) development, and one of the early IDS that considering using hybrid intrusion detection. The researchers focused only on the security-related issues in a single broadcast LAN such as Ethernet. Their system used a four-dimensional matrix with the following elements: Source (a host which generates traffic), Destination (a host to which traffic ID destined), Service (example: mail and login), and Connection ID (a unique identifier for a specific connection) [99]. The NSM was also the first system to use network data directly as source input.

The DPEM (1994) became the successor of IDS and used a policy-focused anomaly-based detection approach [50]. In 1999, the Snort (Lightweight Intrusion Detection for Network) was initiated and became the first commercial NIDS [100]. It is a lightweight and flexible intrusion detection tool for small, lightly utilised networks. Snort is a free IDS and can be used in any environment, and no cost needed to deploy a commercial NIDS sensor [100]. It has also acquired a very high public usage profile.

In 2011, the researchers proposed the AI approaches based on a combination of ANN and GA to identify malicious code (backdoors) in single computers and computer networks [24]. Frank's brief survey of AI approaches used in several IDSs [101] highlights network-based intrusion detection as the biggest problem in intrusion detection. Frank also demonstrated how feature selection could be used to reduce overheads and improve classification of network connections. The testing of feature selection used data collected from the Network Security Monitor (NSM). Feature selection is an essential element of the solution by AI techniques of many pattern recognition problems.

Manninen explored how to create an IDS environment that acknowledged the preferences of a security officer, seeking to make the security officer's work more effective and practically informative by displaying the most viewed anomalies first [102]. He compared the AI-based IDS with traditional IDS solutions and analysed how the AI-based solution might be implemented in the IDS. He proposed three outcome groups: usability of the learning process (introduction of noise into data in different cases); ways to detect intrusion based on learned examples (response to noise in the data); and showing the events to the security officer in the correct order (minimising false alarms). The results showed that AI-based solutions could be used in IDS. Neural networks are the most popular choice of AI implemented in IDSs. Understanding and handling the noise in the learning data to test the accuracy of IDS was highlighted as future work.

SVMs are learning models with a learning algorithm that analyses data and recognise patterns, used for classification. The basic SVM takes a set of input data and predicts, for each given input, which of two possible classes forms the output. Given a set of training samples, each marked as belonging to one of two categories; an SVM training algorithm builds a model that assigns new samples into one category or the other. It represents these samples as points in space mapped so that a clear gap divides the samples of the separate categories. Adigun et al. research proves that the use of SVM with Particle Swarm Optimization (PSO) can reduce the computational loads [103]. The Support-Vector Machine (SVM) technique–based can be used for both anomaly-based and signature-based detection [103]–[105].

Some researchers suggest that Intrusion Prevention Systems (IPSs) are a better solution rather than IDSs because an IPS can identify and counter the intrusion to protect the network at initial stages of an attack. IPS is an extension of IDS combining both the function of firewall and intrusion detection. However, IPS has issues, which affect the system: completeness of any signature database, traffic volume, topology design, quota usage logging, protecting the IPS, and managing the monitoring sensor [106]. Therefore, the IDS still a preferred security solution for devices and networks.

2.5 Intrusion Detection Systems in Mobile Phones

The smartphone can also function similarly as a computer; it is hackable and can be attacked by viruses and malware much as with other platforms. As smartphones become more complex and powerful to provide increased functionality, security concerns are increasing.

A decade ago, the compromising of major computer systems around the world via viruses and other malware made many security experts think that malware for mobile devices would subsequently emerge as a major problem [107]. Indeed, mobile malware and privacy leakage remain significant threats to mobile phone security and privacy. Smartphones are not usually equipped with built-in antivirus software, making them more vulnerable to attack. The mobile phone has now become a favourite platform to attack. Consequently, the development of malware and virus detection for mobile phones is becoming urgent.

IDS for mobile phones has received a fraction of the attention of IDS for networks and more traditional host platforms. Nevertheless, researchers have sought to use IDS based on the three approaches reviewed earlier in this report referred to in Table 2.2 and Table 2.3.

In [108], [109], they suggested some other mobile malware detection such as static analysis, dynamic analysis, hybrid analysis, application permission analysis, cloud-based malware detection which is currently in trend to be used by the researchers worldwide.

Table 2. 3 Summary of Research on Mobile Phone IDS

Researches	Year	Data source	Methodology	Metric	Value	Mobile Platform
Anomaly-based Detection						
SmartSiren [110]	2007	SMS trace collected from a national cellular service provider in India	Statistical Monitoring, Abnormality Monitoring	Not reported	Not reported	Windows Mobile
Tap-Wave-Rub (TWR) [111]	2013	Dynamic - Permission and accelerometer data	Intuitive human gesture recognition, Tapping the detection mechanism based on accelerometer data	Detection Rate	94.67%	Android
DAIDS [112]	2014	Dynamic – (package, process, event, network usage, comm, memory usage, CPU usage)	Behaviour analysis	Not reported	Not reported	Android
Android Botnets: What URLs are Telling Us [113]	2015	Android Genome Malware project ,Malware security blog VirusTotal , Samples provided by a well-known anti-malware vendor	Visualisation	Not reported	Not reported	Android

Researches	Year	Data source	Methodology	Metric	Value	Mobile Platform
		(URLs)				
Misuse-based (Signature-based Detection)						
Secloud [93], [114], [115]	2010 - 2013	ClamAV malware signature database Snort network traffic database	Forensic analysis	Accuracy	Not reported	Android
DroidAnalytics [116]	2013	Using Crawler to automatic download application from the repositor	Multi-layer signatures generator	Not reported	Not reported	Android
Specification-based Detection						
Crowdroid [117]	2011	Dynamic – crowdsourcing system calls	k-means algorithm	Accuracy	100% (PJApps), 85%(Trojan)	Android
SBIDF [118]	2011	Dynamic - Simulate the behaviour of real-world malware	Temporal Logic of Causal Knowledge	Not reported	Not reported	Android

Table 2. 4 Summary of Research on Mobile Phone IDS using Artificial Intelligent Approaches

Researches	Year	Data source	Methodology	Metric	Value	Mobile Platform
Anomaly-based Detection						
Malware Detection using Machine Learning [105]	2010	MIT Reality Mining project - phone calls, SMS, and communication logs	Support Vector Machine (SVM) algorithm, Statistical classification model	Accuracy	Not reported	Symbian
DREBIN [119]	2014	Android Malware Genome Project, Google Play, Russia Market, Chinese Market and Android websites. (i.e.API calls, Intents, permissions)	SVM	Accuracy False Positive Rate	94% 0.1	Android
Ransomware Steals Your Phone. Formal Methods Rescue It [120]	2016	Contagio Mini Dump, Ransom Mobi,DREBIN	Formal Method	F-measure Accuracy	0.99 0.99	Android
Identifying malicious Android apps using permissions and system events [121]	2016	Android Malware Genome Project, Market datasets (permission and system events)	SVM, K-means clustering	True Positive Rate, False Positive Rate, Detection Rate	85.25%, 93.07% 7.12%, 1.13% 85.24%,93.07%	Android

Researches	Year	Data source	Methodology	Metric	Value	Mobile Platform
Anomaly-based Detection						
Effective and Explainable Detection of Android Malware Based on Machine Learning Algorithms [122]	2018	DREBIN	SVM	True Positive Rate False Positive Rate	94%, 94% 1% , 3%	Android
Detecting Application with malicious Behavior in Android Device on GA and SVM [123]	2018	DREBIN	SVM , Genetic Algorithm, N-gram	Accuracy	95%	Android
Misuse-based (Signature-based Detection)						
Neural Fraud Detection [124]	2000	Telecom carrier (i.e. users calls)	Neural network classifier	Not reported	Not reported	Not reported
AmoxID [125]	2012	Dynamic (SMS data, Call Data, GPRS Data)	SVM classification, Pattern recognition algorithms	Not reported	Not reported	Android
MADS [126]	2013	VirusTotal – malware datasets	Naïve Bayes, Bayesian Network, SVM, KNN, J48, Random Forest	True Positive Rate, False Positive Rate Accuracy, ROC Curve	True Positive Rate 0.93,0.71,0.93,0.35,0.83,0.92 False Positive Rate	Android

Researches	Year	Data source	Methodology	Metric	Value	Mobile Platform
					0.17,0.13,0.03,0.02,0.12,0.13 AUC 0.90,0.89,0.95,0.84,0.86,0.96 Accuracy 88.07%,78.68%,94.70%,66.24%,85.54%,89.74%	
MOCDroid [127]	2017	Aptoide, VirusShare, VirusTotal	MOEA, Genetic Algorithm	Accuracy, False Positive Rate	94.6% 2.12%	Android
Coevolution Malware and Anti Malware [128]	2018	Malgenome	Genetic Programming	Detection Rate, False Positive Rate	48.44%, 42.86% 0.00%	Android

Researches	Year	Data source	Methodology	Metric	Value	Mobile Platform
Specification-based Detection						
12 SwarmDroid [103]	2014	NSL KDD dataset	• Support Vector Machine (SVM) classification • Particle Swarm Optimisation (PSO)	Detection time, True Positive Rate , False Positive Rate , and detection accuracy	Accuracy 80.4375%,90.5625%, 93.1937%	Android

2.6 Summary of Major Issues in Intrusion Detection Systems

We can conveniently categorise significant issues as major issues in general IDS and major issues in mobile phone IDS. The big issues in general IDS are: a high rate of false alarm; real-time detection; response to detected intrusions; and IDS sensor placement (efficient sensor placement will reduce the cost). The big issues in mobile phone IDS are coping with limited resources and significant IDS overheads. These are now addressed below.

2.6.1 High false alarm rates

Denning's (1986) paper introduced ideas for creating IDS with low false alarm rates [22]. The false alarm rate must be considered from the early stages of development. High false alarm rates have usually been associated with anomaly-based detection approaches [129], [130]. The rate of false positives in anomaly-based systems is normally higher than in signature-based [26].

Mobile phones are components in a wider MANET, and the high ratio of false alarms in MANETs is an important issue [131]. Patel et al. indicated that triggered false alarms would have a severe effect on the system's operation such as the disruption of information available because of IDPS blockage in suspecting the information to be an attack attempt [82]. False alarms also contribute to low detection efficiency [26]. An IDS cannot give the best response to detecting the attacks. Most algorithms used to detect intrusions try to reduce the false positives and increase the detection rate. However, from previous research, it shows that the higher detection rate, the more false positives will occur, and minimising the false positives is a challenge [132]. There seems to be a natural trade-off being made. Sechi et al. and Michalopoulos et al. agreed that false alarms would be high in IDS in mobile phones [133], [134].

Nevertheless, false alarms can be reduced significantly by using a specification-based detection approach or implementing AI (machine learning) in an anomaly-based detection approach [118], [130]. The specification-based detection approach tries to avoid the high rate of false alarms affected by the legitimate but previously unseen behaviour (not intrusive behaviour, but detected as intrusive) in the anomaly-based detection approach [130]. In the experiments of Uppuluri and Sekar using BSM audit records (corresponding to system calls) and specification-based detection methods, no false alarms were recorded [80]. The experimental results supported their claim that specification-based approaches can detect novel attacks without having to sacrifice on the false alarm front [80].

For system effectiveness, an IDS needs to detect nearly 100% of attacks with minimum false positive detection [77]. However, it seems hard to be achieved because the pattern of attacks usually different and changes as the nowadays attackers are adept at covering their tracks.

2.6.2 Real-time detection

The detection of intruders or attacks has become one of the biggest problems for developed IDS. Intrusion detection should be or be near real-time. Researchers have proposed many models that include AI in their IDS framework to ensure the detection of attackers in real-time. Denning proposed a real-time IDS in 1987 [52]. USTAT was designed to be a real-time system [89], attempting to pre-empt an attack in advance before any damage is caused to the system. This pre-emption is possible only with real-time analysis. Real-time detection can trigger an alarm and invoke a message on the console to react to the intrusion.

For high-speed and high-performance network nodes, IDSs should seek to carry out analysis in real-time [77]. KhorasaniZadeh et al. pointed out that real-time detection advantages are not always achievable [132]. For example, an IDS in a MANET may not be able to react to an attack in real-time due to communication delays [83].

2.6.3 IDS sensor placement

IDSs use sensors to monitor the network for signs of disturbing activity. The problem of IDS sensor placement had been identified in 2000 [84], and in 2014 it was still be mentioned as a big problem for the development of high-performance IDS [135]. McHugh, Christie, and Allen (2000) suggested that sensor placement should be re-examined occasionally to guarantee that the system or network changes have not reduced IDS efficiency [84]. Cost-effective sensor placement in large modern systems requires consideration of many criteria [135]. Chen et al. reported the first experiment using heuristic optimisation techniques to evolve optimal IDS sensor placements in 2010 [136]. The IDSs need a network-wide analysis [77], and often there are constraints on where sensors/probes can be placed on the network.

2.6.4 Limited resources

The greatest challenge for IDSs is to reduce the resources required to carry out their analysis [132]. Smartphones have limited resources [115]. A battery powers a smartphone with a limited life, and that must be recharged when drained [9], [137]. Mobile phones have limited battery and computing resources.

Consequently, many security solutions developed for desktops are not suited for use in mobile phones [118], [125]. Detection must, therefore, be intelligent due to limited battery constraints of these devices [138]. Most recently proposed IDSs for malware detection on Android devices are based on behaviour analysis for anomaly detection [117]. Campbell and Hautala emphasise that mobile phones infected with malware have over an hour shorter battery life than clean phones [71], [107].

Real-time monitoring and unnecessary overheads are problems that have to be considered in the context of mobile phone IDS [105]. Parsing data communication via BT and Wi-Fi to detect malicious activity without incurring high overheads and false positives are challenging [118] because the extra overhead in the processor leads to battery draining in mobile devices [117]. Thus, IDSs developed for the mobile phone should avoid high battery consumption and overheads [103], [112].

Smartphone devices have inadequate energy resources, and so this presents a challenge for mobile phone IDS [125]. Other protection systems such as antivirus need to update their virus signatures from the central repository frequently. Since updating of phone antivirus signatures is energy-expensive, the attackers might try to use newer attack strategies to compromise smartphones [125].

2.6.5 Limited test datasets

The standard data set used for IDS performance evaluation has often been minimal and far from representative of real-world data [132]. Aikelin and Greensmith indicated that constructing a "good" dataset is a significant challenge. A major challenge has collated a dataset without any trace of anomalies. [127]. It is time-consuming and expensive to gather the datasets [132]. Unsurprisingly, datasets made publicly available by the research community are necessarily limited.

2.7 Conclusion

Mobile phones are a modern-day necessity for users and are coming under increasing attack. They have their own specific characteristics, and conventional security techniques do not apply to them. Anti-virus software has been developed for traditional computers and laptops, but mobile phones lack appropriate anti-virus and anti-malware protection. Researchers are now starting to give attention to developing new prevention, detection and response tools for mobile phones.

The surveyed literature shows that there is a pressing need to address mobile phone protection in general and IDS in particular. In this thesis, we aim to address this need for a prevalent form of attack (malicious apps) for a prevalent platform (Android) seeking to leverage perhaps the most promising technology of our age – AI.

CHAPTER 3

Evolution of Malware Classification and Detection in Mobile Phones

This chapter introduces Evolutionary Computation (EC) methods that are used to investigate the malware classification in this thesis. Two EC methods, Genetic Algorithms (GAs) and Genetic programming (GP) are introduced, and the literature detailing the application of EC methods in general to the problem of intrusion detection is surveyed. Justification is provided for the selected method for our development of detectors and classifiers for mobile phones. The proposed approach and the datasets used for its evaluation are detailed.

3.1 Brief Overview of Evolutionary Computation

The term Evolutionary Computation (EC) was introduced in 1991 and is concerned with computational problem-solving methods based loosely on principles adapted from Darwinian evolution [139]–[141]. An EA typically maintains a population of potential solutions ('candidates') and evolves that population using operators analogous to mechanisms from nature, such as *mutation* (where candidates are perturbed in some small way), and *crossover*, where candidate 'parent' solutions swap elements to produce 'children', and *fitness selection* (implementing some variant of 'survival of the fittest') [19] [20].

3.1.1 Genetic Algorithms

John Holland introduced the term Genetic Algorithm (GA) in 1960. Bermermann implemented the fundamental procedure of a GA in the 1960s [139], [142], [143]. GAs usually represent solutions as a linear sequence of components, e.g. solution may be a sequence of bits, integers, doubles, or other fundamental types. (Sometimes mixtures of data types are used).

Mimicking the genetic operators of crossover, mutation and selection allows populations to be evolved to contain (hopefully) increasingly more satisfactory candidate solutions. In the crossover (usually) two 'parent' individuals exchange constituent elements for producing new individuals, and so each of these 'children' inherits elements from each parent. Mutation introduces diversity in individuals in the population. Typically, each element of an individual may be perturbed in some way with a small probability.

In a bit sequence individual, each bit may be flipped with a small probability. For integer sequence individuals an integer element may be replaced with a randomly chosen value or perhaps be incremented or decremented within some chosen range. Selection often implements a 'survival of the fittest regime' where fitter individuals have a greater chance of surviving to the next generation. (There are many ways selection can be implemented.) Figure 3.1 illustrates the basic idea of a GA.

In the figure below, the GA evolutionary process starts with a set of individuals (candidate solutions to the problem to be solved) commonly referred to as the *population.* Members of the population have their fitnesses evaluated. This allows a new population of solutions to be selected based on fitness values. 'Parents' are selected from the new population to be mated to produce 'offspring' using crossover (exchange genes of parents). Members of the new population of solutions are then mutated. The mutation occurs to maintain diversity within the population and prevent premature convergence. The population members then have their fitness evaluated. The cycle either repeats or stops when some criterion stopping criterion is met, e.g., a specified maximum number of cycles has been performed, or a solution has been found.

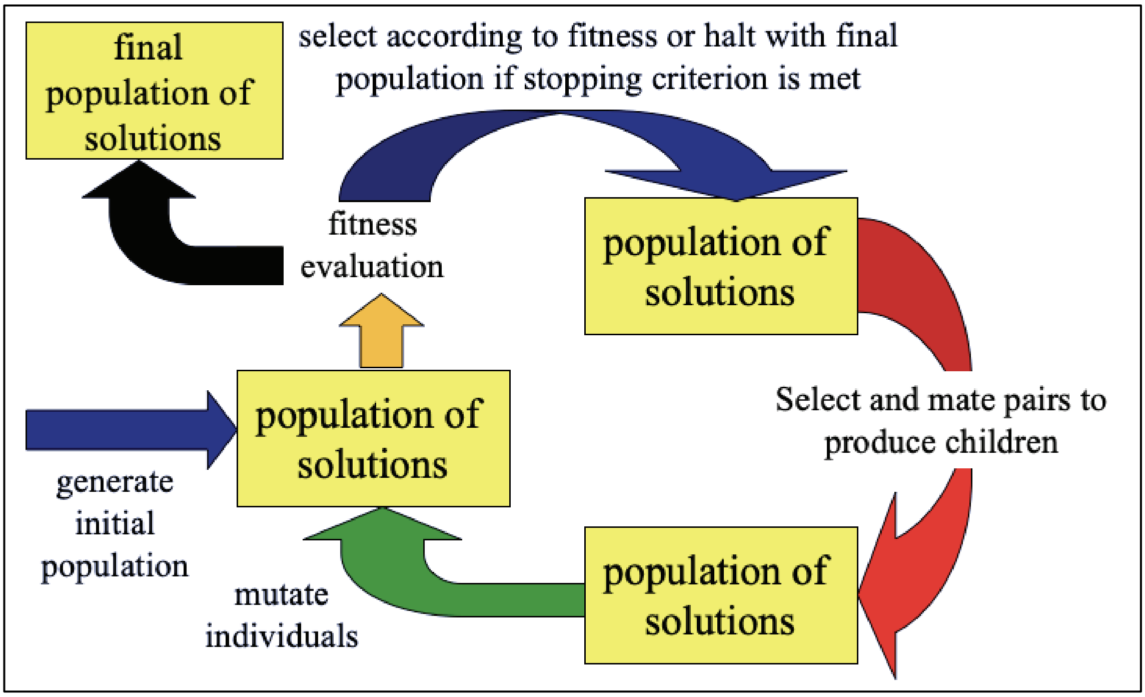

final population of solutions
select according to fitness or halt with final population if stopping criterion is met
fitness evaluation
population of solutions
population of solutions
Select and mate pairs to produce children
generate initial population
mutate individuals
population of solutions

Figure 3. 1 An Example of a GA, based on [144]

3.1.2 Genetic Programming

Genetic Programming (GP) is an extension of the GA, introduced by John Koza. The idea of GP was to represent a computer program as a tree [145], [146]. GP is an EC technique that can work with a wide range of input feature data types: integer, float, binary, string and so on. It can automatically solve problems without necessitating the user to be aware of the structure or form associated with the solution in advance [147]. In GP a population of computer programs is evolved generation by generation. GP implements a tree-based variant of mutation (randomly generated) and crossover (subtrees are swapped between parent trees). The figures below illustrate the mutation (Figure 3.2) and crossover operators (Figure 3.3) of GP. The necessary steps in the GP system programming are presented in Figure 3.4.

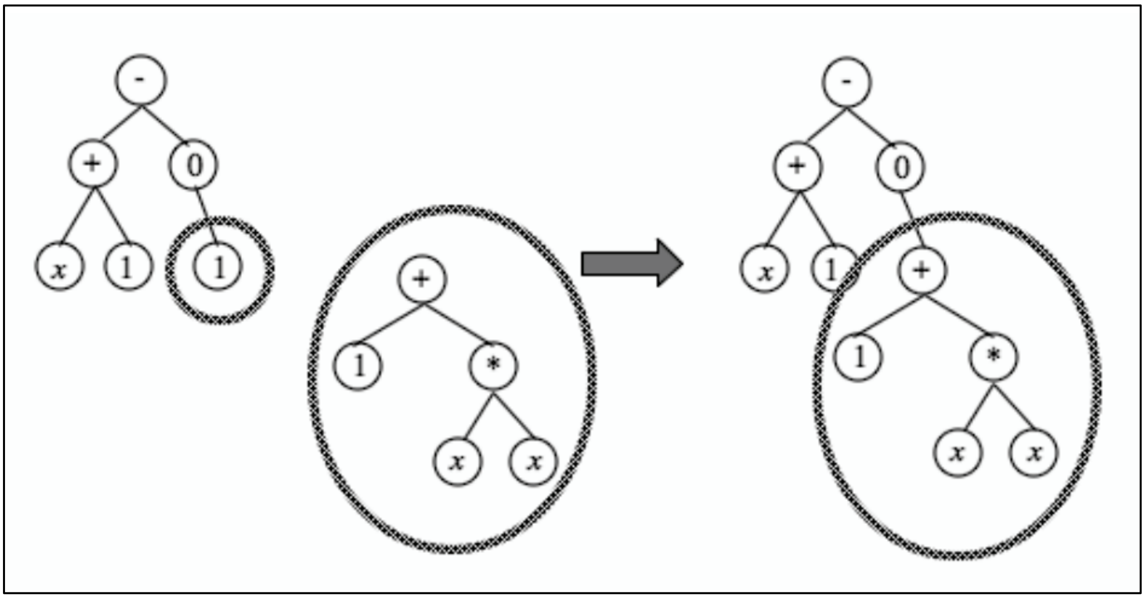


Figure 3. 2 An Example of a Mutation in GP [146]

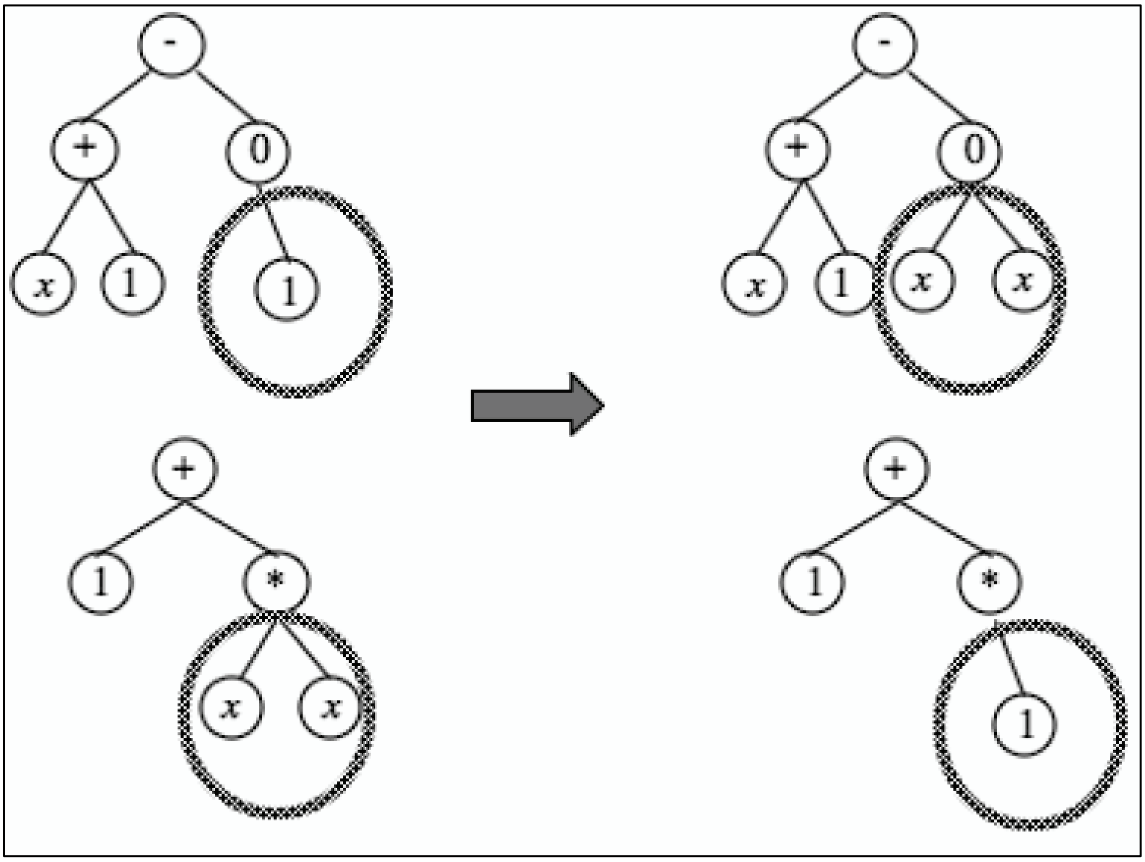


Figure 3. 3 An Example of a Crossover in GP [146]

```
Randomly create an initial population of programs from the available primitives.
repeat
    Execute each program and ascertain its fitness.
    Select one or two program(s) from the population with a probability based on fitness to participate in genetic operations.
    Create new individual program(s) by applying genetic operations with specified probabilities.
until an acceptable solution is found or some other stopping condition is met (e.g., a maximum number of generations is reached).
return the best-so-far individual.
```

Figure 3. 4 GP Algorithm [14]

3.1.3 Application of AI to Detect Attacks: Related Work

AI has been adopted by many researchers. The integration of IDS with AI started in 1986 [22]. Machine learning techniques can be used to develop highly efficient detectors for sleep deprivation or battery exhaustion attacks such as denial-of-service power attacks, malware attacks, spyware attacks and so forth. Narudin et al. reported that 99.97% (true-positive) of malware could be detected using the Bayes network and random forest classifiers on Malgenome datasets [25]. In trend used of machine learning techniques in IDS include ANNs; GAs; decision trees; SVMs; and fuzzy logic [26], [27].

GP has usually been employed for synthesising robust signature-based detectors for IDS. Garcia-Teodoro et al. reported deployment of GA within an anomaly-based IDS, contributing to a flexible and resilient system that did not have prior knowledge about the attacks [26]. The results of Sen et al. reveal GP can perform better than other AI techniques (e.g. SVM and Decision trees) as a lightweight method for detecting known flooding and route disruption attacks against the AODV protocol [23]. Her work inspired us to investigate whether the implementation of GP in our research can contribute to our main aim to deliver techniques that require fewer features, such as using only Android permissions.

A backdoor detection system using an ANN and GA was proposed by Salimi and Arastouie in 2011 [24]. They developed a novel approach to detect backdoor attacks on computers using two clusters such as system behaviour and network traffic as their features for ANN then the outputs from the process are used by the GA as inputs features to identify the backdoor attacks. In [128], the researcher using GP to generate new mobile malware from previously known malware and also do the detection using static analysis focusing on new malware or new variant of existing malware. This is different from us in a part of we used actual datasets that are used by researchers globally [23], [24], [26], [120], [128], [148]–[150].

3.1.4 Why Evolutionary Computation?

EC has achieved promising results for IDS in previous research [24], [26], [28] as discussed in Chapter 2. The modern mobile phone stores and processes highly valuable data and has become a highly attractive target for attack. A small number of researchers have sought to exploit AI in providing effective IDS, but the area is very much unexplored. It is motivating to investigate further how AI can be used to provide a robust framework and an application for IDS that target mobile phones.

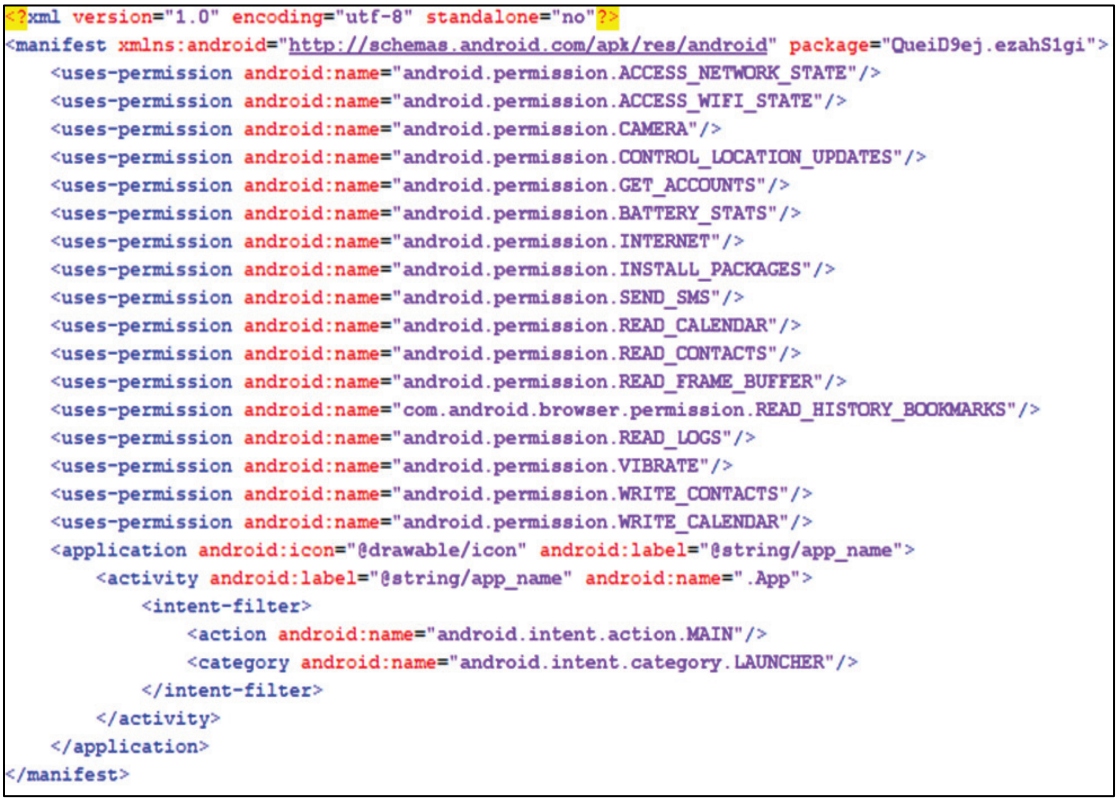
The Android framework has been chosen as the vehicle for experimentation since there is a very active development community, it is a widely used platform, and it is entirely open source. If research results prove that promising a natural community would be interested in further development.

GP has many benefits, but it has not yet (as of the submission of this Thesis) been explicitly used to detect Android mobile attacks by using solely only Android permissions (extracted from Manisfest.xml file). This one of the reasons to use GP as a method to analyse obtained datasets to distinguish attacks. We also inspired to investigate the capability of GP to do detection for mobile malware with fewer features (only permissions). In [128], they used the API calls and permissions to generate their coevaluation mobile malware and anti-malware using GP.

Furthermore, since mobiles are low resource platforms, we want to evolve efficient detectors where low power and low execution time, for example, are essential criteria. Accordingly, we aim to use GP in a multi-objective context. Sen et al. stated that an MOEA could allow the combination of multi-objective optimisation and evolutionary search [23]. A combination of GP and MOEA could produce detectors with excellent trade-offs between detection and resource usage.

3.2 Proposed Framework of Malware Detection

Researchers have employed various methods to detect malware in the Android application package (APK). APK is the file structure employed by the Android OS for installation and distribution of mobile apps (programs designed to run on a mobile device such as the phone, tablet or watch) and middleware (computer programs that deliver services to software programs beyond those available through the OS). The primary aim of our research is to provide ***techniques that require fewer features*** such as using only Android permissions (predicated on Manifest.xml file extraction from APK) to distinguish malware from non-malware APKs. Figure 3.5 is an example of a Manifest.xml file extracted from the APK, and the explanation of the Manifest.xml elements is shown in Table 3.1. Our proposed model in Figure 3.6 is the framework we have used to run the experiments starting from pre-processing the data. The data was obtained from other researchers, as mentioned in section 3.3. The details of the process are explained below.



```xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="QueiD9ej.ezahS1gi">
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.CONTROL_LOCATION_UPDATES"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.BATTERY_STATS"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.INSTALL_PACKAGES"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.READ_CALENDAR"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.READ_FRAME_BUFFER"/>
    <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
    <uses-permission android:name="android.permission.READ_LOGS"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
    <uses-permission android:name="android.permission.WRITE_CALENDAR"/>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name=".App">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Figure 3. 5 An Example of a Manifest.xml file extracted from APK

Table 3. 1 Manifest.xml Element Reference [151]

Element	Description
action	Adds an action to an intent filter. (An intent is a messaging object you can use to request an action from another app component)
activity	Declares an activity component.
application	The application's declaration.
category	Adds a category name to an intent filter.
intent-filter	Specifies the types of intents that an activity, service, or broadcast receiver can respond to.
manifest:	The root element of the AndroidManifest.xml file.
uses-permission	Defines the system permissions that the user must grant for the app to operate correctly.

Figure 3.6 shows the proposed malware detection framework. The details about each step within the experiment framework are as below.

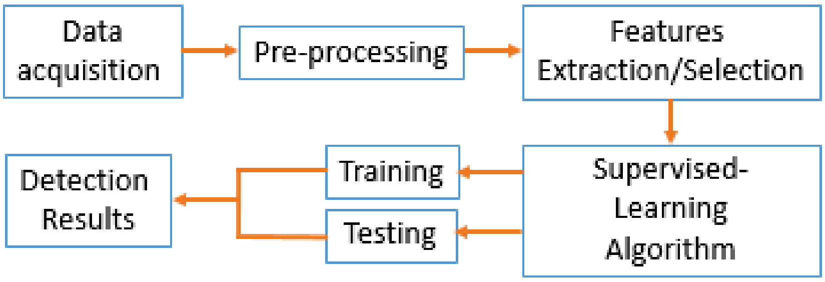


Figure 3. 6 The Framework of the Malware Detection

Data is acquired and pre-processed to remove elements that would disrupt the machine learning process. The resulting data then forms the input to the classifier synthesis process. The synthesis process uses feature extraction and selection prior to invoking a supervised learning approach. The detection performance of the developed classifier is recorded. The details of each step within the experimental framework process are given below.

3.2.1 Data Acquisition

To carry out meaningful work in this area, we need to have available case study datasets. Producing these is a significant task in its own right. In this research, we have sought assistance from the research community. Four international researchers have made datasets available to us, with responsible use and fair acknowledgement restrictions. The datasets and corresponding papers are:

- Drebin datasets from the paper “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket” [11];
- DroidKin datasets from the paper “DroidKin: Lightweight Detection of Android Apps Similarity” [12];
- Droid Analytics dataset from the paper “DroidAnalytics: A Signature-Based Analytic System to Collect, Extract, Analyze and Associate Android Malware” [116]; and
- Ransomware datasets from the paper “Ransomware Steals Your Phone. Formal Methods Rescue It” [120].

Unfortunately, we did not get access to Malgenome datasets from the paper "Dissecting Android Malware: Characterization and Evolution", [152], which is highly cited by many researchers because of their established work for systematic characterisation of existing Android malware. However, we obtained the DREBIN datasets all of whose malware files are derived from the Malgenome datasets.

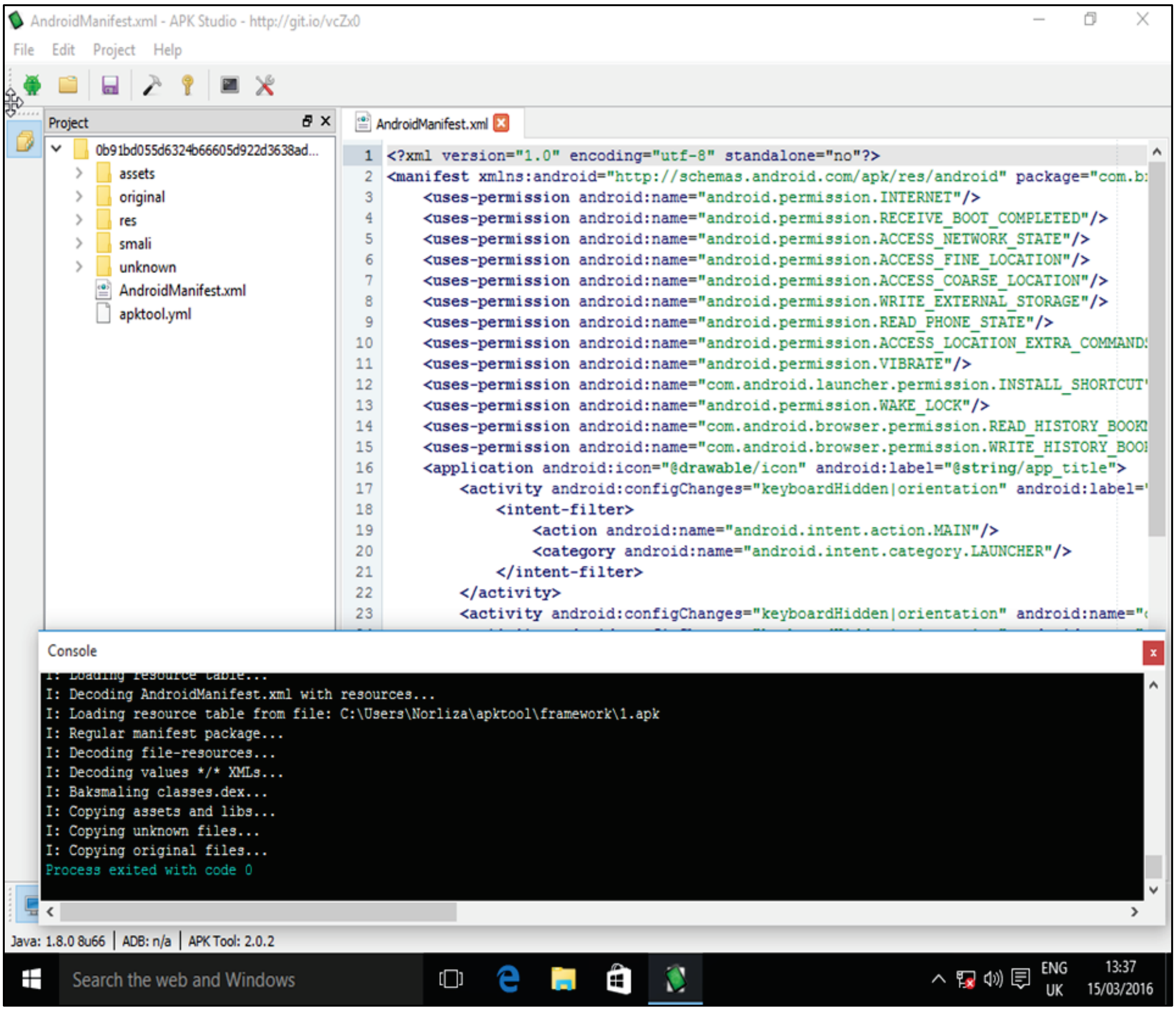
To select the main datasets to be used as the primary source, we performed some preliminary experiments. From the results (shown in Chapter 4), DREBIN is chosen in this research as the primary dataset. It is also an established dataset containing an extensive collection of malicious APK, and it is still being used worldwide by other researchers [23], [24], [26], [120], [128], [148]–[150]. Furthermore, DREBIN inherits most of the malware gathered by the Malgenome Project, a pioneer Android malware project and one whose datasets are very well known in the Android malware research community.

The other datasets will be explained in Chapter 6 (Droidkin datasets, Droid Analytics dataset and Ransomware datasets).

3.2.2 Data Pre-Processing

In this section, we focused on investigating the DREBIN datasets. As mentioned above, DREBIN provided our primary datasets in this research. The DREBIN samples were gathered between August 2010 and October 2012. The datasets contain 123,453 benign and 5,560 malware Android applications. In particular, the datasets hold 96,150 applications from the Google Play Store, 19,545 applications from various alternative Chinese Markets, 2,810 applications from substitute Russian Markets, and 13,106 samples from other resources, such as Android websites, security blogs and malware forums. Furthermore, the DREBIN contains all samples from the Android Malware Genome Project [152]. The file provided by DREBIN also contains the SHA256 hashed non-malware Android applications used in our experiments. We processed the DREBIN raw datasets to identify if the files in the datasets present problems, such as being empty (a file does not contain any information), duplicate files [150] and non-existing filenames (same as in provided listing). The datasets can therefore be 'cleaned' to remove such awkward instances.

The process to remove all unnecessary instances in the raw APK is done in a virtual machine. All data extraction was conducted in an Oracle Virtual Machine (VM) environment [153], and the version of the VM used in the experiments is VirtualBox 5.2.0 for Windows. The purpose of using VM hosts is to prevent the malware from spreading to the local machine and the network. In this experiment, we extract the Manifest.xml file and Dex code from the Android application data sets using APK Studio [154]. The APK Studio software is a cross-platform IDE for reverse-engineering (decompiling/editing) & recompiling of Android application binaries within a single user interface. The software features include a friendly interface, built with a code editor which supports syntax highlighting for Android SMALI (*. smali) code files. Figure 3.7 below is an example of one of the Manifest.xml file extracted using APK Studio. All datasets have been extracted before proceeding to the next procedure as mentioned in Section 3.2.3.



```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.b:
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT'
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKI
    <uses-permission android:name="com.android.browser.permission.WRITE_HISTORY_BOOI
    <application android:icon="@drawable/icon" android:label="@string/app_title">
        <activity android:configChanges="keyboardHidden|orientation" android:label='
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:configChanges="keyboardHidden|orientation" android:name="(
```

```
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Norliza\apktool\framework\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...
Process exited with code 0
```

Figure 3. 7 The Manifest.xml File Get from Decompile Process

3.2.3 Features Extraction and Selection

After finishing the data pre-processing process, twenty new datasets were obtained. It contains a combination of malicious malware applications and non-malware applications details (data retrieved from Manifest.xml files). The new database holds training and testing datasets based on the DREBIN experiments for detecting malware families (a group of malware with the same ability). The twenty malware families we use for evaluation, as mentioned in DREBIN [119], were the top Android malware families in the datasets and are shown in Table 3.2. Table 3.3 summarises the functionality of the five malware families with large samples in DREBIN. Other malware families are described in Chapter 6.

Table 3. 2 Malware Families Fraction [119]

Family	Samples	Family	Samples
FakeInstaller	919	Adrd	85
DroidKungfu	662	DroidDream	80
Plankton	620	LinuxLotoor	64
Opfake	608	GoldDream	64
GingerMaster	334	MobileTx	69
BaseBridge	324	FakeRun	56
Iconosys	145	SendPay	54
Kmin	142	Gapussin	53
FakeDoc	127	Imlog	38
Geinimi	87	SMSreg	36

Table 3. 3 Android Malware Families Used for the Evaluation

Family (Year)	Ability [155]
FakeInstaller (2010)	FakeInstaller sends SMS messages to numbers with the premium rate.
DroidKungfu (2011), also known as Kungfu and evolved into three variants.	DroidKungfu decrypts its exploits, deletes specific files on infected devices, runs specific apps on a phone or tablet, collects system-specific information, and avoids detection by the mobile anti-malware solutions available at that time. The DroidKungfu3 variant is capable of encrypting all malware information related to native binaries and conceals their actions as valid updates from Google.
Plankton or Plangton was also known as Tonclank (2011)	Plankton steals information and attempts to open a backdoor on Android devices, then collects the device ID and permissions and sends this information to a remote server.
Opfake is a variant of FakeInst (2011)	Opfake sends SMS messages to numbers with a premium rate.
GingerMaster (2011)	GingerMaster collects and uploads system-specific information to a remote server.

In the feature selection process, we evaluate and identify all of the suspicious permissions (SPs) that we intend to use in supervised learning using VirusTotal [156]. We carried out some experiments with a small sample to identify which permissions most often appear in Android malware packages, and then we identify the riskiest Android permissions with the VirusTotal database. The details of this process are given in Section 3.3.1. The listing of the SPs we used as features and the capability of the permissions to harm the mobile phone are explained in Table 3.4.

3.2.4 Training and Testing

The machine learning algorithm learns from the datasets provided. The training set is used to build the detection/classification model in the learning phase. The testing set serves to test the algorithm after the learning phase. The purpose of separating training and testing data is to avoid overfitting.

3.2.5 Offline Supervised Learning

Online (real-time) learning offers significant benefits, e.g. it allows the system to adapt as its environment changes. It may have particular strengths when the form of detection model is known, and continuous parameter tuning is the mechanism for adaptivity. However, when the form of the classifier is not known and it is a highly intensive computational task to discover it, then an off-line learning method is generally a good choice. Discovering the form of a classifier is one of the strengths of genetic programming but the approach can be computationally intensive. Accordingly we adopt an offline learning approach which allows us to avail ourselves of whatever computational resources are available offline to discover a good classifier [132].

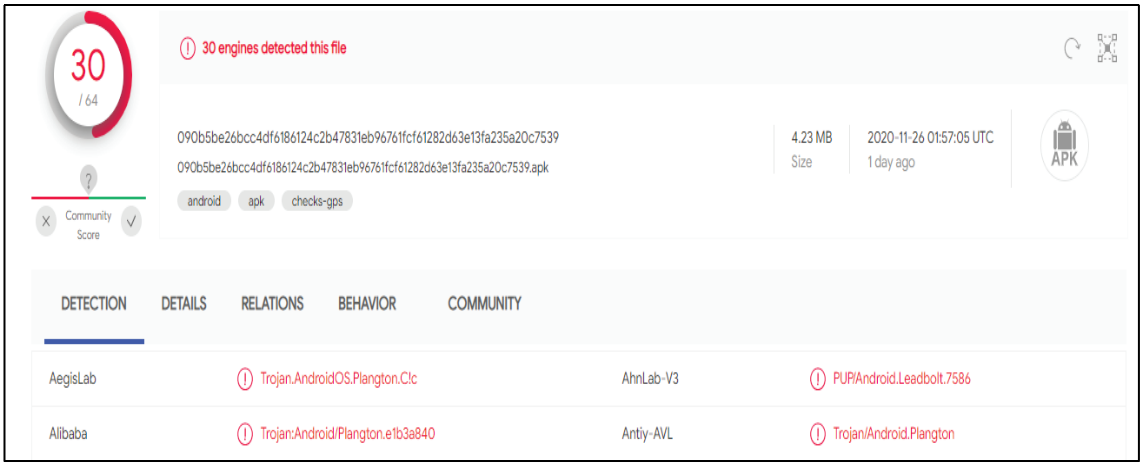
GP can automatically solve problems without necessitating the user to be aware of or require the structure or form associated with the solution in advance [147]. In GP the population of computer programs evolve generation by generation. Our dataset has ‘marked’ examples (i.e. we know whether the examples are malicious or non-malicious) and this allows us to readily adopt a supervised learning approach for our task. The implementation of GP is described in Section 3.3.

3.3 Evolving Detection Rules

In this section, the implementation of an EC technique to develop a detection program classifier for Android mobile phone is detailed.

3.3.1 Feature Selection

Features are characteristics of the system that form the essential data that evolved decision algorithms may use to reach their decisions. In this research, we focus on Android permissions as features. As a feature set, this is very basic and easily accessible from an Android app's Manifest.xml file. Table 3.4 summarises the suspicious permissions and what they enable in mobile phones. All our features are essentially string type values. To identify all suspicious permissions used in this research, we manually uploaded ninety random choosen malware samples from Drebin datasets to VirusTotal [156] website and copy the results. This process has been made in Virtual Machine as a precautionary step to avoid malware samples affecting machine used for the experiments or spreading malware samples in the network. The results we get from this process to identify the suspicious permissions. The analysis starts with listing all suspicious permissions that VirusTotal highlights (at the time we checked it on VirusTotal website) manually in a spreadsheet. Afterwards, we manually cross-checked all permissions to eliminate redundant permissions. Lastly, we listed the nineteen most frequent permissions in the ninety random malware samples from Drebin datasets. Figure 3.8 below shows the result of uploading one sample to VirusTotal website. Figure 3.9 shows the permissions indicated as dangerous by VirusTotal.



30 engines detected this file

090b5be26bcc4df6186124c2b47831eb96761fcf61282d63e13fa235a20c7539
090b5be26bcc4df6186124c2b47831eb96761fcf61282d63e13fa235a20c7539.apk

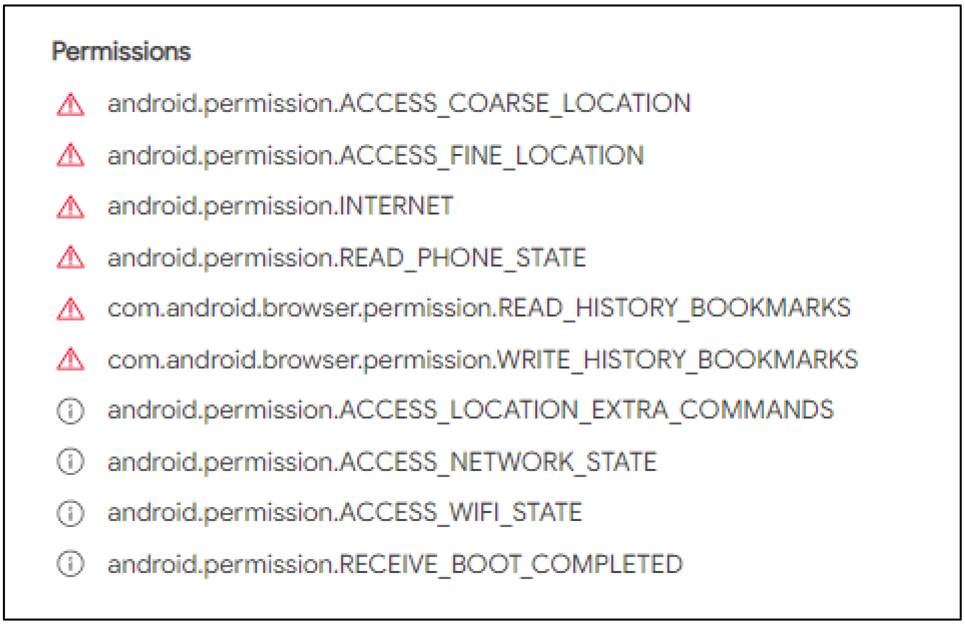
android apk checks-gps

4.23 MB
Size

2020-11-26 01:57:05 UTC
1 day ago

AegisLab	Trojan.AndroidOS.Plangton.C!c	AhnLab-V3	PUP/Android.Leadbolt.7586
Alibaba	Trojan:Android/Plangton.e1b3a840	Antiy-AVL	Trojan/Android.Plangton

Figure 3. 8 Result of One-Sample Uploaded to VirusTotal Website



Permissions

- ⚠ android.permission.ACCESS_COARSE_LOCATION
- ⚠ android.permission.ACCESS_FINE_LOCATION
- ⚠ android.permission.INTERNET
- ⚠ android.permission.READ_PHONE_STATE
- ⚠ com.android.browser.permission.READ_HISTORY_BOOKMARKS
- ⚠ com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
- ⓘ android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
- ⓘ android.permission.ACCESS_NETWORK_STATE
- ⓘ android.permission.ACCESS_WIFI_STATE
- ⓘ android.permission.RECEIVE_BOOT_COMPLETED

Figure 3. 9 Permissions Detect as Suspicious by VirusTotal website.

Table 3. 4 The Features Explanation [157]

Suspicious permissions (android. permission)	Actions
ACCESS_COARSE_LOCATION	Allows applications to access approximate location.
ACCESS_FINE_LOCATION	Allows applications to access approximate location.
INSTALL_SHORTCUT	Allows applications to install a shortcut in Launcher.
INTERNET	Allows applications to open network sockets and get full Internet access.
MODIFY_PHONE_STATE	Allows applications to modify the telephony state.
READ_CONTACT	Allows applications to read the user's contacts data.
READ_HISTORY_BOOKMARKS	Allows applications to read the Browser’s history and bookmarks
WRITE_HISTORY_BOOKMARKS	Allows applications to write the Browser’s history and bookmarks
READ_SMS	Allows applications to read SMS messages.
SEND_SMS	Allows applications to send SMS messages.
WRITE_SMS	Allows applications to write and edit SMS messages.

READ_PHONE_STATE	Allows applications to have read-only access to phone state.
VIBRATE	Allows applications to have access to the phone vibrator.
WRITE_APN_SETTINGS	Allows applications to write the apn settings.
WRITE_EXTERNAL_STORAGE	Allows applications to write the external storage.
BLUETOOTH	Allows applications to connect to paired Bluetooth devices.
DISABLE_KEYGUARD	Allow applications to disable the keyguard if it is not secure.
RECEIVE_BOOT_COMPLETED	Allows applications to obtain the ACTION_BOOT_COMPLETED that broadcast after the system finishes booting.
SET_WALLPAPER	Allows applications to set the wallpaper.
WAKE_LOCK	Allows applications to use the processor to avoid the phone from 'sleeping' mode or dimming the phone screen.

3.3.2 Application of Genetic Programming to Intrusion Detection in Mobile Phones

The experimental setup starts with data acquisition and proceeds to the data pre-processing process to set up the training and testing datasets. GP needs as functions, variables and a fitness function to be defined for the problem at hand. The list of variables utilised in the experiments (adapted from Android permissions) and being used as features is shown in Table 3.4. The functions used alongside the substantial GP parameters presented in Table 3.5.

Population size is the total number of individuals in a population in every generation. *Generations* identify after how many cycles the evolution process will stop. *Crossover probability* indicates how likely individuals nominated for breeding might swap elements. *Reproduction probability* demonstrates how likely this operator is to be applied to the individual selected. The *Tournament selection* is a strategy for selecting individuals for breeding. A set of *tournament size* individuals are selected, and their finesses evaluated. The individual with the highest fitness is deemed the tournament winner and selected for breeding. Tournament selection is a means of favouring fitter individuals in a population. It thus implements a variant of survival of the fittest. The ECJ 23 [158] toolkit is used for the GP implementation. Other parameters not itemised here are the default parameters of the toolkit.

The parameter settings below were developed after some preliminary training and testing with reduced datasets before we ran all experiments using the full dataset. The preliminary results show there is no significant changing of detection rate by using either loosely or strongly typed GP and by increasing the population size and the generations. The best detection rate and fastest duration to complete the experiments are achieved by using the parameters setting below:

Table 3. 5 GP Parameters Settings

Parameter	Value
GP typed	Loosely typed (does not enforce a specific type between the nodes)
GP format	Tree-based
Objective	Find a program to detect Malware using the information in the Android APK.
Non-Terminal Operators	Contains, AND NOT and OR
Terminal Operators	The feature sets in Table 3.4

Fitness Function	The Android APK dataset flagged as malicious or non-malicious.
	A weighted function of TP, FP, TN, FN were
	TP=True positive count
	TN=True negative count
	FP=False positive count
	FN=False negative count
	See below for details.

Standardised Fitness	Same as raw fitness
Parameters	Population Size = 1024 Termination when Generations = 50 Crossover Probability. = 0.9 Reproduction Probability = 0.1 Tournament Size = 7
Termination	Once an individual at fitness much better than 0.1 discover

The fitness function is a critical component of every evolutionary computational search. The search aims to find individual candidate solutions that maximise the fitness function. For classification and detection, there are several criteria we would wish to maximise, most notably the fraction of malware correctly classified and the fraction of non-malware correctly classified. There are trade-offs to be prepared amongst the two, and it is essentially a business decision as to what those trade-offs should be. A typical way of searching for candidate solutions in this trade-off space is to choose a fitness function that weights the two properties. However, it is not clear what the individual properties will be produced when particular weights are used. Accordingly, our searches will experiment with a variety of weights. The fitness function family used in the evaluation is given below:

$$\textbf{Fitness} = 1-\alpha*(TP/(TP+FN))-\beta*(TN/(TN+FP)) \qquad (1)$$

$$\alpha \text{ - (Range } 1 - 0.05)$$

$$\beta \text{ - (Range } 1 - 0.05)$$

TP (true positive) is the number of malicious applications correctly identified as malware. FN (false negative) is the number of malicious applications incorrectly identified as non-malware. TN (true negative) is the number of non-malicious applications correctly identified. FP (false positive) is the number of non-malicious applications incorrectly identified as malware. α and β are parameters used to give weight to the fitness either to emphasise the True Positive Rate or True Negative Rate. In practice, and to provide a means of normalisation, we impose the constraint $\alpha + \beta = 1$ in the experiments in this thesis.

CHAPTER 4

Performance Evaluation of Genetic Programming on Mobile Phones Intrusion Detection System

This chapter presents the results of using GP to synthesise malware detection and classification algorithms for Android mobile phone datasets. The motivation and contribution of the study are given in Section 4.1. Then, parameter settings are determined using preliminary experiments using a sample of datasets, as described in Section 4.2. The performance of programs evolved using GP is evaluated and discussed in Section 4.3. The performance is also compared with the results of previous research (Drebin) in Section 4.44. Section 4.5 examines the effects of different weights for elements of fitness function elements.

4.1 Introduction

4.1.1 Motivation

From Table 2.3 in Chapter 2, we can conclude that most of the researchers have been interested in investigating the performance of SVM to identify malicious and non-malicious applications in Android mobile phones datasets. There seems little reason to believe a priori that SVMs are optimal for such tasks, and there is plenty of scopes to apply other supervised machine learning approaches. GP has shown encouraging results when evolving intrusion detection programs for MANETs [83], and in [128] GP was shown capable of evolving both malware and anti-malware. Conceptually, GP also works in a very different way to SVMs. It is difficult to predict how the performance of the two approaches will compare. In this chapter, we will investigate the performance of GP evolved programs in detecting malware and non-malware applications in Android datasets.

In any search-based approach to solving a problem, a fitness function is used to guide the search (or, equivalently, a cost function). The analyst provides the search system with the function to be maximized. Where there are competing finesses, because we want performance on multiple axes, the fitness function often comprises a *weighted sum* of individual fitness components. There is generally little in the way of the convincing rationale for the specific choice of weights. In our case we wish to minimize false positive (FP) identifications, i.e. incorrectly identifying normal app as malicious, but also minimize the number of false-negative identifications, i.e. incorrectly identifying a malicious app as normal.

However, for malware detection, we are actually seeking to identify many different types of malware, not just one. The choice of weightings used may radically alter the performance of the search process when we attempt to evolve a detector for any one type of malware. ***There seems to be no research indicating how optimal choices of weightings vary across specific target malware types***. It is entirely possible that a "one-size fits all" approach to choosing weights will not prove effective. If so, this is important information for those seeking to use EC approaches for malware detection. We choose to investigate this aspect.

4.1.2 Contributions

The contributions in this chapter are:

- the production of empirical evidence to demonstrate that GP approaches are an effective method to identify malware and non-malware applications in Android mobile applications datasets.
- the production of empirical evidence to show that the efficacy of weight choices for GP's fitness function varies according to malware type targeted. Loosely, we demonstrate clearly what works best for one type of target malware does not work best for others. Thus, any specific choice of weights is limited in what it can achieve over the whole set of targeted malware types.

4.2 Experimental Investigation

In all experiments reported here, the datasets were obtained from the public access Android community website; all datasets can be used with an appropriate citation by the researchers. All experiments conducted in this chapter use the Drebin dataset as the primary source. In Chapter 3, we explained the procedures used to extract Drebin datasets and the data elements we need for our experiments.

As mentioned in Chapter 3, we use the Evolutionary Computation in Java (ECJ) [158] toolkit for our experiments. ECJ is a research EC system written in Java. It is highly flexible and configurable and contains basic implementations of many EAs: GP is just one.

Since Android permissions are recorded as strings within our database, we have developed a GP approach that manipulates Strings and incorporates logical connectives, allowing evolved expressions over the strings to form our malware detection predicates.

4.2.1 Preliminary Analysis

For the preliminary experiments, the method was discussed in Chapter 3. The purpose of preliminary experiments is to test the configurations and select the best parameters and features to use for our further experiments—the GP parameter settings used in this chapter and throughout chapter 6 are those given in Table 3.5.

4.2.1.1 Datasets

The training and testing samples are derived from the Drebin datasets, and we only used the small size of samples. The malware families were randomly selected to evaluate the parameter settings. Five hundred two non-malware samples and four hundred sixty-three malware samples were selected randomly from the Drebin datasets. The reason of random selection for both samples and malware families because it is a precise scientific procedure for the individual unit in a population will get an equal chance to be selected for inclusion in a sample [159]. The details of the malware sample dataset used in this experiment are shown in Table 4.1.

Table 4. 1 Malware Families Sample [44]

Family	Samples	Family	Samples	Family	Samples
Adrd	66	FakeTimer	12	Nandrobox	13
Basebridge	5	Geinimi	65	Nisev	5
DroidDream	5	GinMaster	25	Opfake	25
DroidKungfu	5	Iconosys	25	Plankton	65
FakeDoc	5	Imlog	25	Smforw	2
FakeInstaller	10	Kmin	25	Spitmo	11
FakeRun	63	LinuxLotoor	6		

4.2.1.2 Parameters

Population size and the number of *generations* allowed in a run are known to be important parameters for GP. In these experiments, we test these to determine which combination will produce the best results for detection. The other parameters did not change, and we used the standard configuration supplied by ECJ. Each combination of *population sizes* (1024, 2048, 3072, 4096) and *generation* (50, 100, 150, 200) was investigated *with* 20 runs for each experiment, as shown in Table 4.2.

Table 4. 2 Testing Population Size and Generation

Population sizes	Generations	Results
1024	50,100,150,200	There is no significant change to the results.
2048	50,100,150,200	There is no significant change to the results.
3072	50,100,150,200	There is no significant change to the results.
4096	50,100,150,200	There is no significant change to the results.

When running the experiments with vast datasets, we need to consider the time taken by the program to execute, as increased *population size* and the *generations* will also increase the execution time. From the results, we can conclude that using 1024 as *population size* and 50 as the *generation* in the experiment is a *plausible* approach; there is little reason to increase or decrease these parameter values. We also take into account the opinion in [160], where Langdon indicates most of Koza's original GP work used 50 as the *generation* for termination criterion.

4.2.1.3 Features Selection

In Chapter 3, nineteen features were discussed. For preliminary experiments, we started using nineteen features and gradually reduced the number of features one by one based on the lowest detection rate per family. In the end, fifteen features have been selected to be used in the next experiments. The purpose of this process was to identify the features that give the best performance malware detection with a high detection rate and low false-positive rate for the GP programs evolved. The fifteen features that have been selected are ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, INSTALL_SHORTCUT, INTERNET, MODIFY_PHONE_STATE, READ_CONTACT, READ_HISTORY_BOOKMARKS, WRITE_HISTORY_BOOKMARKS, READ_SMS, SEND_SMS, WRITE_SMS, READ_PHONE_STATE, VIBRATE, WRITE_APN_SETTINGS, and WRITE_EXTERNAL_STORAGE. The actions that can be taken by the permissions are discussed in Chapter 3.

4.2.1.4 Results and Discussion

Chapter 3 describes the procedure for the preliminary experiments. Our experiments target a single malware family dataset (training and testing datasets) per run. There are twenty malware families, as indicated in the table below, involved in preliminary experiments. The features sets are also reduced one by one with five features eliminated at the end of experiments. The results (the best individual of the ten runs) are presented below in Table 4.3 using *Population size = 1024*, *generation = 50,* and fourteen features. The fitness function is described in Section 3.3.2 and is used 0.5 for both α + β (Equation 1).

Table 4. 3 Results for Preliminary Experiments

Malware Family	TPR	TNR	FPR	FNR	Accuracy
Adrd	74.24%	95.62%	4.38%	25.76%	93.13%
Basebridge	60.0%	95.62%	4.38%	40.00%	95.27%
DroidDream	80.00%	59.56%	40.44%	20.00%	59.76%
DroidKungfu	100.00%	59.56%	40.44%	0.00%	59.96%
ExploitLinuxLotoor	83.33%	59.56%	40.44%	16.67%	59.84%
FakeDoc	100.00%	95.62%	4.38%	0.00%	95.66%
FakeInstaller	80.00%	95.62%	4.38%	20.00%	95.31%
FakeRun	100.00%	99.60%	0.40%	0.00%	99.64%
FakeTimer	100.00%	51.20%	48.80%	0.00%	52.33%
Geinimi	98.46%	99.60%	0.40%	1.54%	99.47%
GinMaster	100.00%	65.34%	34.66%	0.00%	69.31%
Iconosys	100.00%	95.62%	4.38%	0.00%	95.83%
Imlog	100.00%	65.34%	34.66%	0.00%	66.98%
Kmin	100.00%	95.62%	4.38%	0.00%	95.83%
Nandrobox	100.00%	95.62%	4.38%	0.00%	95.73%
Nisev	20.00%	99.60%	0.40%	80.00%	98.82%
Opfake	100.00%	95.62%	4.38%	0.00%	95.83%
Plankton	89.23%	99.60%	0.40%	10.77%	98.41%
Smforw	100.00%	95.62%	4.38%	0.00%	95.63%
Spitmo	100.00%	95.62%	4.38%	0.00%	95.71%

Twelve of the twenty malware families have 100% True Positives and 0% False negatives in the testing phase (preliminary experiments). The worst detection of malware occurs with the Nisev malware family; unbalanced datasets might cause a detection rate of only 20% as only five samples (Drebin provides only five samples in their datasets) were used in this experiment. Probably, the GP did not learn properly using the samples. However, as we can see the in Nisev datasets, the FPR is the lowest with only two samples detect as non-malware samples. Six malware families do not get above 90% for TNR, which also affects accuracy. Regarding the results shown above, we are still satisfied with the

GP performance as Geinimi and FakeRun average detection rate of more than 99%, which is the best detection rate.

4.2.1.5 Conclusion

The purpose of preliminary experiments is to identify the parameters and the features that will be used to evaluate the performance of the GP to evolve the best programs to distinguish malware from non-malware samples.

In the end, the values of generation and population size do not significantly affect the detection and false alarm rates. As the number of features was reduced from nineteen to fourteen, we realised the execution time to complete the full cycle of evolved GP program had also been reduced, and the performance had been improved. The five permissions excluded from the features are permissions that do not contribute meaningfully to the detection rate of evolved programs. We now use the remaining permissions in the experiments described below in section 4.2.

4.3 The Performance Evaluation of Genetic Programming

GP was used to evolve classifiers – programs which, when presented with an appropriate input data set indicate whether each of its constituent apps is a malware or normal APK. We have source data sets for non-malware and malware examples from datasets used by the community. Particularly, we used Drebin datasets [2]. Table 4.4 indicates the number of examples of non-malware samples used. The numbers of malware samples used are those given in Table 3.2.

Table 4. 4 Non-malware Samples based on Drebin Datasets

Family	Training	Testing	Family	Training	Testing
FakeInstaller	40,467	42,787	Adrd	42,771	42,742
DroidKungfu	40,365	42,127	DroidDream	40,305	42,945
Plankton	40,570	42,399	LinuxLotoor	40,043	42,729
Opfake	40,552	42,414	GoldDream	40,067	42,707
GingerMaster	40,201	42,419	MobileTx	40,028	42,734
BaseBridge	42,752	42,667	FakeRun	40,092	42,681
Iconosys	40,081	42,649	SendPay	40,294	42,975
Kmin	40,126	42,668	Gapussin	40,038	42,724
FakeDoc	40,078	42,645	Imlog	40,276	42,691
Geinimi	40,056	42,924	SMSreg	40,043	42,714

Each family is targeted in turn (i.e., we seek to evolve a detector for that specific family). We carry out twenty runs for each target malware family with the parameter settings in Table 3.5, and the fitness function is as in equation 1 in Chapter 3. Subsequently, we calculate the average to distinguish any bias that might occur during the experiments is running.

For each evolved program (detector), we calculate the accuracy (ACC), false-positive rate (FPR), and the false-negative rate (FNR). The formula for each of these measures is given below:

$$\textbf{ACC} = (TP/TN)/\ (TP+FP+FN+TN) \times 100 \qquad (2)$$

$$\textbf{FPR} = ((FP/\ (FP+TN)) \times 100 \qquad (3)$$

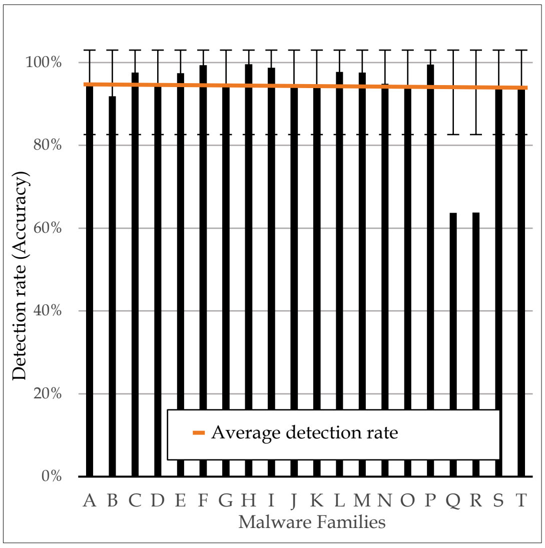
$$\textbf{FNR} = (FN\ /\ (FN + TP)) \times 100 \qquad (4)$$

ACC is the proportion of correct predictions for all detection.
TP rate is the proportion of correct malware detected.
TN rate is the proportion of correct for non-malware detected.
FP rate is the proportion of incorrect predictions for non-malware detected.
FN rate is the proportion of incorrect predictions for malware detected.

For the presentational purpose, it is convenient to denote each malware family by a single letter identifier. Table 4.5 gives the identifiers for each malware family of Figure 4.1.

Table 4. 5 Malware Families Reference

Family	Reference	Family	Reference	Family	Reference
FakeInstaller	A	Kmin	H	MobileTx	O
DroidKungfu	B	FakeDoc	I	FakeRun	P
Plankton	C	Geinimi	J	SendPay	Q
Opfake	D	Adrd	K	Gapussin	R
GinMaster	E	DroidDream	L	Imlog	S
Basebridge	F	LinuxLotoor	M	SMSreg	T
Iconosys	G	GoldDream	N		



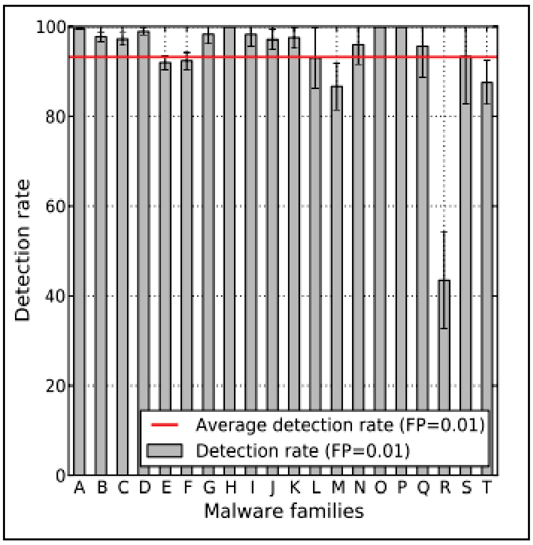


Figure 4. 1 The Performance of GP

Figure 4. 2 Drebin Detection Rate [119]

In these experiments, the fitness function uses a value of 0.5 for both α and β (refer to equation 1 in Chapter 3 for calculating the fitness), essentially viewing false positives and false negatives as equally important. The detection performance using GP for each family is shown in Figure 4.1 for the twenty malware families. The figure shows that GP can detect all families with an average accuracy of 93% and 7% average false positive rate. Correspondingly, almost all families have a detection rate of above 90%, and three families (Basebridge (F), Kmin (H), and FakeRun (P)) have a detection rate above 99%.

However, two families SendPay (Q) and Gappusin (R) give rise to the accuracy of only 64%. Figure 4.2 also shows that the detection rate for family R is only around 44% (but this is better than Drebin's rate). The results are summarised in Table 4.6:

Table 4. 6 SendPay and Gappusin Results Details

Family	TPR	TNR	FPR	FNR	ACC
SendPay	100%	64%	36%	0%	64%
Gappusin	100%	64%	36%	0%	64%

Both families achieve 100% TPR, but their FPR is very high at 36%. Although the TNR for both families is considerably worse at 64%, this happens due to the unbalanced data between both SendPay and Gappusin (refer to Table 3.2 in Chapter 3) and non-malware samples as shown in Table 4.4. In this experiment, we select only five samples for SendPay and Gappusin randomly for the training datasets and the balance of remain samples for both (SendPay and Gappusin) are used as testing dataset. Five samples for training datasets might not be enough for the GP to learn and evolve the best program to identify the malware.

4.4 Results Comparison

The results obtained in Section 4.2 and those from Drebin [119] are now compared. The evolved GP program detects F almost perfectly, which is better than Drebin, but Drebin detects O nearly perfectly. For R, our GP can detect 65% of the malware, which is better than the Drebin detection at 45%. Nevertheless, for Q Drebin gives better detection than our GP approach. The results for other families for both our evolved GP and Drebin reveal little difference. The average detection rate for Drebin is the same as our GP average detection rate at 93%. However, Drebin seems to detect almost perfectly for five families such A, D, H, P and Q. Though, and our detection is better because eighteen out of twenty families got 93% and above whilst with Drebin three families come below the average detection rate.

Our approach uses only permission as features and so may be regarded as a deliberately minimalist approach. The results above seem promising, but our approach may also be too limiting because some APK does not include permissions and also the combination of the permissions sometimes will generate false alarms.

It is well known in the EC community that the fitness function can matter a great deal. The above results were obtained, assuming an equal weighting for FPs and FNs. Our equal weighting of competing factors in the fitness/cost function may have been critical. In the next section, we investigate what are the optimal choices of a fitness function for the detection of each malware family and determine whether there is any commonality across families.

4.5 The Evaluation of the Genetic Programming Improvement Using Optimal Parameters

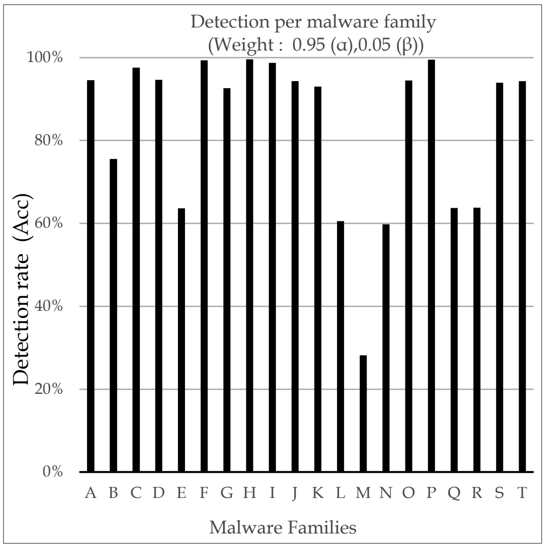
This section aims to evaluate the performance of the weight parameters to get the best performance from the evolved GP programs. In section 4.2, the work used a balanced fitness function (with α and β both having values of 0.5). In this section, we explore a variety of weights for α and β in the fitness function. Approximations to the optimal parameters for each α and β in the fitness function are identified at first. Then fair comparisons of these procedures under their optimal parameter settings are made to identify those that give rise to the best detection rates. All parameter values of the approach are as indicated in Chapter 3.

We explore the values of α and β in the range of 0.05 to 1.0 in steps of 0.05. We also impose a normalisation constraint that $\alpha + \beta = 1$ (refer to equation (1) in Chapter 3).

From first principles, we would expect differences in results to emerge from different choices. Specific choices for these parameters define the relative weights given to the two components. These two components are generally in opposition to each other, and so improving the performance of one aspect will often degrade performance in the other. There will come the point where simultaneously improving both becomes impossible.

As we range over 0.05 to 0.95, we would expect FPs and FNs rates to cross at some point. However, the choice of parameters that gives the most attractive trade-offs is not known. Our experiments should provide insight into this aspect also, and we can check such crossing points against measurements of the balance of the malicious versus non-malicious sample sizes.

All experiments were run twenty times. The accuracy average is calculated, and the graphs are illustrated below to summarise the results. The results indicate a mix of plausible and poor results. Below the poorest and the best performance results are shown in Figure 4.3 and Figure 4.4. The summary of other results is given in Appendix 1. As deduction of outcomes in Appendix 1, the results are shown when the fitness function weight change either on α or β the results also change significantly.



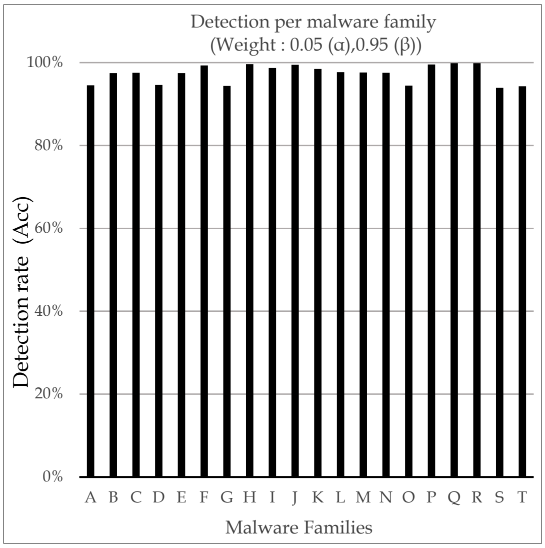


Figure 4. 3 The poorest GP performance **Figure 4. 4 The best GP performance**

In Figure 4.3, the results show a low detection rate for most of the malware families; the results are even worse than the result discussed in Section 4.3. Besides, the fitness function used for experiments, as shown in Figure 4.3 is 0.95 (α) and 0.05 (β). The conclusion can be made when α is at peak and β at the lowest the detection rate decreases for five families (B, E, L, M and N). Nevertheless, there is no significant change for other families, even for family Q and R, which have the lowest detection rate in Section 4.3. It is interesting to figure out how the modification to fitness function calculation affects the detection rate performance.

The best performance of evolved GP programs is shown in Figure 4.4, where the detection rate is almost perfect for all malware families. The graph also shows better results than those presented in [2] (where the Drebin Dataset was introduced). The detection decreases drastically from the results of the first experiment using balance 0.5 weight for α and β in the fitness function. In these experiments, β is set as 0.95, and α is fixed to 0.05 in the fitness function. This fitness function shown the best result for all malware families as shown in Figure 4.4.

The variety of results for the different combination of α and β are shown in Appendix 1. All results in these experiments prove the idea by using different weight in α and β in fitness function would improve or deteriorate the results. To investigate further of the results obtained, Table 4.7 is the matrix of the Pareto front for FPR and FNR for all range of α and β parameter used in the experiments. Table 4.7 shows the indicated values of β in the fitness function, whether the programs evolved using that fitness, function exhibited Pareto optimal performance.

In the table is the matrix of each parameters weight has been used in the calculation of the fitness function, and the details of each item are described subsequently.

Table 4. 7 Pareto Optimal (FPR and FNR) Achievement by Used Beta (β) Value.

Weights Families	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50	0.45	0.40	0.35	0.30	0.25	0.20	0.15	0.10	0.05
Adrd									×	×	×	×	×		×	×	×	×	
Basebridge	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
DroidDream				×															
DroidKungfu								×		×									
LinuxLotoor										×	×	×	×						
FakeDoc	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
FakeInstaller	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×		
FakeRun	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Gapussin	×	×	×	×	×	×	×	×	×	×	×								
Geinimi							×	×	×	×	×	×	×	×	×	×	×	×	×
GinMaster	×	×	×	×	×	×	×												
GoldDream							×	×	×	×	×	×	×	×	×	×	×	×	×
Iconosys	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	
Imlog	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	
Kmin	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
MobileTx	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	
Opfake	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	
Plankton	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
SendPay	×	×	×	×	×	×	×	×	×	×	×	×	×	×					
SMSreg	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	

X – The β weight that give Pareto optimal for Malware Families.

Table 4.7 above summarises the best performance of FPR and FNR; an "x" denotes that a program evolved using the corresponding β weight (to calculate the fitness) as a parameter achieved a Pareto optimal performance. As explained before, α will vary as β is changing. The results showed some of the families get the Pareto optimal results for FPR and FNR in each experiment using different weight β, and some of the families only get Pareto results for FPR and FNR when specific values are given to the weight β. The details of the eight malware families show impressive results clarified below, and other results are given in Appendix 2.

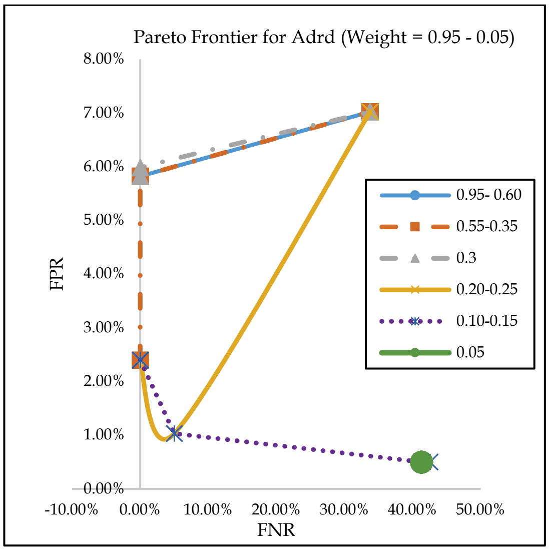


Figure 4. 5 Adrd Pareto Frontier

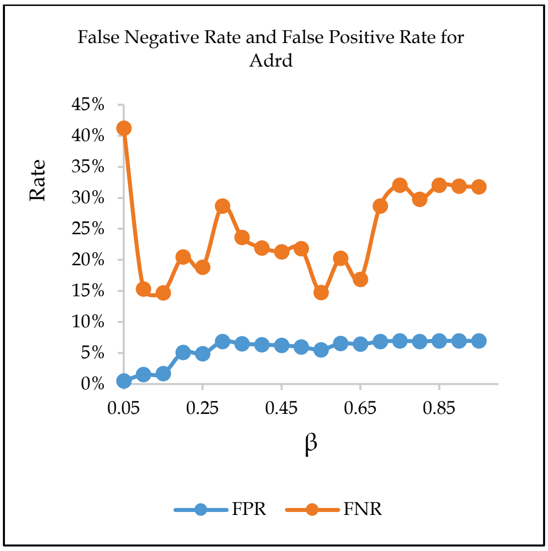


Figure 4. 6 Adrd FNR and FPR

The Pareto front for Adrd is shown in Figure 4.5, the range of β (weight for fitness) is grouping into six, and it indicates there is static FPR and FNR at 1% and 40% subsequently when 0.05 is given as the weight for β. The best Pareto frontier lies down between 0.10 - 0.25 and 0.55 – 0.35 weight for β. Figure 4.6 shows how FPR and FNR vary as with β. Interestingly the FPR also at the lowest but the FNR at the highest. The FNR drastically decrease from 40% to 15% as the β increase from 0.05 to 0.10, but afterwards, the FNR gradually increase as the β also rise. Figure 4.7, Figure 4.9, Figure 4.11, Figure 4.13, Figure 4.15, Figure 4.17, and Figure 4.19 can be summarised as both FPR and FNR are changing to be better or worse.

A conclusion can be drawn from Figure 4.8, Figure 4.10, Figure 4.12, Figure 4.14, Figure 4.16, Figure 4.18, and Figure 4.20. We can see that FPR and FNR meet at one point, which becomes a turning point either the results will be worse or better as the weight for the β increase.

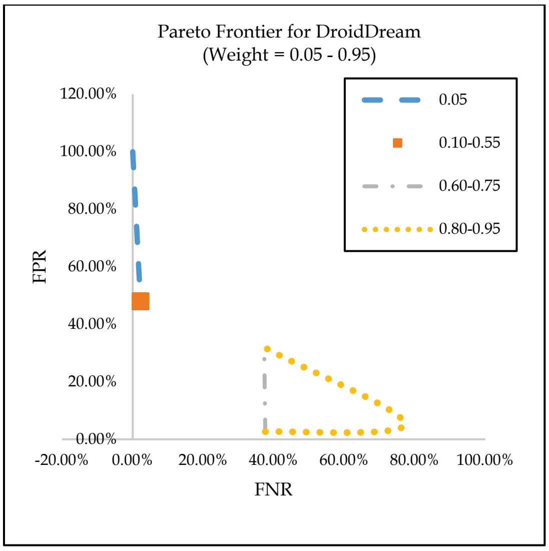


Figure 4. 7 DroidDream Pareto Frontier

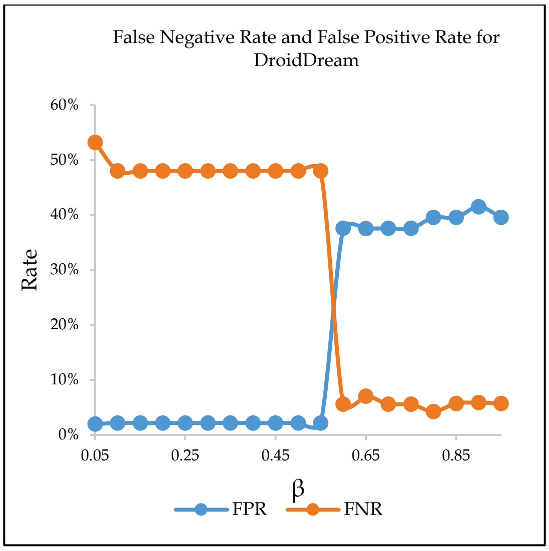


Figure 4. 8 DroidDream FNR and FPR

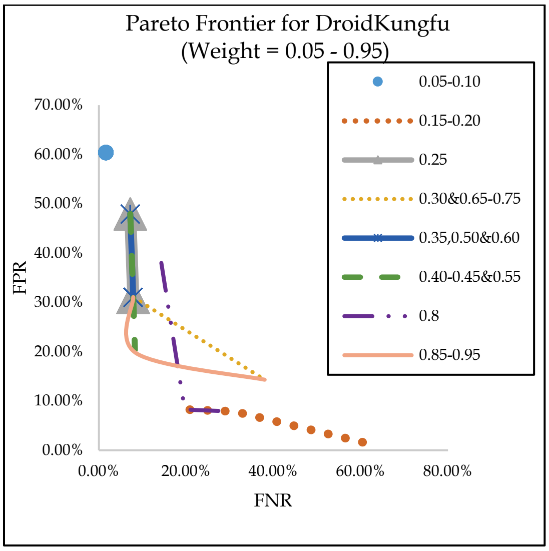


Figure 4. 9 DroidKungfu Pareto Frontier

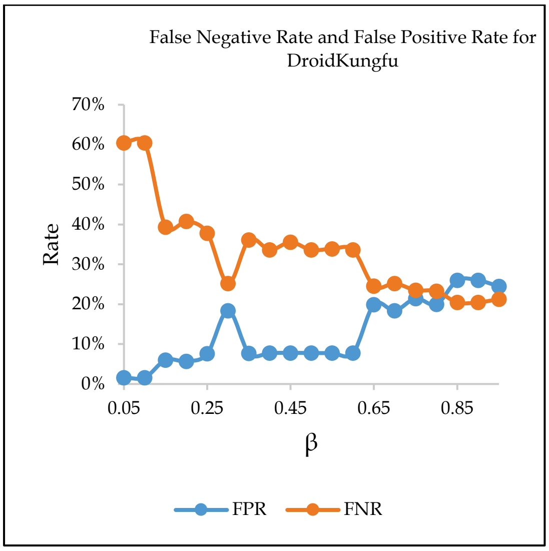


Figure 4. 10 DroidKungfu FNR and FPR

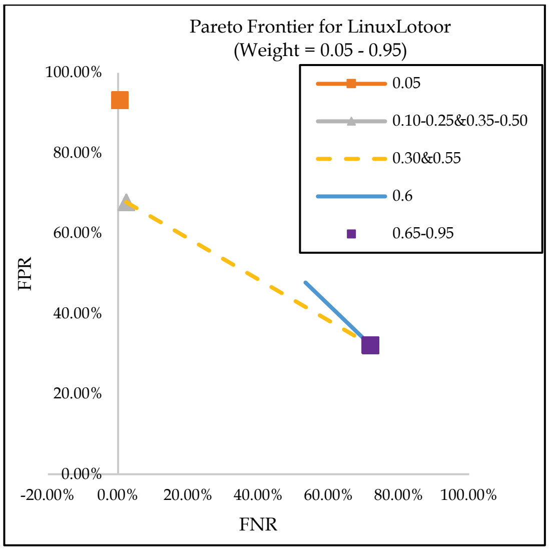


Figure 4. 11 LinuxLotoor Pareto Frontier

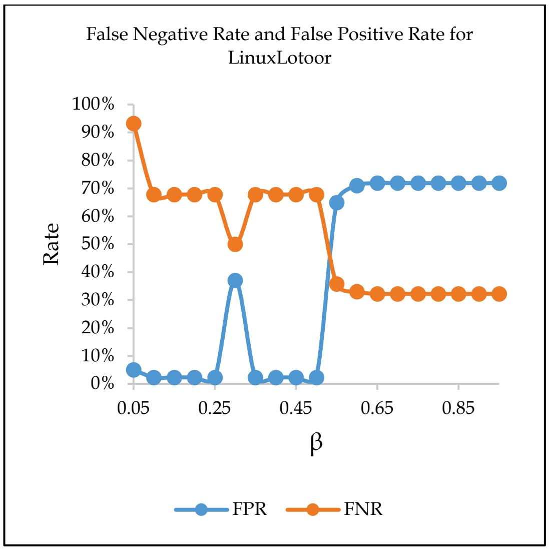


Figure 4. 12 LinuxLotoor FNR and FPR

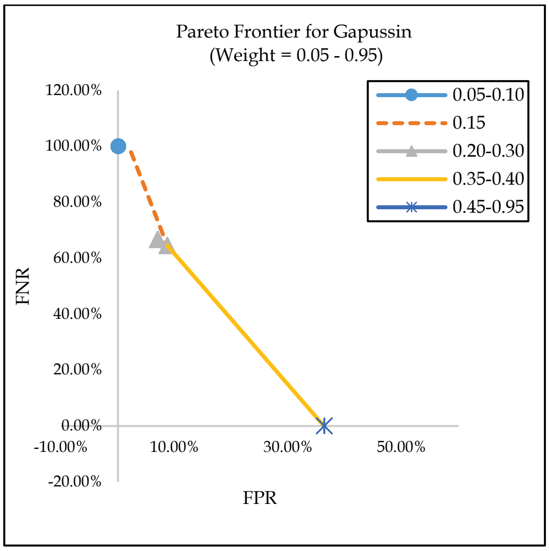


Figure 4. 13 Gapussin Pareto Frontier

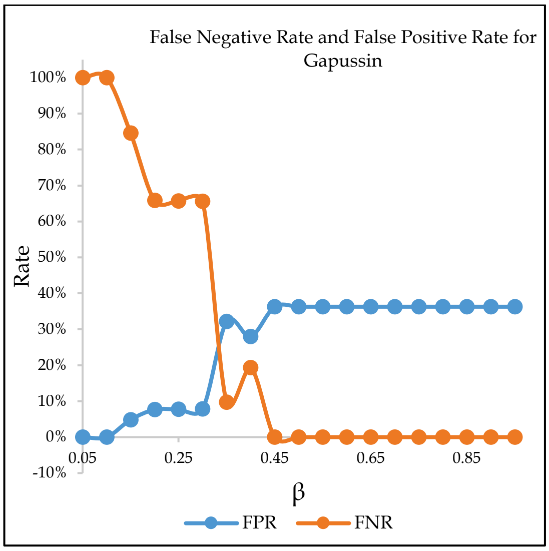
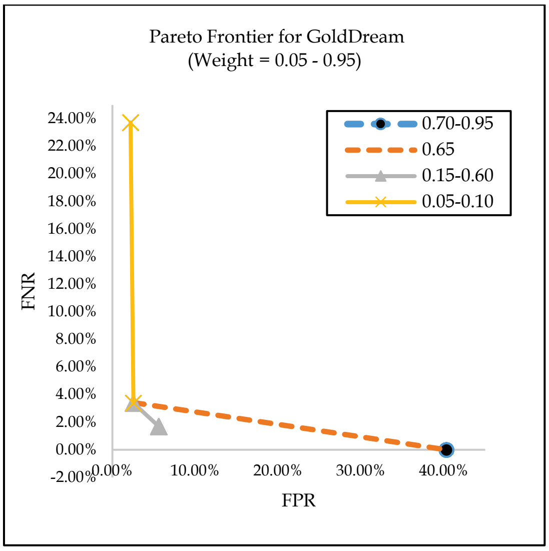


Figure 4. 14 Gapussin FNR and FPR



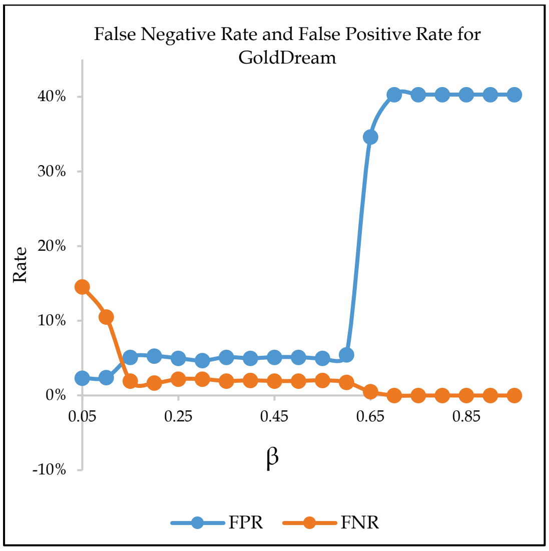
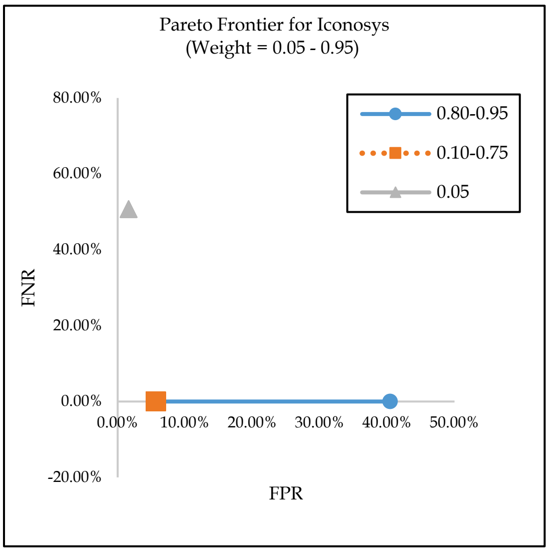


Figure 4. 15 GoldDream Pareto Frontier

Figure 4. 16 GoldDream FNR and FPR



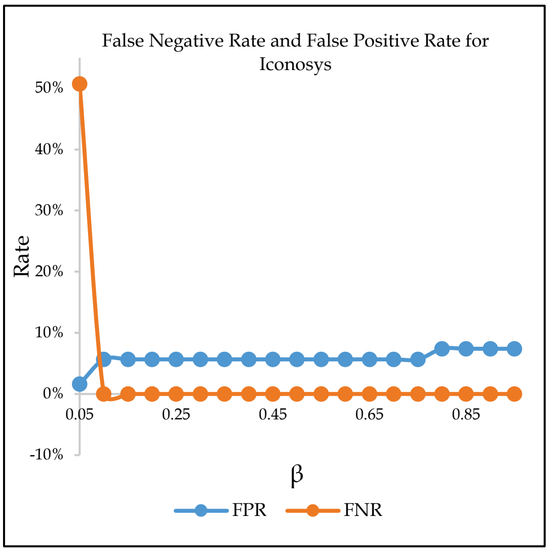
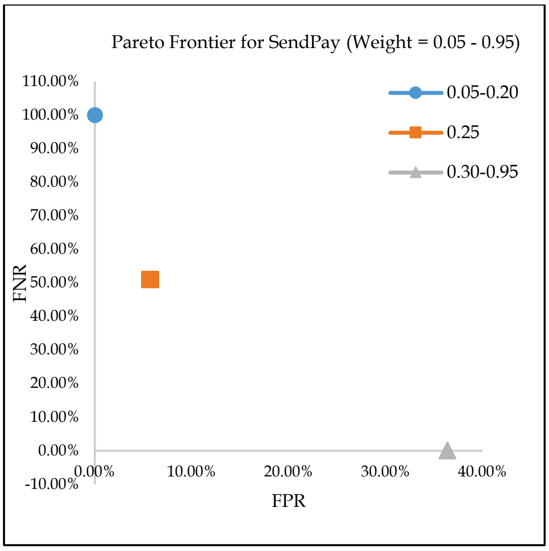


Figure 4. 17 Iconosys Pareto Frontier

Figure 4. 18 Iconosys FNR and FPR



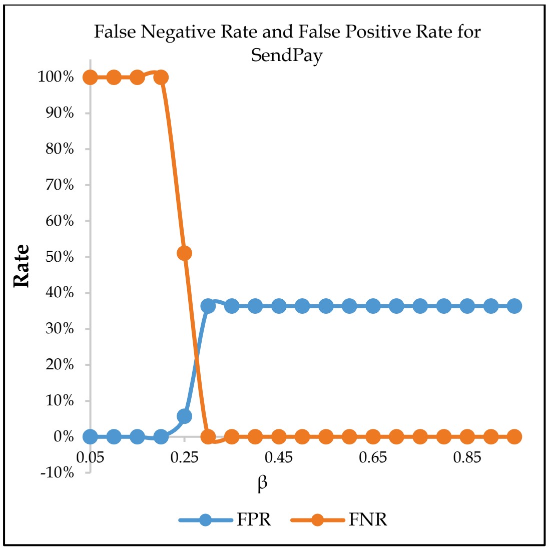


Figure 4. 19 SendPay Pareto Frontier

Figure 4. 20 SendPay FNR and FPR

4.6 Conclusion

When we use 0.5 as α and β values to calculate the fitness function, the best-evolved detectors achieved 93% accuracy. Our experimentation over the range 0.05-0.95 for α and β revealed that GP programs able to achieve a 98.8% average detection rate accuracy with 0.89% false-positive rate. As a conclusion (refer to Appendix 1) each time β value is increased, the detection rate accuracy declined. Nevertheless, when α increases, the detection accuracy improves. Even a slight change of the fitness function (i.e. via different weightings) can affect the detection rate.

In this chapter, an EC technique is presented to distinguish Android mobile attacks from the normal samples of APK. The evaluation results support the main hypothesis of this research - that GP will be able to evolve programs to distinguish malicious applications from non-malicious applications in mobile phone datasets. The results of the detection rate improve at the optimal detection as we used different weight for the fitness function calculation at α = 0.05 and β = 0.95. The experiments result also indicate a slight change of α and β in fitness function could give impact to the performance of the evolved GP program. Finally, we figure out the best weight for the fitness function to get optimal performances of GP to detect and classify the Android malware and its family.

This research was the first IDS for Android APK using GP for detection synthesis. The evolved GP program used only permissions collected from Manifest.xml file. A GP approach has been shown capable of evolving programs that can distinguish malware from non-malware Android applications.

MOEA has been shown to be capable of finding high performing solutions that make trade-offs between detector performance, execution time and power consumption.

CHAPTER 5

Improving Resource Efficiency

In this chapter, we consider how functional performance (detection rate) and non-functional properties (resource consumption and execution time) can be traded off using MOEC. We motivate our addressing this topic and summarise contributions. The multi-objective EC trade-offs are explained in Section 5.2. The power consumption of GP evolved programs is analysed in Section 5.3 along with the different trade-offs that can be made between classification accuracy, power consumption, and execution time of the evolved programs. Finally, the results are evaluated and discussed in Section 5.3.2.

5.1 Introduction

5.1.1 Motivation

According to KhorasaniZadeh et al., the ultimate challenge for malfeasance detectors is to decrease the resources needed to perform their function [132]. Mobile phones are well-known for being 'resource-hungry' devices [9], [115], [161]–[163]. The integration of multiple hardware parts accessible in modern smartphones increases their usability but decreases their battery lifespan to a couple of hours of functioning without recharge [137]. Energy-efficiency is the crucial constraint in mobile application design nowadays [163], [164] and app developers may not be entirely aware of how power-hungry their apps are [165].

Power usage can be measured or estimated in a variety of ways such as using dedicated hardware, cycle-accurate simulators and OS-level instrumentation, through to carefully calibrated software-based energy profilers that offer coarse-grained energy predictions to measure the power consumption of the device [166]. In [167], it was noted that power consumption varies across different mobile phone models.

The mobile phones research community is starting to investigate improving energy consumption in Android mobile phone applications [162], [164], [166], [168]–[171]. The Non-dominated Sorting Genetic Algorithm (NSGA-II) has been used for minimising energy consumption whilst maintaining desirable colour palette properties of a GUI screen [164]. Android developers can evaluate their applications using *vLens* to estimate the source code line-level energy consumption [169], or *eLens* to calculate the energy consumption per-instruction for the whole application [166].

Researchers have examined the trade-offs between power consumption and detection rate [172]–[174] in mobile phones. The energy consumption and performance of mobile antivirus (AV) software (Sophos, AVG, NQ, Avast, Dr Web and Norton) in Android has been investigated [172] at a low-level, concluding that AV software is often inefficient. Researchers have also investigated the security versus energy trade-offs along two axes: attack surface and malware scanning frequency, for both code and data-based rootkit detectors [19]. (We note in passing that details of the power consumption waveform itself can be used to detect malware, as proposed in [173]. Such use is interesting for IDS, but it is not the focus of the research reported here.).

The faster a detection is made, the less is the damage to mobile phones. In addition, a quicker detection program means that resources are available for other uses. Therefore, the execution time of a check is also of some importance. Therefore, in this chapter, we propose to investigate the use of the MOEA to explore the potential trade-offs between functional and non-functional performance measures.

5.1.2 Contribution

The contributions in this chapter are:

- Establishment of empirical evidence to demonstrate how optimisation can be used to explore trade-offs between functional properties (detection rate) and non-functional properties such as execution time and power consumption.
- Establishment of empirical evidence of how SPEA2 can give the best trade-offs for three objectives (detection rate, power consumption and execution time).

5.2 Multi-Objective Evolutionary Computation

The majority of real-world engineering problems have multiple objectives [141], [175]. A multi-objective optimisation finds values for multiple objective functions (typically in conflict with each other) acceptable to a designer (decision maker) [176], [177]. Optimisation problems with conflicting objectives are often addressed by aggregating the objectives into a single scalar function and solving the resulting single-objective optimisation problem [178]. Optimising a weighted sum of the individual objectives is commonly used. However, designers may have reservations about such an approach, e.g., the specific values of weights may be critical, and choices are often made in a somewhat ad hoc manner. However, there are principled alternatives. Multi-objective optimisation (MOO) (also known as multiperformance, multicriteria, vector, or Pareto optimisation) is defined as finding a vector of decision variables that satisfy constraints and optimises a vector function whose elements represent the objective functions [176], [179]. In a multi-objective search, either one solution 'dominates' another or neither dominates [178]. In contrast to single-objective optimisation, a solution to this problem is not a single point, but a group of points known as the Pareto-optimal set or 'front' [180].

In Figure 5.1 below, we can see a set of points labelled with '1' that make up the primary Pareto front. For each point labelled '1', there is no other point that is simultaneously better (higher performing) on criteria f1 and f2, where we seek to minimise f1 and f2. If 'dominance' means better on all (here both) criteria, none of these points is dominated.

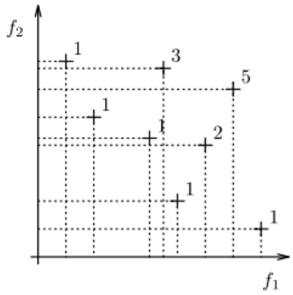
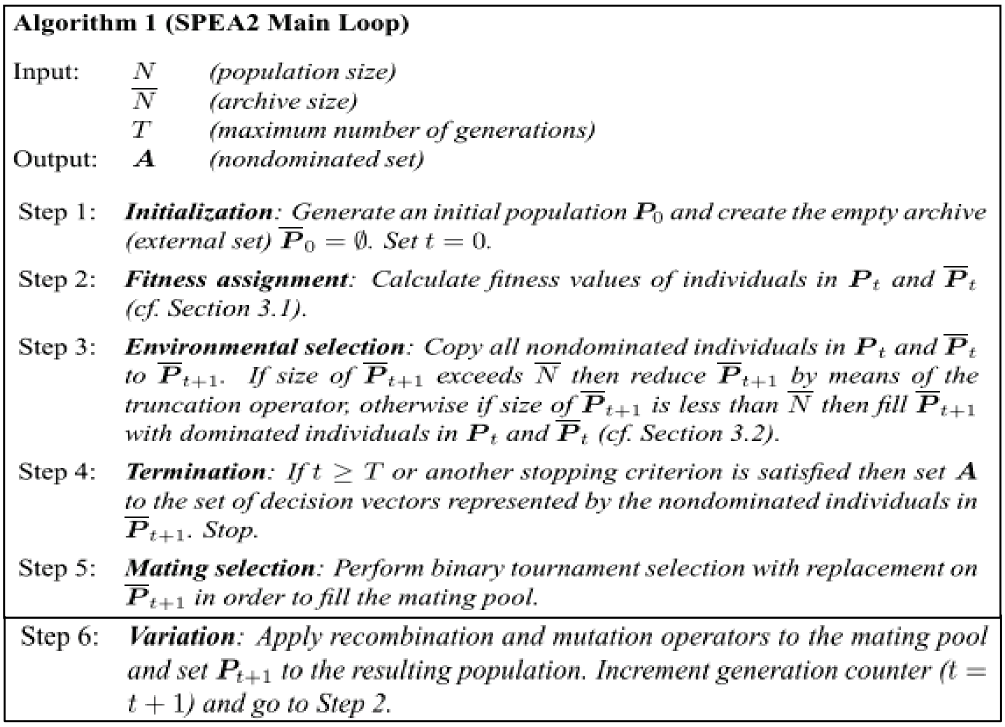


Figure 5. 1 Example of Multi-Objective Ranking using Pareto-optimal Solutions [180]

The multi-objective optimisation problem (MOOP) solving in EC has focused on two approaches: weight-based and Pareto-based [181]. The weight-based technique uses a single fitness function derived as a weighted sum of single property fitness functions. However, this approach cannot find Pareto-optimal solutions in the non-convex portion of the Pareto-optimal front [175], [181]. The Pareto-based technique offers a set of solutions acknowledged as Pareto-optimal solutions. In this thesis, we use the Strength Pareto Evolutionary Algorithm 2 (SPEA2) MOEA algorithm to find such sets of solutions.

5.2.1 Strength Pareto Evolutionary Algorithm 2 (SPEA2)

The Strength Pareto Evolutionary Algorithm 2 (SPEA2) is a successor to Strength Pareto Evolutionary Algorithm [181]–[183] (SPEA) introduced by Zitzler et al. in 2001 [184]. The steps of the SPEA2 algorithm are shown in Figure 5.2.



Algorithm 1 (SPEA2 Main Loop)

Input: N *(population size)*
$\overline{N}$ *(archive size)*
T *(maximum number of generations)*
Output: $\boldsymbol{A}$ *(nondominated set)*

Step 1: ***Initialization**: Generate an initial population $\boldsymbol{P}_0$ and create the empty archive (external set) $\overline{\boldsymbol{P}}_0 = \emptyset$. Set $t = 0$.*

Step 2: ***Fitness assignment**: Calculate fitness values of individuals in $\boldsymbol{P}_t$ and $\overline{\boldsymbol{P}}_t$ (cf. Section 3.1).*

Step 3: ***Environmental selection**: Copy all nondominated individuals in $\boldsymbol{P}_t$ and $\overline{\boldsymbol{P}}_t$ to $\overline{\boldsymbol{P}}_{t+1}$. If size of $\overline{\boldsymbol{P}}_{t+1}$ exceeds $\overline{N}$ then reduce $\overline{\boldsymbol{P}}_{t+1}$ by means of the truncation operator, otherwise if size of $\overline{\boldsymbol{P}}_{t+1}$ is less than $\overline{N}$ then fill $\overline{\boldsymbol{P}}_{t+1}$ with dominated individuals in $\boldsymbol{P}_t$ and $\overline{\boldsymbol{P}}_t$ (cf. Section 3.2).*

Step 4: ***Termination**: If $t \geq T$ or another stopping criterion is satisfied then set $\boldsymbol{A}$ to the set of decision vectors represented by the nondominated individuals in $\overline{\boldsymbol{P}}_{t+1}$. Stop.*

Step 5: ***Mating selection**: Perform binary tournament selection with replacement on $\overline{\boldsymbol{P}}_{t+1}$ in order to fill the mating pool.*

Step 6: ***Variation**: Apply recombination and mutation operators to the mating pool and set $\boldsymbol{P}_{t+1}$ to the resulting population. Increment generation counter ($t = t + 1$) and go to Step 2.*

Figure 5. 2 The SPEA2 Algorithm [184]

The SPEA2 archive size is fixed (when non-dominated individuals are less than predefined archive size it will be filled with dominated individuals). Only participants of the archive participate in the mating selection process. In SPEA2 the fitness assignment is different from SPEA (defined as a fine-grained fitness assignment strategy which integrates density D (i) and raw fitness R (i)) to avoid the individuals dominated by the same archive members that have identical fitness value.

The equation 5.3 is the fitness of an individual i in SPEA2. The equation 5.3 is defined by two components: the raw fitness R(i) (equation 5.1) and the density D(i) (equation 5.2). The raw fitness (R) of an individual i is determined by the strengths (S) of its dominators in both the archive and the population. The fitness is aimed to be minimised. The density (D) is calculated by the adaption from the k-th nearest neighbour element $k\ = \sqrt{N + \overline{N}}$ and distance as (σ_i^k) to the individual.

$$R(i) = \sum_{j\in P_t + \overline{P_{t,j>i}}} S(\text{j}) \qquad (5.1)$$

$$D\ (i) = \frac{1}{\sigma_i^k + 2} \qquad (5.2)$$

The individual fitness *F (i)*:

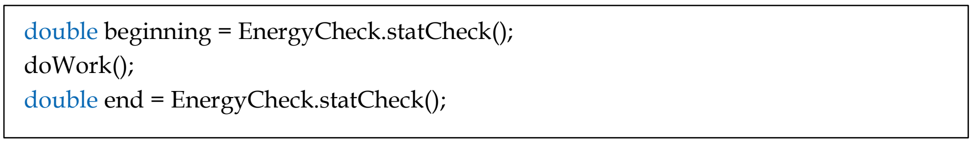
$$F\ (i) = R\ (i) + D\ (i) \qquad (5.3)$$

SPEA2 is known to reduce *bloat* in GP evolved programs [34] and has out-performed NSGAII in high-dimensional objective problems [35]. We are motivated to investigate SPEA2 as it is successfully used in the research of [23], [176] where three objectives are adopted: detection rate, false-positive rate, and energy consumption of the program. In our study, we are also using three objectives, malware detection rate and two non-functional properties (power consumption and execution time of the evolved program).

5.3 Implementation

In this section, we implement the SPEA2, one of the established multi-objective evolutionary algorithms (MOEAs) used by researchers [23], [136], [185], [186]. We are evaluating the optimising of non-functional and functional properties of the GP program using SPEA2. In the previous chapter 4, programs evolved using ECJ are evaluated. An implementation of SPEA2, which is an extension to ECJ [158], is used in this research. The power consumption and execution time of the GP programs evolved to detect twenty malware families (each malware family was targeted individually) are analysed in this section.

In our implementation framework, we integrate jRAPL [187]with our evolved programs. The jRAPL tool is capable of calculating the power consumption directly in the evolved GP program. jRAPL is a framework for profiling a Java program executing on CPUs with Running Average Power Limit (RAPL) support with the capability to control, monitor, and receive notification of energy and power consumption from different hardware stages, such as DRAM and CPU [187]. The tool is relatively easy to use. An example code block is given below.



```
double beginning = EnergyCheck.statCheck();
doWork();
double end = EnergyCheck.statCheck();
```

Figure 5. 3 Java Code to Calculate Power Consumption [187]

The jRAPL tool runs under the Linux operating system with sudo permission (superuser account). This is because the tool needs to access model-specific registers (MSR) in the Linux kernel to calculate power consumption for DRAM, CPU and Java packages separately. jRAPL can only calculate the energy for two sockets CPUs; consequently, all our experiments are carried out in a two sockets CPU machine. Subsequently, to estimate the precise actual execution time, we used Java's System function, the nanoTime() method [188]. We use the jRAPL tool (to calculate the power consumption) and Java's nanoTime() method (to estimate the execution time) in the experiments discussed in the next section.

5.3.1 Analysis of Power Consumption and Execution Time of Evolved Programs

In this section, our purpose is to study the power consumption and the execution time for GP evolved programs. The configuration and parameters of these experiments are the same as in Table 3.5, and the fitness function are same as in equations 1 in Chapter 3. GP programs are evolved for each malware family individually. The best individual programs for each malware families were executed ten times. Their power consumption and execution times are shown in Figure 5.4.

Figure 5.4 presents the graph of time (execution time) versus power (power consumption) for the twenty malware families. For FakeDoc, DroidKungfu, and Opfake the execution time and power consumption is reduced continuously for all ten runs. For other malware families, the evolved program's execution time and power consumption are continuously increased. A phenomenon in GP called bloat might cause these results, where the code bloating consumes increasing resources, and ultimately the search grinds to a halt because all available resource has been used [189]. Furthermore, such increasing resource consumption is not usually associated with an increase in functional performance. The SMSreg results show clear trade-offs between execution time and power consumption. For conclusion, we can identify that for most malware families, the execution time for the evolved program increases in tandem with power consumption.

In these experiments, we demonstrate that different trade-offs can be made between execution time and power consumption in the GP programs and the results encourage us to identify whether there are adequate trade-offs between these objectives. Therefore, in the next section, we use a MOEC algorithm (SPEA2) to investigate possible trade-offs among three objectives detection rate (DR), execution time, and power consumption. The SPEA2 approach has also been proven to reduce *bloat* in GP evolved program [23], [190].

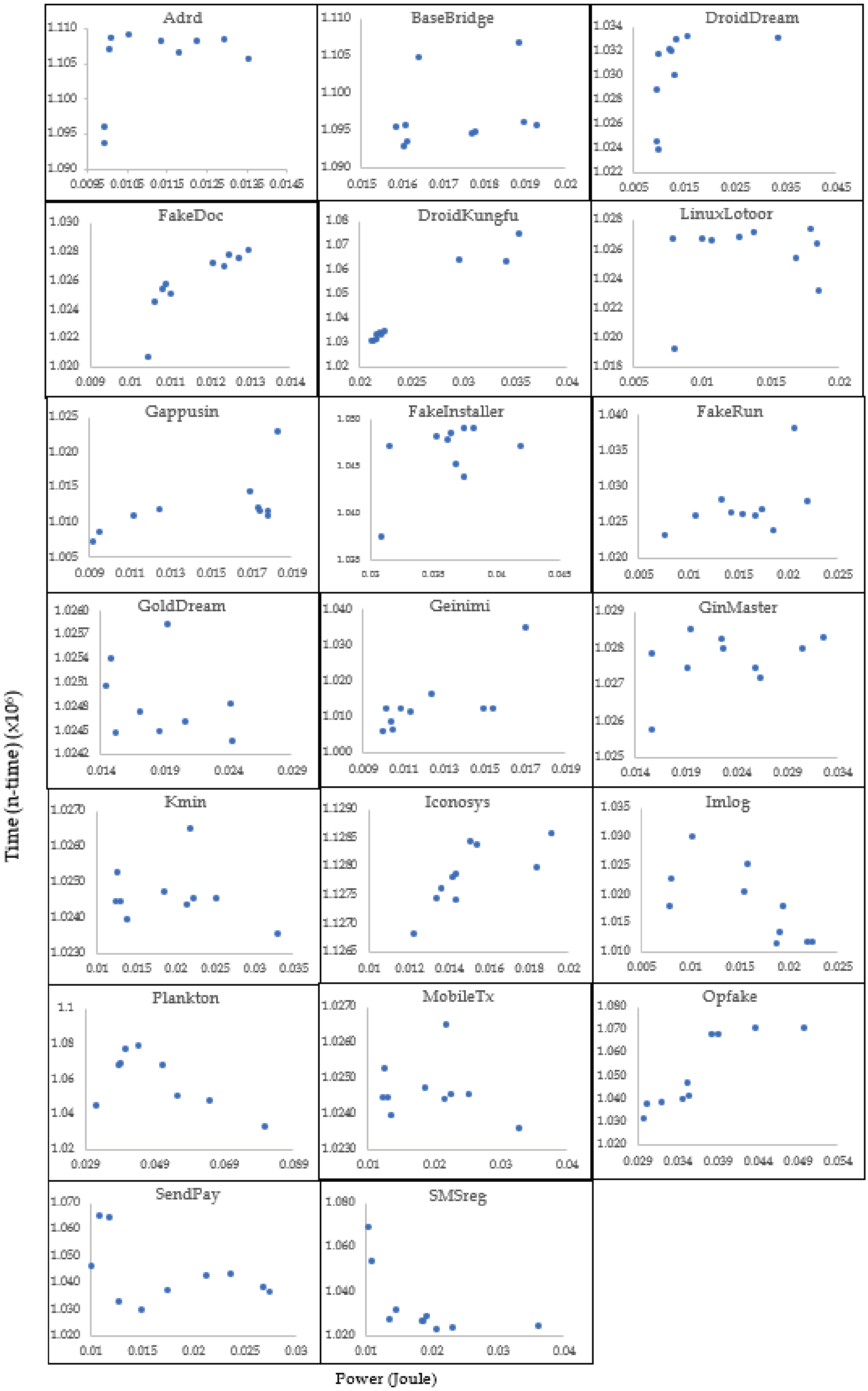


Figure 5. 4 Evolved Program for Execution Time and Power Consumption

5.3.2 Discovering Trade-offs in the Intrusion Detection Programs

In this section, we investigate trade-offs between the detection rate (DR), power consumption and execution time of the program. The following three objectives will be minimised concurrently.

$$f_1 = 1\text{-}(\textit{no. of attacks detected /no. of attacks}) \qquad (5.4)$$

$$f_2 = \textit{power consumption} \qquad (5.5)$$

$$f_3 = \textit{time-consumed} \qquad (5.6)$$

The conceptual diagram of the experimental framework is shown in Figure 5.5.

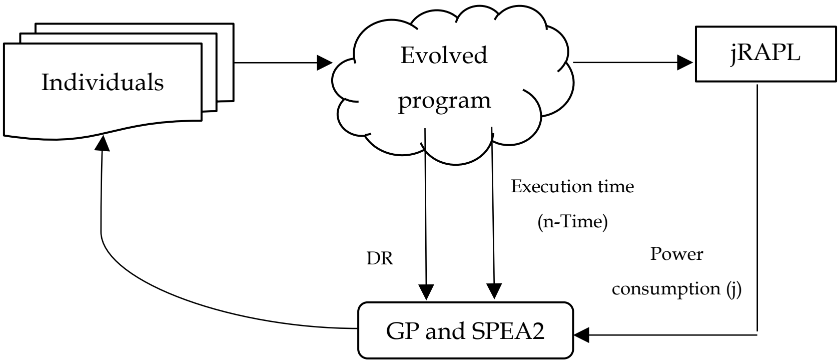


Figure 5. 5 Simplified Concept of Experiments

Figure 5.5 illustrates the process of the experiments reported in this section. The power consumption of the evolved program is calculated by jRAPL, and GP and SPEA2 used the results as the first objective. The second objective is derived from the execution time of the evolved program using nanoTime (measured elapsed time). The third objective is derived from the detection rate. SPEA2 is used to optimise the objectives. We have used the same experimental procedure as in Chapter 4; a program is evolved separately for each of the twenty malware families. The parameters used in this experiment are those in Table 3.5, and we set the SPEA2 archive size to 512. The results are explained below.

5.3.2.1 Experiment: Trade-offs in Detection Programs using DR, Power Consumption and Execution Time.

In this experiment, we evolved the detection programs using three objectives: DR, power consumption, and execution time. Furthermore, we simultaneously minimise power consumption and execution time for the evolved program and maximise the detection rate. We aim to investigate whether there is a trade-off between power consumption and execution time when the detection rate is optimal.

The outcomes of these experiments can be divided into two, as displayed in Figure 5.6 and Figure 5.7. The conditional plot below shows the optimal results proposed for the evolved program using three objectives for the twenty-malware families. For fourteen malware families (Adrd, Basebridge, DroidDream, DroidKungfu, LinuxLotoor, FakeDoc, FakeRun, GinMaster, GoldDream, Imlog, Kmin, MobileTx, Plankton, and SendPay) a high detection rate was obtained in these experiments with trade-offs between power consumption and execution time.

These experiments show that different trade-offs could be obtained between power consumption and execution time when the detection rate is optimal. The trade-offs are discovered by using the MOEC method.

I think the clearest deduction from the 14 graphs is that there is not much variation in the time consumption. However, there is a fair amount of variation in the power consumed. This means effectively that there for optimal detection rates; some programs may be more power-efficient than others.

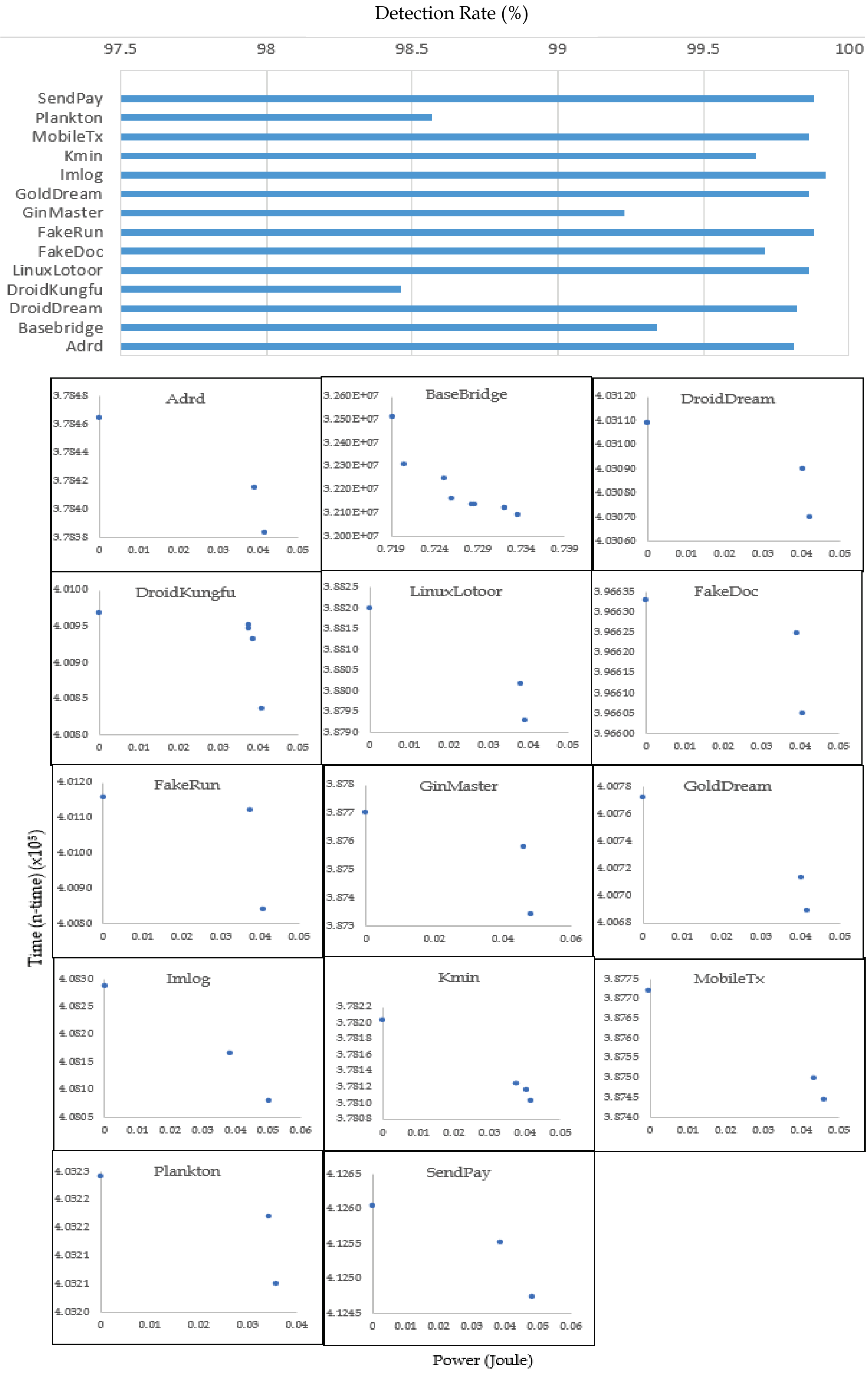


Figure 5. 6 Coplot for Program Evolved with Trade-offs using Three Objectives

The Figure below illustrates the outcomes for six malware families without trade-offs for power consumption and execution time as results at optimal detection rate. All families show that power consumption and execution time vary inversely (i.e. they are in conflict). All detection programs evolved for all five malware families used less than 0.04 Joule. The results in Figure 5.7 show the time to execute the detection program is less than shown in Figure 5.4. This proves the implementation of MOEC capable of reducing power consumption and execution time.

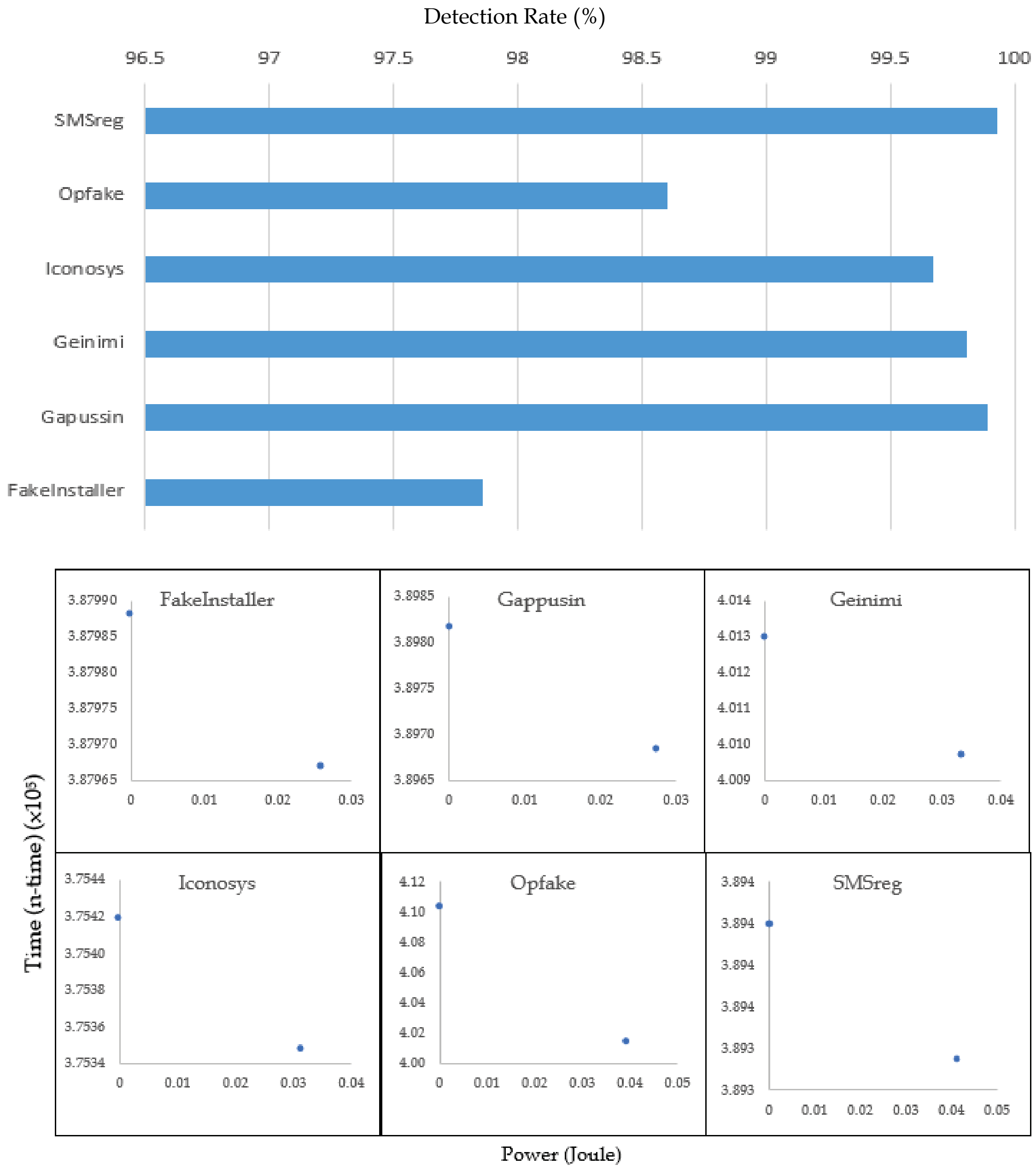


Figure 5. 7 Coplot for Program Evolved without Trade-offs using Three Objectives

CHAPTER 6

Further Investigation of GP Performance on Various Threats on Mobile Phones

This chapter investigates the evolved GP program's performance on three different mobile phone datasets. The current trends for mobile phone attacks are explained in Section 6.1 as our motivation that led to our contribution to this Chapter. The experimental parameters are discussed in Section 6.2. In section 6.3, the efficiency of the evolved GP programs are evaluated using three different mobile phone datasets obtained from other researchers. Finally, in Section 6.4 sumarries the effectiveness of GP performance for detecting various mobile phone attacks datasets.

6.1 Introduction

6.1.1 Motivation

Google Play store was introduced a decade ago, but Google still struggles to protect it. As reported by McAfee, millions of new threats have affected users since its launch, malware in particular [55]. TrendMicro, McAfee, and CSO Online have also released a report about the diversity of mobile phone threats from around 2016 through to 2020, such as mobile ransomware, banking trojans, adware, spyware, ad and click fraud, botnet, dead apps, and Internet of Things (IoT) malware. There are significant trends towards monetisation using mobile malware such as toll fraud and premium SMS scams [55], [191]–[193].

In 2018, McAfee reported that a botnet was used as a spying method to hijack IP cameras. The botnet was also capable of distributing DDoS attacks by bombarding a popular website [55]. The first mobile botnet targeting Android mobile phones was VikingHorde, first detected in 2016 [192] and the first mobile botnet detected in 2009 that effected Symbian phone users [55], [194]. VikingHorde is a botnet, working with root or non-rooted Android mobile phones that use proxied IP addresses, which is capable of masking ad clicks and making revenue for the attackers. Subsequently, in mid-2016, Hummingbad infected over 10 million Android mobile phones [192].

Ransomware has emerged as a major problem in the past few years. For example, in January 2018, mobile phone users from Indiana and New Mexico were affected by the SamSam ransomware [195]. The targeted victims of SamSam ransomware were from hospitals, city municipalities, universities, corporate companies, telecommunication companies, and others. Ransomware criminals now use Bitcoin, LiteCoin, and Monero (digital currency) as the payment method instead of a money transfer using Western Union and PayPal, which was common a few years ago [191], [196].

The events usually impact computer users, but it is possible that mobile phones fell victim to ransomware since they are often connected to the network and other devices by Wifi or Bluetooth. Today, smartphones are also targeted because they are essentially portable and powerful computers. The first iPhone ransomware attacks in May 2014 occurred in Australia, and the first Android ransomware occurred in late August 2014 in United States users [197].

Extensive research has been conducted in the field of mobile phones threats classification, detection, and analysis to counter the rapid growth of mobile phones attacks over the past several years. Trend Micro has identified significant trends for threats to mobiles [175]. Challenges have emerged that pose significant problems, e.g. a diverse range of mobile malware and advanced and targeted malware that includes mobile botnets and a variety of mobile ransomware. In this chapter, we examine three different Android mobile datasets obtained from international researchers: the first is from the DroidAnalytics project and focused on further Android malware; the second concerns Android botnets; and the third concerns ransomware [113], [116], [120].

6.1.2 Contribution

The contribution of the research reported in this chapter is:

- the provision of empirical evidence that GP can produce programs that detect other Android types of attack in the highlighted datasets.

6.2 Experimental Parameters

The parameters used in all three experiments are the same as those given in Table 3.5 in Chapter 3. In this section, we run a GP program with two different sets of features for ransomware datasets. The first experiments use the same selected features as the experiments in Chapter 4. The second experiments use all twenty selected features. We used two different features in these experiments because we want to test which combination of features are best suited to the different type of datasets.

For Malware datasets and Android botnet datasets, only fifteen features were used. For all three datasets, the experiments use 0.5 for both α and β in the fitness function. Table 6.1 shows the shortened identifiers used for features, used to make graphical representations of trees manageable.

Table 6. 1 Feature used in GP and the Short Form to Build Tree

Features	Short-form
ACCESS_COARSE_LOCATION	ACL
ACCESS_FINE_LOCATION	AFN
INSTALL_SHORTCUT	IS
INTERNET	I
MODIFY_PHONE_STATE	MPS
READ_CONTACT	RC
READ_HISTORY_BOOKMARKS	RHB
WRITE_HISTORY_BOOKMARKS	WHB
READ_SMS	RS
SEND_SMS	SS
WRITE_SMS	WS
READ_PHONE_STATE	RPS
VIBRATE	V
WRITE_APN_SETTINGS	WAS
WRITE_EXTERNAL_STORAGE	WES
BLUETOOTH	B
DISABLE_KEYGUARD	D
RECEIVE_BOOT_COMPLETED	RBC
SET_WALLPAPER	SW
WAKE_LOCK	WL
Contain	C
Not	N
O	O
X	X

Each malware family is the target of ten runs. The ten runs in each case are executed using ECJ's job function. The best results achieved are discussed in Section 6.3.

6.3 Discovering GP Performance using Different Datasets

In this section, the effectiveness of our proposed IDS is investigated by using three different Android mobile phones threat datasets. The datasets used are from real Android applications and real malware. They have been acquired from established researchers. All datasets were pre-processed to extract the Manifest.xml information from APK.

6.3.1 Malware Datasets

The evaluation of the evolved GP programs against Android malware samples is described in this section. These experiments are extended from experiments completed in Chapter 4, where GP successfully detected Android malware in the DREBIN datasets [119]. Here we test our evolved GP programs with different Android malware datasets. The DroidAnalytics datasets contain real malware (current at the time it was collected).

DroidAnalytics is a cloud based APK scanner and functions as an Android malware analysis system. The research generated signatures for the malware and facilitated information retrieval [116]. The DroidAnalytics datasets include 98 malware families (although their paper indicates 102 malware families). The datasets comprise 2,475 malware samples and include 327 zero-day malware samples from six different malware families.

6.3.1.1 Experiments Overview

We use 17 of the 98 malware families supplied in DroidAnalytics datasets (i.e. the same as for the experiments in Chapter 4). These 17 are selected because they contain enough samples to allow a plausible split into meaningfully sized training and testing components. We also use precisely the same features (permissions) as the experiments in Chapter 4 identified as giving the best detection.

The training datasets used in these experiments are presented in Table 6.2 and the testing datasets, as shown in Table 6.3 below.

Table 6. 2 Training Datasets

Family	Malware	Non-malware	Family	Malware	Non-malware
FakeInstaller	919	40,467	Adrd	85	42,771
DroidKungfu	662	40,365	DroidDream	80	40,305
Plankton	620	40,570	LinuxLotoor	64	40,043
Opfake	608	40,552	GoldDream	64	40,067
GingerMaster	334	40,201	MobileTx	69	40,028
BaseBridge	324	42,752	FakeRun	56	40,092
Iconosys	145	40,081	SendPay	54	40,294
Kmin	142	40,126	Gapussin	53	40,038
FakeDoc	127	40,078	Imlog	38	40,276
Geinimi	87	40,056	SMSreg	36	40,043

Table 6. 3 Testing Datasets

Family	Malware	Non-Malware	Family	Malware	Non-Malware
FakeInstaller	3	42,787	Adrd	141	42,742
DroidKungfu	142	42,127	Rooter	14	42,945
Plangton	126	42,399	LinuxLotoor	112	42,729
Opfake	8	42,414	GoldDream	6	42,707
GingerMaster	31	42,419	MobileTx	15	42,734
BaseBridge	546	42,667	SendPay	9	42,975
Kmin	192	42,668	Imlog	2	42,691
FakeDoc	1	42,645	SMSreg	1	42,714
Geinimi	97	42,924			

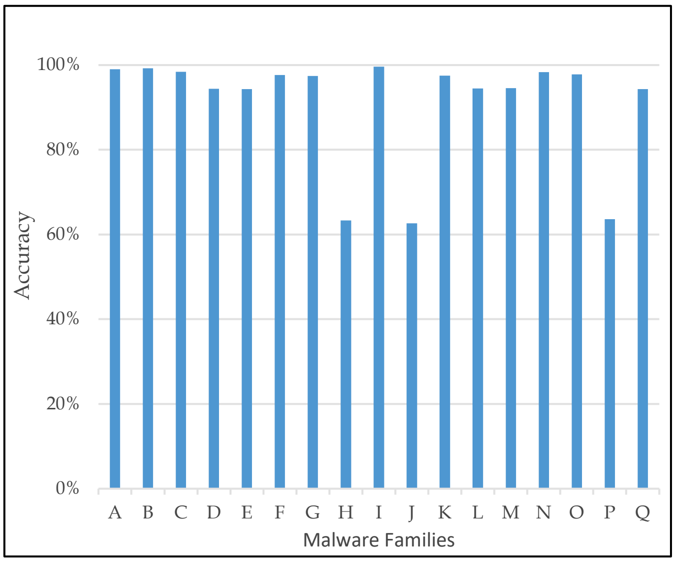
6.3.1.2 Results and Discussion

In this section, the details of each particular malware family and the results are discussed. Table 6.4 below provides descriptions for each of the Android malware families, the year they were discovered, and their capabilities. In the table, only 12 malware families are described as another 5 of malware families have been described in Chapter 3. Figure 6.1 shows the best individual results.

Table 6. 4 Android Malware Families, Year Detected and Their Capabilities

Family (Year)	Capabilities [155], [198]–[204]
Geinimi (2010)	Transmits info, contact details and geographic location from the device to a remote location. It can also upload SMS data to remote servers, call or send an SMS to a specified number, delete SMS messages, silently downloading files, snatching a list of installed applications and uploading it to the command and control (C&C) server, installing or uninstalling the software. It also can show a map or a Web page, show a pop-up message, change the device wallpaper, create a shortcut, and change a list of command and control servers when instructed by hackers.
Adrd (2011)	Uploads device specific data to remote servers via DES-encrypted communication.
BaseBridge (2011)	Sends SMS messages to predetermined numbers with a premium rate, deletes SMS messages, dials phone numbers, monitors phone usage and terminates browser application.
GoldDream (2011)	Spies on SMS messages received and incoming/outgoing phone calls by users and then uploads them to a remote server without the knowledge of the users. It can fetch and execute commands from a remote C&C server.

Rooter (2011) was known as DroidDream	It first gains root privilege on a remote device then takes control of the mobile system.
Imlog (2011)	Sends device info such as maker, manufacture and model to a particular website.
FakeDoc (2012)	Sends sensitive information such as Contact List, User email address, Phone number, Device Information (IMEI, model, manufacturer, OS version, screen size) and Device Location to a remote server.
Lotoor also was known as ExploitLinuxLotoor(2012)	This is a universal detector for hack tools that use vulnerabilities to gain root privileges on affected Android devices.
MobileTx (2012)	Steals info from the affected device and sends SMS messages to a premium rate number.
Kmin (2012)	Sends IMEI and phone number to a remote server, sending SMS messages to a premium number such as 10669500718. It also can download and install another application without the user being aware of it.
SendPay (2012)	Can handle remote access connections, accomplish DoS or DDoS, capture keyboard inputs, delete files or objects, or terminate processes.
SMSReg (2012)	Collects the API key, application ID, carrier, device manufacturer, device model, GPS location, IMEI number, network operator, package name, and SDK version.



Legend			
A	Adrd	J	Kungfu
B	Basebridge	K	Lotoor
C	FakeDoc	L	MobileTx
D	FakeInst	M	Opfake
E	Geinimi	N	Plangton
F	GinMaster	O	Rooter
G	GoldDream	P	SendPay
H	Imlog	Q	SMSreg
I	Kmin		

Figure 6. 1 GP Evolved Program Performance

Performance seems varied. Figure 6.1 shows eleven out of nineteen malware families being detected with an average accuracy of more than 91% and three families (Adrd, BaseBridge and Kmin) with an accuracy of nearly 100%. Nevertheless, the detection accuracy of three malware families (Imlog, Kungfu and SendPay) is low (63%, 63% and 64% respectively). Table 6.5 below provides details for these three malware families. As we can see, the real issue is with false positives.

Table 6. 5 Imlog, Kungfu and SendPay Results

Family	TPR	TNR	FPR	FNR	ACC
Imlog	100%	63%	37%	0%	63%
Kungfu	100%	63%	37%	0%	63%
SendPay	100%	64%	36%	0%	64%

The low detection rate for SendPay is the same as found in Table 4.6. We can conclude that this has happened because the evolved GP program combines permissions that could not give the best results. As shown in Figure 6.2 for Imlog, Kungfu and SendPay, the combination of permissions evolved by the GP includes quite a few that might also be included in non-malicious samples. The trees of the best individual evolved by GP for each of six malware families are shown in Figure 6.2 and Figure 6.3. All evolved programs gave a different tree for each malware family because each of them contains different permission being selected for the best individual.

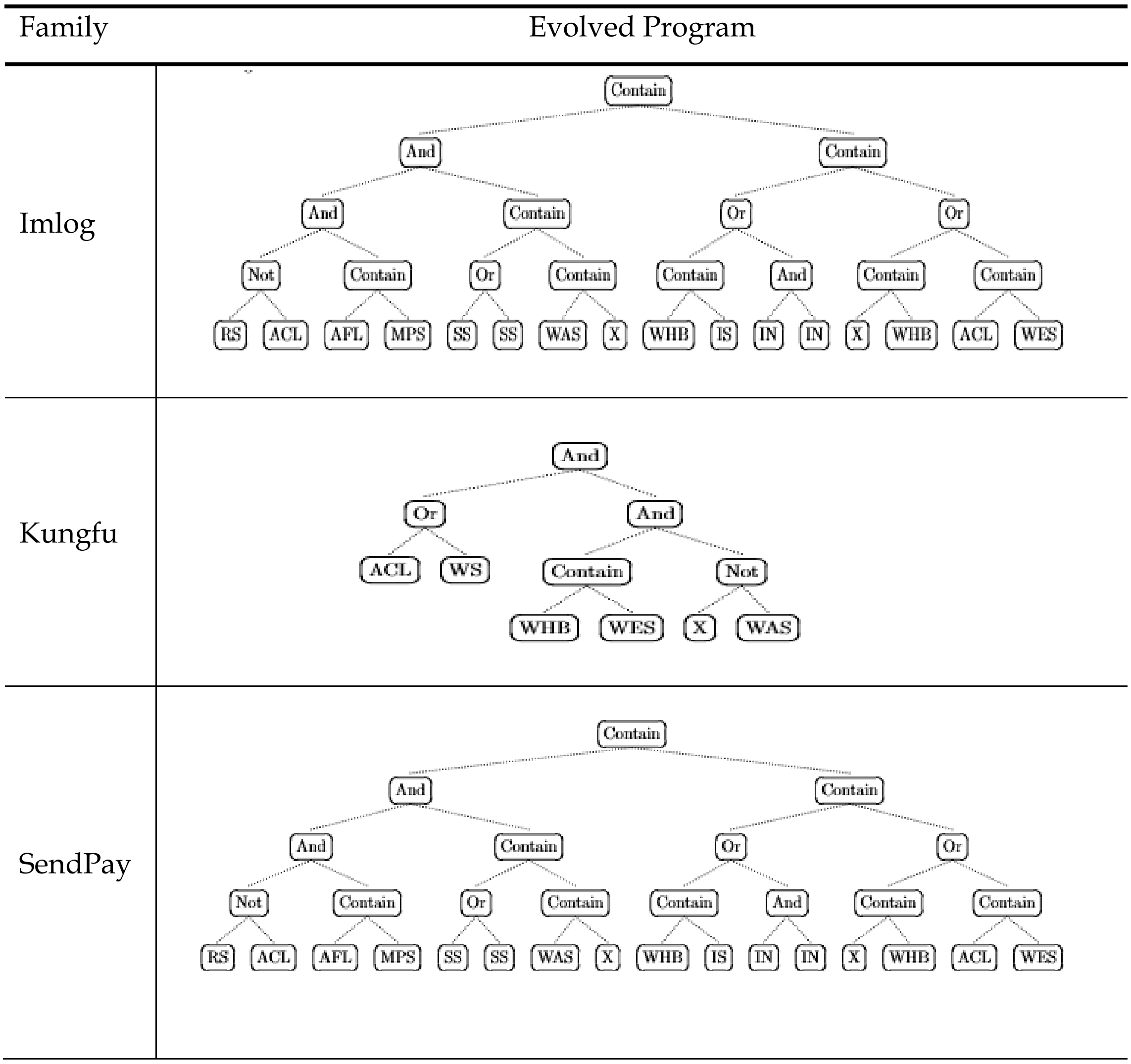

Family
Evolved Program
Imlog
Contain
And Contain
And Contain Or Or
Not Contain Or Contain Contain And Contain Contain
RS ACL AFL MPS SS SS WAS X WHB IS IN IN X WHB ACL WES
Kungfu
And
Or And
ACL WS Contain Not
WHB WES X WAS
SendPay
Contain
And Contain
And Contain Or Or
Not Contain Or Contain Contain And Contain Contain
RS ACL AFL MPS SS SS WAS X WHB IS IN IN X WHB ACL WES

Figure 6. 2 Best Individual Trees for Imlog, Kungfu, and SendPay

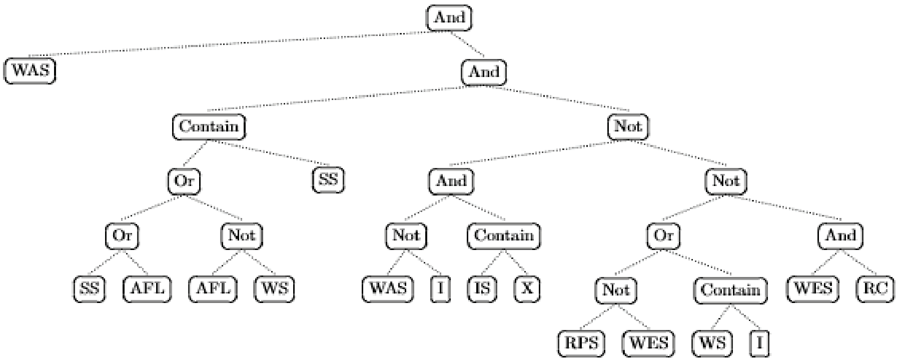
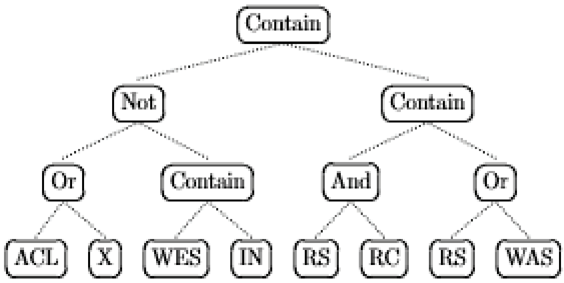
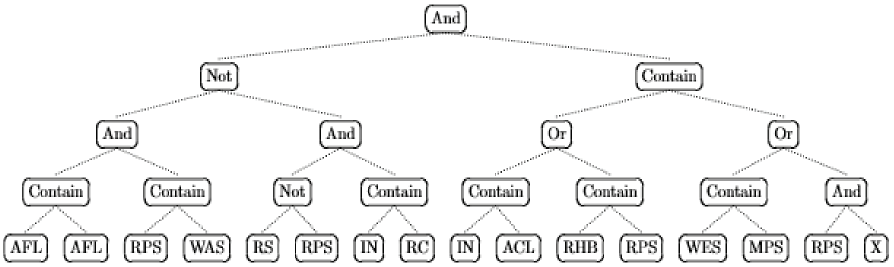
Family	Evolved Program
Adrd	
BaseBridge	
Kmin	

Figure 6. 3 Best Individual Trees for Adrd, BaseBridge, and Kmin

6.3.1.3 Results Conclusion

The results are mixed. We can achieve more than 91% accuracy for 14 out of 17 malware families. However, some apps are less amenable to detection via our approach. For three apps the lack of accuracy is entirely due to false positives, so all malware is detected, but time may be wasted analysing apps erroneously flagged as malware. In all cases these results have been obtained by programs with limited complexity/depth (as is apparent from Figure 6.2 and Figure 6.3).

6.3.2 Android Botnets Datasets

An Android botnet can disguise itself as a trusted Android APK and the damage only happens when the hackers activate the botnet. Once users realise their device is infected by the botnet, it is too late because the botnet owner has already taken control of their devices using command and control (C&C) software. Mobile phone targetted botnet attacks began around 2012. The following year saw an increase ([205]) that served to inspire researchers to address the issue.

Researchers from Georgia Teach [206] revealed that 23% of Windows systems showed marks of a botnet infection. According to McAffee, mobile botnets and C&C outbreaks increased from 2016 to 2017 by 22% [55]. This rise in botnets motivated us to consider the use of our GP approach for their detection. As far as we are aware, GP has never been investigated for this purpose.

Kadir et al. [113] provide a significant collection of Android mobile botnet datasets for 14 botnet families. The collection includes botnet datasets from the Malgenome project, malware security blog, VirusTotal and samples supplied by acknowledged anti-malware suppliers. In general, the mobile botnets dataset consists of 1929 samples covering a period between 2010 (the first presence of Android botnet) and 2014.

6.3.2.1 Experiments Overview

In these experiments, all mobile botnet APKs have been processed to extract the information from the Manifest.xml. The source of trusted APKs came from the DREBIN datasets [119] because the provided botnets dataset did not contain any trusted APKs. For two families indicated in [113], the data pre-processing led to the remove of some samples. The two families are Geinimi and Pjapps. Both training and testing datasets are randomly selected. The division of testing and training datasets is as in Table 6.6 and Table 6.7 below:

Table 6. 6 Training Datasets

Family	Botnets	Trusted APK	Family	Botnets	Trusted APK
Anserverbot	5	42,771	PJapps	5	40,570
Bmaster	5	42,752	Pletor	5	40,365
DroidDream	5	40,304	Rootsmart	5	40,275
Geinimi	5	40,056	Sandroid	5	40,126
MisoSMS	5	40,043	TigerBot	5	40,552
NickySpy	5	40,294	Wroba	5	40,043
NotCompatible	5	40,304	Zitmo	5	40,078

Table 6. 7 Testing Datasets

Family	Botnets	Trusted APK	Family	Botnets	Trusted APK
Anserverbot	239	42,742	PJapps	231	42,399
Bmaster	1	42,667	Pletor	80	42,127
DroidDream	358	42,945	Rootsmart	23	42,691
Geinimi	233	42,924	Sandroid	39	42,668
MisoSMS	95	42,714	TigerBot	91	42,414
NickySpy	194	42,975	Wroba	95	42,729
NotCompatible	71	42,945	Zitmo	75	42,645

6.3.2.2 Results and Discussion

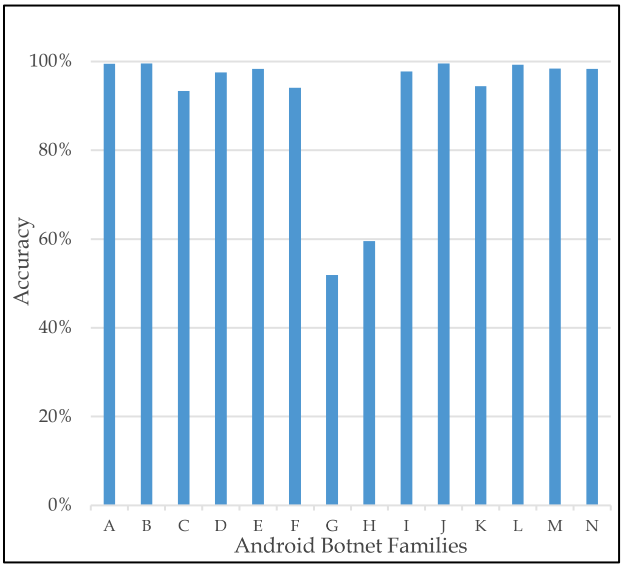
The details of the Android mobile botnet families and the results are discussed in this section. The Android botnet families, the year it was detected, and their abilities are reflected in Table 6.8 below. In the table, 12 botnet families are discussed, as Geinimi and DroidDream can be referred to in Table 6.4.

Table 6. 8 Android Botnets Families, Year Detected and Their Capabilities

Family (Year)	Capabilities [155], [200], [203], [207]–[209]
Zitmo, 2010	This botnet can obstruct one-time passcodes supplied by banks to mobile devices as a security feature of logging into their accounts or making account modifications relating to sensitive data, and it will send all incoming text messages to a remote server.
Anserverbot (2011)	This botnet ran quietly in the background to connect to the blog, decrypt a URL string, and then connect to that server.
NickySpy (2011)	This botnet collects devices IMEI, access cell-ID, WIFI location and updates, GPS location and WIFI network details and sends it via SMS message. It can initiate a phone call without going through the dialer GUI so that the user is unaware of any outgoing calls, monitor, modify, or abort outgoing calls, open network sockets, read SMS messages, obtain the user's contacts data, record audio, send SMS messages, and write (but not read) the user's contacts data.
PJapps (2011)	This botnet had numerous features included application installation, visiting Web sites, adding bookmarks to the browser, and sending and blocking text messages.
TigerBot (2012)	This botnet opens a back door on the compromised device and listens for specifically created SMS messages, allowing an attacker to execute, stop and start processes and services, change network settings, send the contact list to a remote location, take screenshots, reboot the compromised device, record incoming and outgoing call numbers, and deactivate the software.
Rootsmart (2012)	This botnet links to a remote location to get the GingerBreak root exploit required to gain root permission on the compromised device.
Bmaster (2012)	This botnet exfiltrates sensitive data from the phone, including the device ID, GPS data, and IMEI number. It is also capable of sending SMS messages to premium numbers.

MisoSMS (2013)	This botnet was capable of gathering and sending SMS messages to remote servers in China.
NotCompatible (2014)	This botnet was proficient at delivering access to private networks by transforming an infected device into a network proxy, which could then be used to gain access to other protected data or systems.
Pletor (2014)	This botnet can control the phone and its data, lock user devices then ask for ransom.
Sandroid (2014)	This botnet can intercept all incoming SMS messages and gathers relevant banking information. The victim’s code, username and password are unprotected. These data are automatically transferred to the botnet-master.
Wroba (2014)	This botnet can start remote access connections, capturing keyboard input, collecting system information, downloading/uploading files, dropping other malware into the infected system, performing denial-of-service (DoS) attacks, and running/terminating processes.

Figure 6.4 below shows the GP evolved program performance on several Android Botnet families; an average accuracy at 91.55% is demonstrated. Shortened identifiers for the families are given alongside the graph. The results show that four (A, B, J, and L) botnets families gave rise to an accuracy of 99% and eight (C, D, E, F, I, K, M and N) botnet families an accuracy of 93%. Nevertheless, two (G and H) botnets families incurred a very low accuracy of 60%.



Legend			
A	Anserverbot	H	PJapps
B	Bmaster	I	Pletor
C	DroidDream	J	Rootsmart
D	Geinimi	K	Sandroid
E	MisoSMS	L	Tigerbot
F	NickySpy	M	Wroba
G	NotCompatible	N	Zitmo

Figure 6. 4 GP Performance on Android Botnets

Our work differs from that of [113]. They do behavioural analysis based on URL patterns analysis, and we seek to detect Android botnets and distinguish them from clean APKs using permission use. We are able to improve the detection rate for the Sandroid family to 94.45%. Previously the best rate was 86% [113].

In detail, NotCompatible and PJapps have the lowest accuracy; however, their TPR shows 100% and 91.53% accordingly. Our GP evolved program can detect all Android samples from clean APK samples, but the permission combination is also contained in the clean APK, and so the GP identifies it as a suspicious. Thus, permissions here are simply insufficiently discriminating (whatever technique is used for detection). Below in Figure 6.5 and Figure 6.6 are the examples of the best individual trees evolved in these experiments for the best detection performance and the lowest detection rate. The best individual trees for PJapps show when the APK permission contains RPS it will be detected as a botnet but that clean APKs with that permission will also be subject to such classification.

Figure 6.5 and Figure 6.6 give the best individual trees for six Android botnet families. It shows there is a varied form of tree evolved by GP for each of the botnet families. We can conclude the combination of permissions needed to detect each botnet family is different.

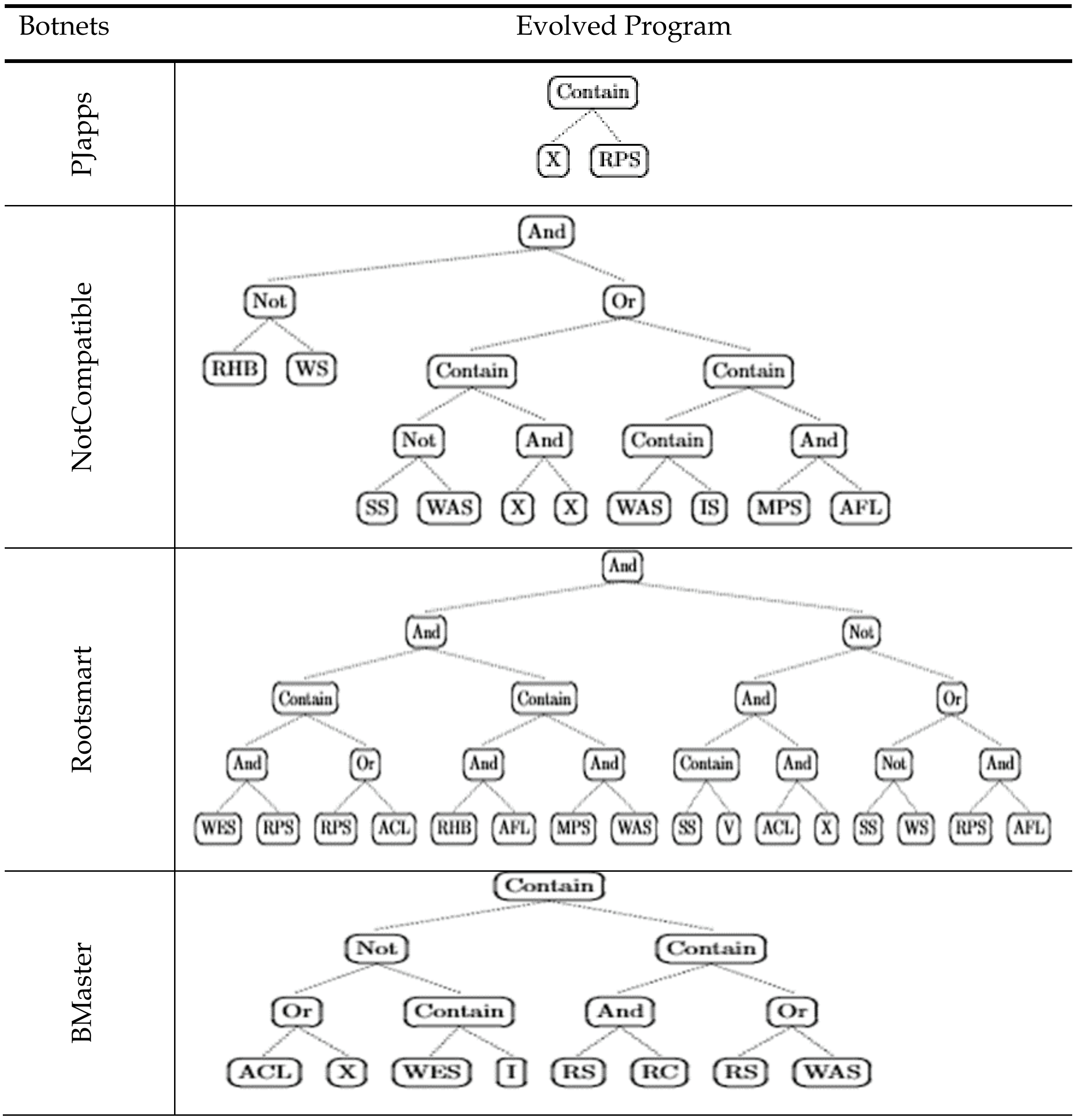


Figure 6. 5 GP Tree Evolved for Four Botnet Families

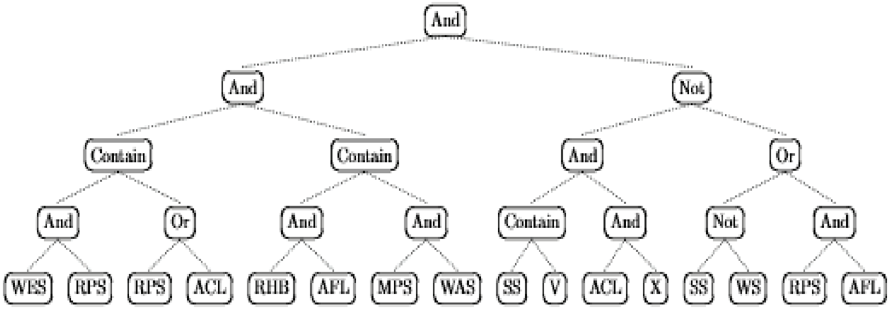
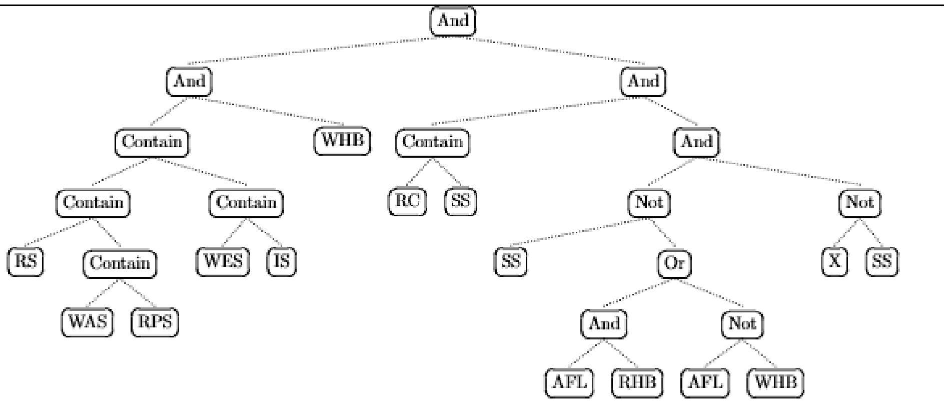
Botnets	Evolved Program
Anserverbot	 And And Not Contain Contain And Or And Or And And Contain And Not And WES RPS RPS ACL RHB AFL MPS WAS SS V ACL X SS WS RPS AFL
DroidDream	 And And And Contain WHB Contain And Contain Contain RC SS Not Not RS Contain WES IS SS Or X SS WAS RPS And Not AFL RHB AFL WHB

Figure 6. 6 GP Tree Evolved for Two Botnet Families

6.3.2.3 Results Conclusion

Since the first appearance in 2010 [8] Android botnets remained a major form of a threat right through to 2018 [55]. Previously researchers used approaches such as static analysis [194], [210], behavioural analysis [113], [211], SVM [103] to study Android botnet characteristics. In this section, we have investigated whether a GP evolved program can distinguish Android botnets from clean APK. We show that GP is capable of evolving programs to detect Android botnets. Nevertheless, some results do not achieve a detection rate of 90% for some Android botnet families.

No optimal tree size can be discerned that gives the best performance overall. Bigger is certainly not better. It would seem prudent to experiment with several max tree depths for GP tree evolution. The results may also reflect limitations of the datasets e.g., the BMaster dataset contained only six samples [113].

6.3.3 Ransomware Datasets

On Friday 12 May 2017, nations across the world experienced outbreaks of WannaCry ransomware attacks. WannaCry encrypted data and asked for payment to recover files. The UK's NHS (National Health Service) was one of the worst affected [72]. In mid-2018, a new mobile ransomware variant detected by Check Point researchers called Charger succeeded in breaking into the Google Play store by using numerous obfuscation methods [73].

Mobile ransomware is predicted to continue to grow and develop new and more robust capabilities targeting increased profits. We are therefore motivated to test whether our GP evolved programs can detect mobile ransomware. The mobile ransomware datasets used are those mentioned in the paper "Ransomware Steals Your Phone. Formal Methods Rescue It" [120]. They used two sets of datasets for mobile ransomware [120]. The first mobile ransomware sample is a publicly available collection from two well-known websites: Contagio Mobile [212] and Ransom Mobi [213]. Only the Contagio Mobile page is still available up to now, and the Ransom Mobi webpage is now obsolete. The datasets were collected between December 2014 – June 2015. The datasets are selected because they also used 600 samples from the DREBIN datasets as the second source of samples. The researchers did not mention which samples they used in their paper [120]. Therefore, we randomly select the samples.

6.3.3.1 Experiments Overview

The samples obtained from downloading the APK from the websites stated above had to undergo preprocessing as described in Chapter 3 to extract the information needed for the experiments from the Manifest.xml files. Six hundred and eighty samples were gathered. We removed three duplicates in the files provided by the authors of [9].

Some files did not contain Android permissions in their Manifest.xml file; therefore, we could not use them in our experiments. In these experiments, the trusted APK samples come from the DREBIN datasets [119], and both training and testing datasets are randomly selected. Details of the testing and training datasets are as follows in Table 6.9 below:

Table 6. 9 Ransomware Datasets

Training		Testing	
Ransomware	Trusted APK	Ransomware	Trusted APK
5	40126	675	42668

6.3.3.2 Results and Discussion

Table 6.10 shows the performance of GP using fifteen features selection, and Figure 6.6 shows the best individual tree for the GP evolved program:

Table 6. 10 GP Performance towards Ransomware using 15 Features

Experiments	TPR	TNR	FPR	FNR	Accuracy
Job 0	96.59%	59.67%	40.33%	3.41%	60.25%
Job 1	96.59%	59.67%	40.33%	3.41%	60.25%
Job 2	96.59%	59.67%	40.33%	3.41%	60.25%
Job 3	96.59%	59.67%	40.33%	3.41%	60.25%
Job 4	96.59%	59.67%	40.33%	3.41%	60.25%
Job 5	96.59%	59.67%	40.33%	3.41%	60.25%
Job 6	96.59%	59.67%	40.33%	3.41%	60.25%
Job 7	96.59%	59.67%	40.33%	3.41%	60.25%
Job 8	96.59%	59.67%	40.33%	3.41%	60.25%
Job 9	96.59%	59.67%	40.33%	3.41%	60.25%

Table 6.10 gives results from the best-evolved program for each of ten runs. At the end of each run, the TPR, TNR, FPR, FNR and Accuracy are calculated. There is no variation in results between jobs. The results above are not encouraging as the accuracy and TNR of all runs are 60.25% and 59.57% respectively.

Nevertheless, the TPR and the FNR show some encouraging results at 96.59% and 3.41%. These results show that the GP evolved program can contribute to truncated FPR at 40.33%. However, the FNR is still at low and did not even achieve 5%. The results applied for 23 ransomware APK samples not correctly detected as ransomware.

In these experiments, we used unbalanced datasets (with more non-malware samples than malware samples). The results are identical across all runs but, as can be seen in Figure 6.8, the actual program trees vary.

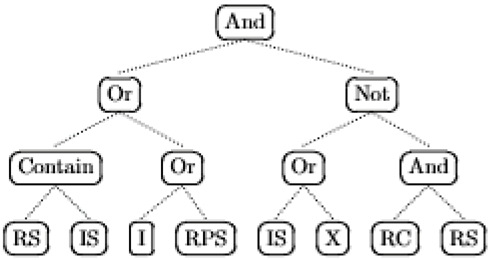
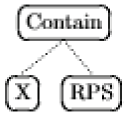
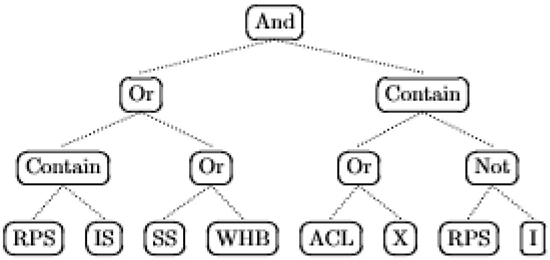
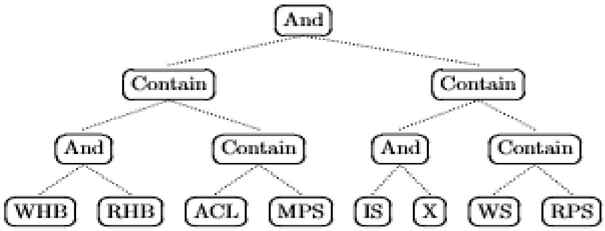
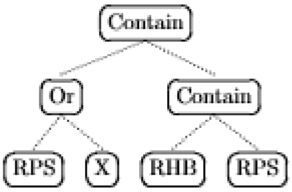
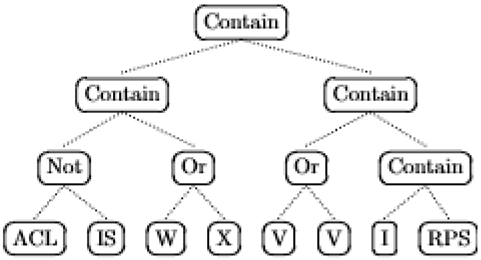
Experiments	Evolved Program
Job 0	 And Or Not Contain Or Or And RS IS I RPS IS X RC RS
Job 1	
Job 2	 And Or Contain Contain Or Or Not RPS IS SS WHB ACL X RPS I
Job 3	 And Contain Contain And Contain And Contain WHB RHB ACL MPS IS X WS RPS
Job 4	
Job 5	 Contain Contain Contain Not Or Or Contain ACL IS W X V V I RPS

Figure 6. 7 Best Individual GP Evolved Program for Ransomware using 15 Features

In the figure above, there are five different trees evolved by GP to distinguish ransomware APK samples from clean APK samples, but they have the same performance. The other four experiments run to display the same tree build in Job 1, which is only one feature that turns out could detect ransomware APK samples. The short form of used features can be referred to in Section 6.2. In all trees from Job 0 to Job 5 shown, the tree must contain RPS to be able to distinguish the ransomware APK sample from clean samples. The RPS is capable of allowing access to the phone state, which is now identified as the permission that allows an intruder to get access to the mobile phone. As we know ransomware accomplished to lock the mobile phone from the user unless they pay the ransom. We want to investigate further either GP evolved program can detect ransomware samples more efficiently than the dataset's owner managed to get (99.53% TPR [120]). Table 6.11 shows the results when we use 20 features in our GP framework. All 20 features are discussed in Table 3.4.

Table 6. 11 GP Performance towards Ransomware using 20 Features

Experiments	TPR	TNR	FPR	FNR	Accuracy
Job 0	99.41%	93.02%	6.98%	0.59%	93.12%
Job 1	99.41%	93.02%	6.98%	0.59%	93.12%
Job 2	99.56%	93.02%	6.98%	0.44%	93.12%
Job 3	99.41%	93.02%	6.98%	0.59%	93.12%
Job 4	99.41%	93.02%	6.98%	0.59%	93.12%
Job 5	99.41%	93.02%	6.98%	0.59%	93.12%
Job 6	99.41%	93.02%	6.98%	0.59%	93.12%
Job 7	99.41%	93.02%	6.98%	0.59%	93.12%
Job 8	99.41%	93.02%	6.98%	0.59%	93.12%
Job 9	99.41%	93.02%	6.98%	0.59%	93.12%

In these experiments, ten runs of experiments are represented as Job 0 to Job 9 in Table 6.11. For each Job, the TPR, TNR, FPR, FNR and Accuracy are calculated. Job 2 shows the highest TPR at 99.56% and FNR at 0.44% which denote only three ransomware samples flag as non-attacks out of 675 samples; the results are better than others' detection rate TPR at 99.41% and the FNR at 0.59. Although the owner of the dataset managed to get 99.53% TPR [120], slightly different 0.02% from detection using GP evolved program; we cannot compare the results directly. Only used 675 samples out of 1,271 samples used by them because they did not specify which samples they selected from DREBIN datasets and which other samples they were using.

Other results do not indicate any significant change as the TNR is 93.02%, the FPR is 6.98%, and the accuracy for all experiments is 93.12%. This results of high TPR are induced by the unbalanced datasets implicated in Table 6.11. Nine out of ten experiments gave the same results. Figure 6.9 shows some of the best individual GP evolved program trees.

The results in Table 6.11 above show almost identical performance across the 10 runs. (9 of the 10 are identical, with Job 2 improving TPR and FNR marginally.) The results in Figure 6.9 show they need RPS to distinguish ransomware samples from clean samples. Figure 6.8 shows that the RBC permission can have a very significant impact, allowing a TPR rise to 99%. (RBC was not an available feature in the first experiments.) The RBC permission allows an application to obtain the ACTION_BOOT_COMPLETED that is broadcast after the system finishes booting. This feature gives hackers the ability to control the mobile phone once they penetrate the system.

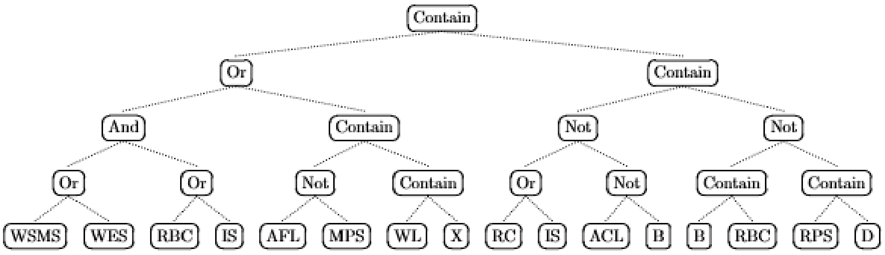
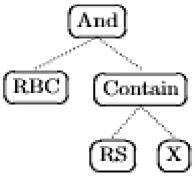
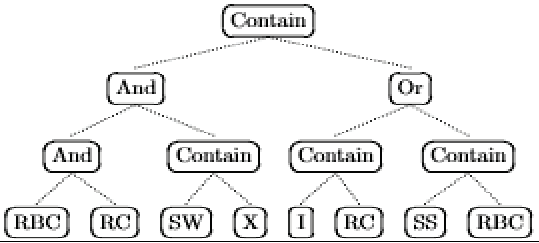
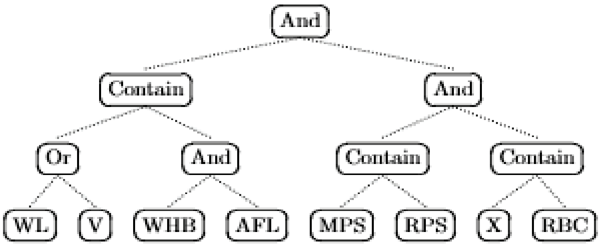
Experiments	Evolved Program
Job 0	 Contain Or Contain And Contain Not Not Or Or Not Contain Or Not Contain Contain WSMS WES RBC IS AFL MPS WL X RC IS ACL B B RBC RPS D
Job 1	
Job 3	 Contain And Or And Contain Contain Contain RBC RC SW X I RC SS RBC
Job 5	 And Contain And Or And Contain Contain WL V WHB AFL MPS RPS X RBC

Figure 6. 8 Best Individual GP Evolved Program for Ransomware

6.3.3.3 Results Conclusion

Results showed the mobile ransomware could be detected using the evolved GP program at 96.59% TPR in the Android platform when the GP program was evolved with fifteen features. The evolved GP program's TPR increased to a 99.56% detection rate when using twenty features. We run the experiments using two sets of selected features because this is the first time GP has been used to evolve programs to detect ransomware. The result is encouraging; our sole focus on APK permissions facilitates a simple charactersation of benign and malicious apps.

6.4 Conclusion

In this section, we run three different datasets of infected Android APK by malware, botnets and ransomware to investigate the performance of an evolved detection program. For all experiments, the evolved GP program run targeted a specific family and the outcome shows each family has a different character because the evolved trees are different from each other. All datasets for training and testing used unbalanced samples of harmful APKs and clean APKs. The reason is that we want to consider real-world situation in GooglePlay there are millions of APKs uploaded, and there are probably one out of thousand is a 'harmful' APK. We also followed the DREBIN implementation, using a combination training and testing datasets with unbalance data.

In summary, our GP-based approach, when applied to three datasets achieved 90% accuracy detection rate, and some of them gave rise to an accuracy of 99%. The results prove GP can be used for a different type of malware detection.

CHAPTER 7

Conclusion

This chapter summarises the research carried out and discusses the contributions made. The thesis hypothesises presented in Chapter 1 is revisited in the light of the results obtained. Finally, future work is identified.

7.1 Review of Experimentation

The problems of existing methods for intrusion detection in Android mobile phones have been identified in Chapter 2. The primary concerns can be summarised as follows:

a) The previous detection of malfeasance in Android mobile phones suffers from low detection efficiency caused by high false alarm rates.

b) Detection of attacks and malicious activities in mobile phones is a challenging research problem because fast detection response is needed to avoid further damage to mobile phones.

c) Android mobile phones have limited resources. Their batteries have a limited life, and existing solutions do not acknowledge this problem appropriately.

Researchers have mainly focused on the first two issues so far. Nevertheless, the concern about power consumption and execution time for detection is vital. In this thesis, the limited power resources of the mobile phones are taken into consideration to deliver more effective detectors, and execution time has also has been taken into account in our mobile IDS synthesis experiments.

This research explores the implementation of EC approaches, particularly the use of GP, to evolve intrusion detection programs for Android mobile phones. We aimed to synthesise a high performing and reliable IDS for Android mobile phones.

This thesis shows how to use EC methods for the synthesis of detectors and classifiers of malware on Android mobile phones platform. A variety of experiments evaluates the hypothesis of *thesis statement 1: GP will be able to evolve programs to distinguish malicious applications from the non-malicious applications in mobile phones datasets*.

The performance of GP evolved programs is evaluated on real-world mobile phone application datasets that have been obtained from Android research community. The approach shows good performance for detecting twenty malware families at an average of 93% accuracy with a poor false alarm rate of 7%. Our approach achieves the same average accuracy but has a higher false alarm rate than Drebin. Limitations and possible causes of those limitations have been identified, e.g. unbalanced datasets for training and testing and very limited size of datasets for some malware families.

A standard form of fitness function was adopted. We proceeded to investigate variation in weights for the components of the overall fitness function. The results of GP evolved program with optimal parameter using different weight for fitness function show that GP can achieve high detection rate with a low false alarm rate. The average accuracy increases to 98.8%, and the average false alarm rate decrease to 0.89%. With all malware families, the detection rate (accuracy) is above 95% per family. The results outperform those of the dataset's owner in both average accuracy and false alarm rate. The results have shown that GP can indeed evolve programs that can distinguish suspicious Android APKs from clean APKs only using Android permissions extracted from Manifest.xml file as its selected features.

The work on weightings for our GP synthesis of detectors shows that the optimal values for alpha and beta may vary significantly between malware families. This may place limits on the best that can be achieved by any ML approach based on optimising a fitness function such as the one used in this work, i.e. it is unclear just how much further improvement can be expected.

There are potential limits on the overall approach. Privileges have a purpose, and specific applications need specific privileges. With some applications, we are happy for them to run with them; they are trusted to exercise those privileges in a means that accords with our interests. However, malicious applications may simply have sets of privileges that are identical to those possessed by valid apps. In which case any detection based purely on permissions will necessarily is classify. In some ways we might well be surprised at just how well permission based discrimination has worked.

Weighted fitness functions are a staple approach in GP and related evolutionary search approaches. They provide *one means* of approaching multi-criteria tradeoffs.

One of the critical problems mobile phones face is the limited power supply. MOEC is employed to investigate the relations between detection rate, power consumption and execution time of the evolved program.

The findings show there are a set of solutions with different trade-offs amongst the three chosen objectives is achieved for each malware family. The results show to obtain optimal detection rate the power consumption is decreased, but the execution time is increased and vice-versa when the power consumption increases the execution time will plunge. Nevertheless, we also obtained results from five malware families that do not furnish us with trade-offs among the objectives. The findings also show that the best detection rate is achieved with high execution time and low power consumption. Thus, the power consumption will increase when the execution time is reduced. The conclusion can be drawn that the best detection rate can be achieved with low power consumption (but with response time being the price we have to pay). The outcomes from the experiments serve as an evaluation of *thesis statement 2: GP and MOEA can be set to synthesise a system capable of the efficient detection of malware on mobile phones, e.g. using limited battery power.*

Finally, we evaluated the GP evolve program performance towards other types of malware, as mentioned in *thesis statement 3*: *The performance of new IDS is evaluated by using different mobile phone datasets*. Estimation of this hypothesis is performed by using three different datasets acquired from three different Android malware projects, and a range of experiments was carried out as described in Chapter 6. This work explores the performance of GP evolved programs on malware families using a different source of training and testing datasets. All evaluation used different GP evolved programs based on the malware families.

The results for the DroidAnalytics malware datasets indicate that the evolved GP program can detect 14 malware families at average accuracy at 91% with three malware families attaining the lowest accuracy detection rate at almost 63% (but their false negative rate is 0%). The reason is that the fraction of training and testing datasets are unbalanced. The performance of the Android botnets datasets shows the GP evolved program achieved a 93% average accuracy for eight Android botnets families with four Android botnets families producing 99% accuracy detection rate. However, two families resulted in less than 60% accuracy detection rate. The average accuracy for both families decreases to 91.55%. The last datasets which are ransomware show the performance of GP evolved programs can achieve 99.56% TPR, 6.98% FPR and 93.12%accuracy. We can conclude that other type of malware can be detected using GP evolved programs with average results at above 91%. We had suffered from a high false-positive rate as we knew anomaly-based IDS faced this problem [214]. However, for some malware families, the results are plausible, and on average our detector results still acceptable (due to almost perfect TPRs).

7.2 Thesis Contributions

The primary contributions of this research are defined as follows:

Evolutionary computation approaches for intrusion detection in mobile phones:

This study investigates the use of AI to develop intrusion detection programs for this challenging new environment. EC methods principally “breed" intrusion detection programs by assessing populations of prospective programs and subjecting them to a variety of genetical operators. In this thesis, we have demonstrated that GP can be used to evolve effective detectors for mobile phone attacks such as viruses, Trojan horses, mobile botnets, adware, rootkits, spyware and worms. To the best of our knowledge, this is the first investigation of Android mobile phones IDSs developed using GP used only Android permissions as features. Previous work has usually implemented SVM or

GA (EC method), as explained in Chapter 2. In [128], they used GP as a method to do coevaluation of mobile malware and anti-malware, which they generate the malware using GP and then they use GP to detect the malware. The features they used are API features and permissions for the anti-malware, which is different from us that only solely used permissions as features.

An anomaly approach:
This study demonstrates how GP can evolve programs to detect mobile phone threats such as malware. We took a risk by implementing an anomaly-based approach for our detectors since their FPR can be high with a low detection rate for known attacks [214]. However, we overcame it by using different weights for calculating the fitness function implementation. We showed that the detection system we developed could distinguish maliciously behaving Android applications.

Efficiency:
Our work explored trade-offs between functional and non-functional properties of programs. They showed how our approaches could synthesise programs with excellent trade-offs between intrusion detection capability, power consumption and execution time. Furthermore, we investigated whether SPEA2 gave the best trade-offs.

Mobile malware detector framework: Our framework is the first to combine GP and MOEA, to consider limited resources and to use only Android permissions as detection features. Energy consumption and execution time of programs are also taken into account.

Significant different datasets: In this thesis, we evaluate the performance of our proposed system on several enormous datasets. We sought four different datasets from Android community researchers and tested our proposed system on them. The datasets contain different types of Android attacks, including ransomware.

This thesis demonstrates that EC approaches can learn the complex properties of Android mobile phones and synthesise appropriate intrusion detection programs for this environment. The properties of Android mobile phones taken into consideration in this research are Android app permissions and power consumption and execution time of evolved programs. We used only Android permissions for features in our proposed system because it is the common thing in the Manifest.xml file in Android APK, which can be manipulated by the attackers. This has happened when the mobile phone users allow permissions to be used inside a mobile phone after they install apps. Power consumption and execution time are important non-functional properties.

7.3 Future Research

The undeveloped areas for future research are summarised below:

Applying evolutionary computation techniques to other areas: In this study, we demonstrated how to utilise EC approach to overcome the issue of intrusion detection in Android mobile phones and exactly how to explore different trade-offs in such resource-limited devices. The approaches examined in this thesis might be adapted easily to other areas, for instance, like the detection of malware in the iOS platform. As mentioned by Price, it is rare for an iPhone, or iPad to get infected by a virus, but it is still possible [215] as mobile platforms become a target of choice. Similar concerns about power consumption will apply.

Exploration of new attacks: Android mobile phones are still under attack. Attackers are changing their 'prey' from computers and laptops to mobile phones. There should be more research to identify the range of possible Android mobile phone attacks. The proposed system in this thesis can be used to explore new attacks that are not mentioned in this research.

Exploration of new attacks: Our work reported here targeted specific known families of malware that had been assembled by researchers. As new malware (e.g., the recently discovered Man-in-the Disk attack [216]) and variants are discovered, our approach can be redeployed on an enhanced dataset that covers these elements.

Improving our approach: In this study, we used only Android permissions as our main features selection to identify malicious behaviour APK from normal behaviour APK. It remains perfectly plausible to expand the range of attributes used, e.g. using different attributes such as api_call, intent, activity, URL and other information we can extract from the APK.

Off-line and online approaches: all work reported in this thesis is off-line. The potential for online collaborative and adaptive approaches is an important avenue to consider. This could again cover functional and non-functional properties.

In conclusion, AI approaches for program syntheses such as GP and MOEC can give significant benefits for the evolution of IDS programs for challenging complex environments such as Android mobile phones. We recommend this area to the research community.

REFERENCES

[1] N. Abdullah and N. Md.Salleh, “Permission-based Android Malware Detection System Using Genetic Programming,” *Diges PMU (Technology Innov. Int. Conf. Techon 2019*, vol. 6, no. October 2019, pp. 240–248, 2019.

[2] J. R. Koza and H. Iba, “Genetic programming : proceedings of the first annual conference, 1996,” *Proc. 1st Annu. Conf. Genet. Program.*, p. 568, 1996.

[3] T. Nakamura, “5G Evolution and 6G,” 2020.

[4] G. A. Fowler, “The 5G lie: The network of the future is still slow,” *The Washington Post*, 2020. [Online]. Available: https://www.washingtonpost.com/technology/2020/09/08/5g-speed/. [Accessed: 31-Oct-2020].

[5] S. Meena, “The Data Digest: Forrester Forecasts Single-Digit Growth For Global Smartphone Unique Subscribers For The First Time in 2018,” *Forrester*, 2018. [Online]. Available: https://go.forrester.com/blogs/the-data-digest-forrester-forecasts-single-digit-growth-for-global-smartphone-unique-subscribers-for-the-first-time-in-2018/. [Accessed: 27-Nov-2018].

[6] S. O’Dea, “Smartphone Users from 2016 to 2021,” *Statista*, 2020. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Accessed: 22-Nov-2020].

[7] IANS, “Smartphone users spend over 90 mins online daily,” *THe Economic Times*, 2018. .

[8] M. Boyle, “Mobile Internet Satistics,” *Finder UK*, 2020. [Online]. Available: https://www.finder.com/uk/mobile-internet-statistics#:~:text=Quick overview,up from 66%25 in 2018. [Accessed: 22-Nov-2020].

[9] Y. Wang, K. Streff, and S. Raman, “Smartphone security challenges,” *Computer*, vol. 45, pp. 52–58, 2012.

[10] P. Ruggiero and J. Foote, “Cyber Threats to Mobile Phones,” 2011.

[11] L. Wu, X. Du, and X. Fu, “Security threats to mobile multimedia applications:

Camera-based attacks on mobile phones," *IEEE Commun. Mag.*, vol. 52, no. March, pp. 80–87, 2014.

[12] Kaspersky Lab, "Top 7 Mobile Security Threats in 2020," 2020. [Online]. Available: https://usa.kaspersky.com/resource-center/threats/top-seven-mobile-security-threats-smart-phones-tablets-and-mobile-internet-devices-what-the-future-has-in-store. [Accessed: 22-Nov-2020].

[13] Kaspersky Lab, "Top 7 Mobile Security Threats in 2020," 2020. .

[14] U.-C. C. R. Team, "Technical Information Paper-TIP-10-105-01 Cyber Threats to Mobile Devices," 2010.

[15] D. Crăciunescu, "A Short History of Mobile Malware," 2020. [Online]. Available: https://proandroiddev.com/a-short-history-of-mobile-malware-296570ed5c1b. [Accessed: 10-Nov-2020].

[16] S. Keach, "GOOG-HELL Google 'eavesdropping on your PRIVATE conversations' with Android phones and smart speakers," *The Sun*, 2019. [Online]. Available: https://www.thesun.co.uk/tech/9487748/google-listen-private-conversations-android-phones-home-assistant-smart-speakers/. [Accessed: 20-Nov-2020].

[17] J. P. Anderson, "Computer security threat monitoring and surveillance," Washington, 1980.

[18] Á. MacDermott, Q. Shi, M. Merabti, and K. Kifayat, "Intrusion Detection for Critical Infrastructure Protection," *ISBN:9781902560267*, pp. 1–6, 2012.

[19] N. K. M and R. Kumar, "Survey on Network Based Intrusion Detection System in MANET," *Int. J. Comput. Sci. Mob. Comput.*, vol. 3, no. 4, pp. 660–663, 2014.

[20] T. Verwoerd and R. Hunt, "Intrusion detection techniques and approaches," *Comput. Commun.*, vol. 25, pp. 1356–1365, 2002.

[21] E. Biermann, E. Cloete, and L. M. Venter, "A Comparison of Intrusion Detection Systems," *Comput. Secur.*, vol. 20, pp. 676–683, 2001.

[22] D. E. Denning, "An Intrusion-Detection Model," in *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, 1986, pp. 118–131.

[23] S. Şen, J. a Clark, and J. E. Tapiador, "Power-Aware Intrusion Detection in

Mobile Ad Hoc Networks," in *Ad Hoc Networks*, Springer Berlin Heidelberg, 2010, pp. 224–239.

[24] E. Salimi and N. Arastouie, "Backdoor Detection System Using Artificial Neural Network and Genetic Algorithm," *Comput. Inf. Sci. Int. Conf.*, pp. 817–820, 2011.

[25] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Comput.*, pp. 1–15, 2014.

[26] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, no. 1–2, pp. 18–28, 2008.

[27] C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin, "Intrusion detection by machine learning: A review," *Expert Syst. Appl.*, vol. 36, no. 10, pp. 11994–12000, 2009.

[28] S. Sen and J. A. Clark, "Evolutionary computation techniques for intrusion detection in mobile ad hoc networks," *Comput. Networks*, vol. 55, no. 15, pp. 3441–3457, 2011.

[29] Statista, "Number of apps available in leading app stores as of 3rd quarter 2018," 2018.

[30] C. Nimodia and H. Deshmukh, "Android Operating System," *Softw. Eng. ISSN*, vol. 3, no. 1, pp. 10–13, 2012.

[31] Android OS Project, "Android (OS)," 2013.

[32] Google, "Protect every Android user," 2020. [Online]. Available: https://www.android.com/security-center/. [Accessed: 30-Oct-2020].

[33] T. Android and O. Source, "Android (OS)," 2008.

[34] M. Narmatha and S. V. Krishnakumar, "Study on Android Operating System And Its Versions," *Int. J. Sci. Eng. Appl. Sci. -*, vol. 2, no. 2, pp. 439–445, 2016.

[35] J. Callaham, "The history of Android: The evolution of the biggest mobile OS in the world," 2020. [Online]. Available: https://www.androidauthority.com/history-android-os-name-789433/. [Accessed: 22-Nov-2020].

[36] J. Callaham, "The history of Android OS: its name, origin and more," 2018. [Online]. Available: https://www.androidauthority.com/history-android-os-

name-789433/. [Accessed: 24-Oct-2018].

[37] Android, “Android,” 2018. [Online]. Available: www.android.com/history. [Accessed: 24-Oct-2018].

[38] Android Open Source Project, “Secure an Android Device,” 2020. [Online]. Available: https://source.android.com/security. [Accessed: 30-Oct-2020].

[39] J. Agar, *Constant Touch: A Global History of the Mobile Phone*, Second. Cambridge, UK: Icon Book Ltd, 2003.

[40] S. P. Hall and E. Anderson, “Operating systems for mobile computing,” *J. Comput. Sci. Coll.*, vol. 25, pp. 64–71, 2009.

[41] V. Beal, “The History of Microsoft Operating Systems,” *IT Business Edge*, 2012. [Online]. Available: http://www.webopedia.com/DidYouKnow/Hardware_Software/history_of_microsoft_windows_operating_system.html. [Accessed: 25-Oct-2018].

[42] H. Dwivedi, C. Clark, and D. Thiel, *Mobile Application Security*. McGraw Hill, 2010.

[43] K. W. Tracy, “History and Evolution of the Android OS,” *2011 Second Int. Conf. Innov. Bio-inspired Comput. Appl.*, no. November, pp. 1–8, 2011.

[44] Android, “Android - Security,” 2018. [Online]. Available: https://source.android.com/security/. [Accessed: 24-Oct-2018].

[45] Google. Inc, “Android Security 2017 Year In Review: March 2018,” no. March, 2018.

[46] Google. Inc, “Android Security - 2014 Year in Review,” 2014.

[47] Google. Inc, “Android Security - 2015 Year In Review,” no. April, 2016.

[48] Google. Inc, “Android Security - 2016 Year In Review,” no. March, 2017.

[49] A. W. Rufi, “Vulnerabilities, Threats, and Attacks,” in *Network Security 1 and 2 Companion Guide (Cisco Networking Academy)*, First Edit., Indianapolis, USA: Cisco Press, 2006, pp. 1–49.

[50] S. Axelsson, “Research in Intrusion-Detection Systems: A Survey,” 1999.

[51] T. Lunt, “A survey of intrusion detection techniques,” *Comput. Secur.*, vol. 12, pp. 405–418, 1993.

[52] D. E. Denning, "An Intrusion-Detection Model," *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 2, pp. 222–232, 1987.

[53] V. Chebyshev, "IT threat evolution Q2 2020. Mobile statistics," 2020. [Online]. Available: https://securelist.com/it-threat-evolution-q2-2020-mobile-statistics/98337/. [Accessed: 22-Nov-2020].

[54] V. Chebyshev, F. Sinitsyn, D. Parinov, A. Liskin, and O. Kupreev, "IT Threat Evolution Q1 2018. Statistics," 2018. [Online]. Available: https://securelist.com/it-threat-evolution-q1-2018-statistics/85541/. [Accessed: 29-Oct-2012].

[55] G. Davis and R. Samani, "Mobile Threat Report Q1," 2018.

[56] L. La Porta, "4 Ways Hackers are Infiltrating Phones with Malwares on Android Phones," 2018. [Online]. Available: https://www.wandera.com/malware-on-android/. [Accessed: 29-Oct-2018].

[57] J. Dhaliwal, "How to Detect and Remove a Virus from Your Android Phone," 2018. [Online]. Available: https://blog.avast.com/remove-android-virus. [Accessed: 29-Oct-2018].

[58] Cisco, "What Is the Difference: Viruses, Worms, Trojans, and Bots?," 2018. [Online]. Available: https://www.cisco.com/c/en/us/about/security-center/virus-differences.html. [Accessed: 14-Oct-2018].

[59] S. Allam, S. V. Flowerday, and E. Flowerday, "Smartphone information security awareness: A victim of operational pressures," *Comput. Secur.*, vol. 42, pp. 55–65, 2014.

[60] A. Mylonas, A. Kastania, and D. Gritzalis, "Delegate the smartphone user? Security awareness in smartphone platforms," *Comput. Secur.*, vol. 34, pp. 47–66, 2013.

[61] Tektonika Staff, "5 Mobile Threats You Should Shut Down in 2018," 2018. [Online]. Available: https://www.tektonikamag.com/index.php/2018/05/04/5-mobile-threats-you-should-shut-down-in-2018/. [Accessed: 29-Oct-2018].

[62] Y. R. Man, *Wireless Mobile Internet Security*, Second Edi. West Sussex, UK: John Wiley and Sons Ltd, 2013.

[63] RSA Security Inc, "Phishing, Vishing and Smishing : Old Threats Present New Risks," Ireland, 2009.

[64] W. Lee and B. Rotoloni, "Emerging Cyber Threats Report 2015," Atlanta, 2015.

[65] J. Patel and P. S. D. Panchal, "A survey on Pharming attack Detection and prevention methodology," *IOSR J. Comput. Eng.*, vol. 9, no. 1, pp. 66–72, 2013.

[66] J. B. D. Cabrera, C. Gutiérrez, and R. K. Mehra, "Ensemble methods for anomaly detection and distributed intrusion detection in Mobile Ad-Hoc Networks," *Inf. Fusion*, vol. 9, no. 1, pp. 96–119, 2008.

[67] D. Palmer, "Mobile malware attacks are booming in 2019: These are the most common threats," *ZDNet*, 2019. [Online]. Available: https://www.zdnet.com/article/mobile-malware-attacks-are-booming-in-2019-these-are-the-most-common-threats/. [Accessed: 26-Nov-2020].

[68] C. Cimpanu, "New WAPDropper malware abuses Android devices for WAP fraud," *ZDNet*, 2020. [Online]. Available: https://www.zdnet.com/article/new-wapdropper-malware-abuses-android-devices-for-wap-fraud/. [Accessed: 26-Nov-2020].

[69] C. Cimpanu, "New 'Ghimob' malware can spy on 153 Android mobile applications," *ZDNet*, 2020. [Online]. Available: https://www.zdnet.com/article/new-ghimob-malware-can-spy-on-153-android-mobile-applications/. [Accessed: 20-Nov-2020].

[70] D. Palmer, "Android malware returns and this time it will record what is on your screen, too," *ZDNet*, 2020. [Online]. Available: https://www.zdnet.com/article/android-malware-returns-and-this-time-it-will-record-what-is-on-your-screen-too/. [Accessed: 26-Nov-2020].

[71] L. Hautala, "4 signs your Android phone has hidden malware, and how to deal with it," *Cnet*, 2020. [Online]. Available: https://www.cnet.com/how-to/4-signs-your-android-phone-has-hidden-malware-and-how-to-deal-with-it/. [Accessed: 22-Nov-2020].

[72] National Cyber Security Centre, "The Cyber Threat to UK Business 2017 - 2018 Report," 2018.

[73] Check Point Software Technologies LTD, "When Ransomware Goes Mobile,"

2018. [Online]. Available: https://blog.checkpoint.com/2018/06/15/when-ransomware-goes-mobile/. [Accessed: 18-Oct-2018].

[74] S. McKinley, "Before We Solve the World's Problems, We Need to Connect It to the Internet," 2018. [Online]. Available: https://qz.com/1233010/before-we-solve-the-worlds-problems-we-need-to-connect-it-to-the-internet/. [Accessed: 30-Oct-2018].

[75] L. Dixon, "Internet Access a Necessity in the Talent Economy," 2018. [Online]. Available: https://www.clomedia.com/2017/04/10/internet-access-necessity-talent-economy/. [Accessed: 30-Oct-2018].

[76] L. Gordon and P. Marceux, "Has Internet Access Become a Basic Human Right?," 2017. [Online]. Available: https://blog.euromonitor.com/internet-access-basic-human-right/. [Accessed: 30-Oct-2018].

[77] R. A. Kemmerer and G. Vigna, "Intrusion detection : A Brief History and Overview," *Secur. Priv.*, vol. 35, no. 4, pp. 27–30, 2002.

[78] C. Fung and R. Boutaba, *Intrusion Detection Networks A Key to Collaborative Security*. Boca Raton, USA: CRC Press, 2014.

[79] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson, "A Methodology for Testing Intrusion Detection Systems 1," *IEEE Trans. Softw. Eng.*, vol. 22, no. October, pp. 719–729, 1996.

[80] P. Uppuluri and R. Sekar, "Experiences with Specification Based Intrusion Detection," in *Fourth International Symposium on Recent Advances in Intrusion Detection*, W. Lee, L. Mé, and A. Wespi, Eds. CA, USA: Springer-Verlag Berlin Heidelberg, 2001, pp. 172–189.

[81] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[82] A. Patel, M. Taghavi, K. Bakhtiyari, and J. Celestino Júnior, "An intrusion detection and prevention system in cloud computing: A systematic review," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 25–41, Jan. 2013.

[83] S. Sevil and J. A. Clark, "Intrusion Detection in Mobile Ad Hoc Networks," in *Computer Communications and Networks*, S. Misra, I. Woungang, and S. Chandra Misra, Eds. London: Springer London, 2009, pp. 427–454.

[84] J. McHugh, A. Christie, and J. Allen, “Defending Yourself: The role of Intrusion Detection Systems,” *IEEE Softw.*, no. October, pp. 42–51, 2000.

[85] C. Ko, M. Ruschitzka, and K. Levitt, “Execution Monitoring of Security-Critical Programs in Distributed Systems: A specification-based Approach,” *Proceedings. 1997 IEEE Symp. Secur. Priv. (Cat. No.97CB36097)*, pp. 175–187, 1997.

[86] S. Sen, “A Survey of Intrusion Detection using Evolutionary Computation,” 2015, p. 24.

[87] D. Anderson, T. Frivold, and A. Valdes, “Next-generation Intrusion Detection Expert System (NIDES): A summary,” Menlo Park, 1995.

[88] M. M. Sebring, E. W. Shellhouse, M. E. Hanna, and R. A. Whitehurst, “Expert systems in intrusion detection: A case study,” in *Proceedings of the 11th …*, 1988, no. October, pp. 74–81.

[89] K. Ilgun, “USTAT: A Real-time Intrusion Detection System for UNIX,” in *IEEE Symposium on Security and Privacy,* 1993, pp. 16–28.

[90] S. E. Smaha, “Haystack: An Intrusion Detection System,” *IEEE*, pp. 37–44, 1988.

[91] P. a Porras and P. G. Neumann, “EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances,” in *Proc. 20th NIST-NCSC National Information Systems Security Conference*, 1997, pp. 353–365.

[92] P. G. Neumann and P. a Porras, “Experience with EMERALD to date,” *Proc. 1st Conf. Work. Intrusion Detect. Netw. Monit. - Vol. 1*, p. 8, 1999.

[93] S. A. Zonouz, K. R. Joshi, and W. H. Sanders, “Cost-aware systemwide intrusion defense via online forensics and on-demand IDS deployment,” *SafeConfig*, vol. 6894 LNCS, pp. 71–74, 2010.

[94] S. A. Zonouz, K. R. Joshi, and W. H. Sanders, “FloGuard: Cost-aware systemwide intrusion defense via online forensics and on-demand IDS deployment,” *Comput. Safety, Reliab. Secur.*, vol. 6894 LNCS, pp. 338–354, 2011.

[95] D. M. Farid and M. Z. Rahman, “Anomaly network intrusion detection based on improved self adaptive Bayesian algorithm,” *J. Comput.*, vol. 5, no. 1, pp. 23–31, 2010.

[96] J. D. Cannady, “Artificial neural networks for misuse detection,” *Proc. Natl. Inf.*

Syst. Secur. Conf., pp. 368–381, 1998.

[97] C. Sinclair, L. Pierce, and S. Matzner, "An application of machine learning to network intrusion detection," *Proc. Comput. Secur. Appl. Conf. 1999.(ACSAC'99) . 15th Annu.*, no. 0293, pp. 371–377, 1999.

[98] N. B. Idris and B. Shanmugam, "Artificial Intelligence Techniques Applied to Intrusion Detection," *2005 Annu. IEEE India Conf. - Indicon*, pp. 52–55, 2005.

[99] T. L. Heberlein, V. G. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," *IEEE*, pp. 296–304, 1990.

[100] M. Roesch, "Snort: Lightweight Intrusion Detection for Networks.," *LISA '99 13th Syst. Adm. Conf.*, pp. 229–238, 1999.

[101] J. Frank, "Artificial Intelligence and Intrusion Detection : Current and Future Directions," *Proc. 17th Natl. Comput. Secur. Conf.*, vol. 10, pp. 1–12, 1994.

[102] M. Manninen, "Using Artificial Intelligence in Intrusion Detection Systems," *Helsinki Univ. Technol.*, vol. 13, p. 6, 2007.

[103] A. A. Adigun, T. M. Fagbola, and A. Adegun, "SwarmDroid : Swarm Optimized Intrusion Detection System for the Android Mobile Enterprise," *Int.Jour. Com. Sci. Issues*, vol. 11, no. 3, pp. 62–69, 2014.

[104] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "AntiMalDroid: An efficient SVM-based malware detection framework for android," *Commun. Comput. Inf. Sci.*, vol. 243 CCIS, no. PART 1, pp. 158–166, 2011.

[105] A. S. Shamili, C. Bauckhage, and T. Alpcan, "Malware Detection on Mobile Devices Using Distributed Machine Learning," *2010 20th Int. Conf. Pattern Recognit.*, pp. 4356–4359, 2010.

[106] D. Stiawan, A. H. Abdullah, and M. Y. Idris, "The Trends of Intrusion Prevention System Network," *ICETC 2010 - 2010 2nd Int. Conf. Educ. Technol. Comput.*, vol. 4, pp. 217–221, 2010.

[107] M. Campbell, "Phone invaders," *New Scientist*, vol. 223, no. 2977, pp. 32–35, Jul-2014.

[108] R. Riasat, M. Sakeena, C. Wang, A. H. Sadiq, and Y. Wang, "A Survey on Android Malware Detection Techniques," *DEStech Trans. Comput. Sci. Eng.*, no.

wcne, 2017.

[109] Y. Salah, I. Hamed, S. Nabil, A. Abdulkader, and M. M. Mostafa, "Mobile Malware Detection : A Survey," *Int. J. Comput. Sci. Inf. Secur.*, vol. 17, no. 1, 2019.

[110] J. Cheng, S. H. Y. Wong, H. Yang, and S. Lu, "SmartSiren: Virus Detection and Alert for Smartphones," *Proc. 5th Int. Conf. Mob. Syst. Appl. Serv. - MobiSys '07*, p. 258, 2007.

[111] H. Li, D. Ma, N. Saxena, B. Shrestha, and Y. Zhu, "Tap-Wave-Rub: Lightweight Malware Prevention for Smartphones Using Intuitive Human Gestures," *ACM Conf. Secur. Priv. Wirel. Mob. Networks WISEC'13*, pp. 25–30, 2013.

[112] A. Salman, I. H. Elhajj, A. Chehab, and A. Kayssi, "DAIDS : An Architecture for Modular Mobile IDS," *28th Int. Conf. Adv. Inf. Netw. Appl. Work.*, pp. 328–333, 2014.

[113] A. Fitriah, A. Kadir, N. Stakhanova, and A. A. Ghorbani, "Android Botnets: What URLs are Telling Us," vol. 9408, pp. 78–91, 2015.

[114] A. Houmansadr, S. a. Zonouz, and R. Berthier, "A cloud-based intrusion detection and response system for mobile phones," *2011 IEEE/IFIP 41st Int. Conf. Dependable Syst. Networks Work.*, pp. 31–32, 2011.

[115] S. Zonouz, A. Houmansadra, R. Berthiera, N. Borisova, and W. Sanders, "Secloud: A cloud-based comprehensive and lightweight security solution for smartphones," *Comput. Secur.*, vol. 37, pp. 215–227, 2013.

[116] M. Zheng, M. Sun, and J. C. S. Lui, "DroidAnalytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware," *2013 12th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun.*, pp. 163–171, 2013.

[117] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," *Proc. 1st ACM Work. Secur. Priv. smartphones Mob. devices - SPSM '11*, p. 15, 2011.

[118] A. Chaugule, Z. Xu, and S. Zhu, "A Specification Based Intrusion Detection Framework for Mobile Phones," *Proceeding 9th Int. Conf. Appl. Cryptogr. Netw. Secur. June 07-10,2011, Nerja, Spain*, pp. 19–37, 2011.

[119] D. Arp, M. Spreitzenbarth, H. Malte, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," *Symp. Net. Dist.*

Sys. Sec., pp. 23–26, 2014.

[120] F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, “Ransomware Steals Your Phone. Formal Methods Rescue It,” in *International Federation for Information Processing*, vol. 1, 2016, pp. 212–221.

[121] H. Han, R. Li, and X. Gu, “Identifying malicious Android apps using permissions and system events,” vol. 8, no. 1, pp. 46–58, 2016.

[122] R. Kumar, Z. Xiaosong, R. U. Khan, J. Kumar, and I. Ahad, “Effective and Explainable Detection of Android Malware Based on Machine Learning Algorithms,” *Proc. 2018 Int. Conf. Comput. Artif. Intell. - ICCAI 2018*, pp. 35–40, 2018.

[123] N. Liu, M. Yang, and S. Zhang, “Detecting Applications with Malicious Behavior in Android Device Based on GA and SVM,” vol. 140, no. Ecae 2017, pp. 257–261, 2018.

[124] A. Boukerche, A. Notare, and S. M. Moretti, “Neural Fraud Detection in Mobile Phone Operations,” *Lect. Notes Comput. Sci. Distrib. Process.*, vol. 1800, pp. 636–644, 2000.

[125] M. Halilovic and A. Subasi, “Intrusion Detection on Smartphones,” *arXiv eprint arXiv:1211.6610*, 2012.

[126] B. Sanz, I. Santos, J. Nieves, C. Laorden, I. Alonso-Gonzalez, and P. G. Bringas, “MADS: Malicious Android applications detection through string analysis,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7873 LNCS, pp. 178–191, 2013.

[127] A. Martín, H. D. Menéndez, and D. Camacho, “MOCDroid: multi-objective evolutionary classifier for Android malware detection,” *Soft Comput.*, vol. 21, no. 24, pp. 7405–7415, 2017.

[128] S. Sen, E. Aydogan, and A. I. Aysan, “Coevolution of Mobile Malware and Anti-Malware,” *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 10, pp. 2563–2574, 2018.

[129] A. Patcha and J. M. Park, “An overview of anomaly detection techniques: Existing solutions and latest technological trends,” *Comput. Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[130] R. Sekar *et al.*, “Specification-based anomaly detection: a new approach for

detecting network intrusions," in *Proceedings of the Ninth ACM Conference on Computer and Communication Security*, 2002, pp. 265–274.

[131] P. Mukesh Krishnan, M.B Sheik Abdul Khader, "Estimating Detection Trust Hold For Intrusion Detection Systems in Mobile Ad hoc Network: A Comprehensive Study," *J. Comput. Appl. Res. Dev.*, vol. 1, no. 1, pp. 1–7, 2011.

[132] H. KhorasaniZadeh, Z. Mohamed Sidek, and J.-L. Ab Manan, "An Overview on Intrusion Detection Systems," Kuala Lumpur, 2013.

[133] M. Sechi, M. Annoni, A. Boukerche, and F. Augusto, "An Intrusion Detection System to Mobile Phone Networks," in *EXPO 2000*, 2000, pp. 362–370.

[134] D. S. Michalopoulos and N. L. Clarke, "Intrusion Detection System for mobile devices," in *Advances in Networks, Computing and Communications*, 2000, pp. 205–212.

[135] L. O. Babatope, L. Babatunde, and I. Ayobami, "Strategic Sensor Placement for Intrusion Detection in Network-Based IDS," *Int. J. Intell. Syst. Appl.*, vol. 6, no. 2, pp. 61–68, 2014.

[136] H. Chen, J. A. Clark, S. A. Shaikh, H. Chivers, and P. Nobles, "Optimising IDS Sensor Placement," *ARES 2010 - 5th Int. Conf. Availability, Reliab. Secur.*, pp. 315–320, 2010.

[137] N. Vallina-Rodriguez and J. Crowcroft, "ErdOS: Achieving Energy Savings in Mobile OS," in *Proceedings of the sixth international workshop on MobiArch - MobiArch '11*, 2011, pp. 37–42.

[138] M. Alam and S. T. Vuong, "An Intelligent Multi-Agent Based Detection Malware," pp. 226–237, 2014.

[139] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary Computation Basic Algorithms and Operators*, First. Taylor & Francis, 2000.

[140] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. 2003.

[141] K. A. De Jong, "Evolutionary Computation," in *Evolutionary Computation: A Unified Approach*, Cambridge, UK: A Bradford Book, 2006, p. 272.

[142] C. Reeves, "Genetic algorithms," *Handb. metaheuristics*, pp. 109–140, 2010.

[143] M. Mitchell, "An Introduction to Genetic Algorithms (Complex Adaptive

Systems)," *MIT Press*, p. 221, 1998.

[144] J. A. Clark, "Evolutionary Computation (EVO) Genetic Algorithms (L3)." 2016.

[145] S. Marsland, *Machine Learning: An Algorithmic Perspective*, First. Boca Raton, USA: Chapman & Hall/CRC, 2009.

[146] H. Crc and S. Marsland, *MACHINE Learning - an algorithmetic view 2nd Edition*. 2015.

[147] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*, no. March. 2008.

[148] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial Perturbations Against Deep Neural Networks for Malware Classification," 2016.

[149] M. Dimjasevic, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Android Malware Detection Based on System Calls," *UUCS-15-003*, vol. 11, no. 1, pp. 209–216, 2015.

[150] P. Irolla and A. Dey, "The duplication issue within the Drebin dataset," *J. Comput. Virol. Hacking Tech.*, pp. 1–5, 2018.

[151] Android, "Developer Guide Manifest," 2018. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro. [Accessed: 03-May-2018].

[152] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," *2012 IEEE Symp. Secur. Priv.*, no. 4, pp. 95–109, 2012.

[153] Oracle, "Oracle Virtual Oracle VM VirtualBox." 2016.

[154] V. Pandey, "APK Studio." 2015.

[155] K. Dunham, S. Hartman, J. A. Morales, M. Quintans, and T. Strazzere, *Android malware and analysis*, First. Boca Raton, USA: CRC Press, 2014.

[156] VirusTotal, "VirusTotal." [Online]. Available: https://www.virustotal.com/. [Accessed: 10-Jan-2016].

[157] Android, "Developer Guides Permissions," 2018. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview. [Accessed: 25-Sep-2018].

[158] S. Luke *et al.*, "ECJ," *The ECJ Owner's Manual ver. 23*, 2015. [Online]. Available: https://cs.gmu.edu/~eclab/projects/ecj. [Accessed: 14-Sep-2015].

[159] N. Salkind, "Encyclopedia of Research Design." Thousand Oaks, California, 2010.

[160] W. B. Langdon, *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, First Edit. Springer Science & Business Media, 1998.

[161] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," in *USENIX Annual Technical Conference*, 2010, p. 14.

[162] D. Li, S. Hao, J. Gui, and W. G. J. Halfond, "An Empirical Study of the Energy Consumption of Android Applications," in *IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 121–130.

[163] Y. Zhu, M. Halpern, and V. J. Reddi, "Event-based scheduling for energy-efficient QoS (eQoS) in mobile Web applications," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015*, 2015, pp. 137–149.

[164] C. Bernal-Cárdenas, "Improving energy consumption in Android Apps," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 1048–1050.

[165] N. Li, N. Zhang, S. K. Das, and B. Thuraisingham, "Privacy preservation in wireless sensor networks: A state-of-the-art survey," *Ad Hoc Networks*, vol. 7, no. 8, pp. 1501–1514, Nov. 2009.

[166] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Proceedings - International Conference on Software Engineering*, 2013, pp. 92–101.

[167] S. Zhidkov, A. Sychev, A. Zhidkov, and Alexander Petrov, "On Smartphone Power Consumption in Acoustic Environment Monitoring Applications," *Appl. Syst. Innov.*, vol. 1, no. 1, p. 8, 2018.

[168] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating Android applications' CPU energy usage via bytecode profiling," in *2012 1st International Workshop on Green and Sustainable Software, GREENS 2012 - Proceedings*, 2012, pp.

1–7.

[169] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, “Calculating Source Line Level Energy Information for Android Applications,” in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013, pp. 78–89.

[170] M. Bokhari and M. Wagner, “Optimising Energy Consumption Heuristically on Android Mobile Phones,” in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO ’16 Companion*, 2016, pp. 1139–1140.

[171] M. A. Bokhari, B. R. Bruce, B. Alexander, and M. Wagner, “Deep Parameter Optimisation on Android Smartphones for Energy Minimisation-a tale of Woe and a Proof-of-Concept,” in *GECCO 2017 - Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2017, pp. 1–8.

[172] I. Polakis, M. Diamantaris, T. Petsas, F. Maggi, and S. Ioannidis, “Powerslave: Analyzing the energy consumption of mobile antivirus software,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015, vol. 9148, pp. 165–184.

[173] H. Yang and R. Tang, “Power Consumption Based Android Malware Detection,” *J. Electr. Comput. Eng.*, vol. 2016, pp. 1–7, 2016.

[174] J. Bickford, F. Park, A. Varshavsky, and F. Park, “Security versus Energy Tradeoffs in Host-Based Mobile Malware Detection,” *MobiSys’11 Proc. 9th Int. Conf. Mob. Syst. Appl. Serv.*, no. July, 2011.

[175] K. Deb, “Multi-Objective Optimisation,” in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Springer Science Business Media,Inc, 2005, pp. 273–316.

[176] S. Şen, “Evolutionary Computation Techniques for Intrusion Detection in Mobile Ad Hoc Networks,” University of York, 2010.

[177] Samadar Salim Majeed, “Multi-Objective Optimization,” 2014. [Online]. Available: https://www.slideshare.net/SEMEDARSALIM/multi-objective-optimization. [Accessed: 12-Nov-2018].

[178] E. Zitzler, K. Deb, and L. Thiele, “Comparison of Multiobjective Evolutionary Algorithms: Empirical Results,” *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, 2000.

[179] Samadar Salim Majeed, “Multi-Objective Optimization,” 2014. .

[180] C. M. Fonseca and P. J. Fleming, “Genetic algorithms for multiobjective optimization: formulation, discussion and generalization,” in *Fifth International Conference on Genetic Algorithms*, 1993, vol. 93, no. July, pp. 416–423.

[181] E. Zitzler, “Evolutionary Algorithms for Multiobjective Optimization : Methods and Applications,” 1999.

[182] E. Zitzler, “Evolutionary Algorithms for Multiobjective Optimization : Methods and Applications,” *Berichte aus der Inform. Shak. Verlag, Aachen-*, vol. no13398, no. 30, p. 120, 1999.

[183] E. Zitzler, K. Deb, and L. Thiele, “Comparison of Multiobjective Evolutionary Algorithm: Empirical Results,” 1999.

[184] E. Zitzler, M. Laumanns, and L. Thiele, “SPEA2 : Improving the Strength Pareto Evolutionary Algorithm,” 2001.

[185] J. L. Guardado, J. Torres, S. Maximov, and E. Melgoza, “An Encoding Technique for Multiobjective Evolutionary Algorithms Applied to Power Distribution System Reconfiguration,” vol. 2014, 2014.

[186] R. T. F. A. King, K. Deb, and H. C. S. Rughooputh, “Comparison of NSGA-II and SPEA2 on the Multiobjective Environmental / Economic Dispatch,” vol. 16, pp. 485–511, 2010.

[187] K. Liu, G. Pinto, and Y. D. Liu, “Data-oriented characterization of application-level energy optimization,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015, vol. 9033, pp. 316–331.

[188] Tutorials Point, “Java.lang.System.nanoTime() Method,” 2015. [Online]. Available: https://www.tutorialspoint.com/java/lang/system_nanotime.htm. [Accessed: 20-Jul-2015].

[189] A. Purohit, N. S. Choudhari, and A. Tiwari, “Code Bloat Problem in Genetic Programming,” *Int. J. Sci. Res. Publ.*, vol. 3, no. 4, p. 1612, 2013.

[190] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler, “Multiobjective genetic programming: Reducing bloat using SPEA2,” in *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, 2001, vol. 1, pp. 536–543.

[191] I. Grutze, “The New Mobile Threat Landscape, circa 2017 to 2018,” 2018.

[Online]. Available: https://blog.trendmicro.com/the-new-mobile-threat-landscape-circa-2017-to-2018/. [Accessed: 09-Apr-2018].

[192] S. Collet, "Five new threats to your mobile security," 2017. [Online]. Available: https://www.csoonline.com/article/2157785/data-protection/five-new-threats-to-your-mobile-security.html. [Accessed: 09-Oct-2018].

[193] R. Samani, "McAfee Mobile Threat Report Mobile Malware Is Playing Hide and Steal," 2020.

[194] A. Karim, "On the Analysis and Detection of Mobile Botnet," *J. Univers. Comput. Sci.*, vol. 22, no. 4, pp. 567–588, 2016.

[195] C. Boyd, "SamSam ransomware: what you need to know," 2018. [Online]. Available: https://blog.malwarebytes.com/cybercrime/2018/05/samsam-ransomware-need-know. [Accessed: 14-Oct-2018].

[196] Sarah, "Spotlight on ransomware: Ransomware payment methods," 2017. [Online]. Available: https://blog.emsisoft.com/en/28256/ransomware-payment-methods. [Accessed: 15-Oct-2018].

[197] S. Sjouwerman, "The Evolution of Mobile Ransomware," 2018. [Online]. Available: https://blog.knowbe4.com/evolution-of-mobile-ransomeware. [Accessed: 15-Oct-2018].

[198] McAfee, "Virus Profile: Android/FakeDoc.A," 2013. [Online]. Available: https://home.mcafee.com/virusinfo/virusprofile.aspx?key=2290837. [Accessed: 20-Oct-2018].

[199] McAfee, "Virus Profile: Android/Imlog.A," 2011. [Online]. Available: https://home.mcafee.com/VirusInfo/VirusProfile.aspx?key=731165. [Accessed: 20-Oct-2018].

[200] F-Secure, "Mobile Threat Report Q1 2012," 2012.

[201] Symantec Corporation, "Android.Lotoor," 2012. [Online]. Available: https://www.symantec.com/security-center/writeup/2012-091922-4449-99. [Accessed: 20-Oct-2018].

[202] Symantec Corporation, "Android.Mobiletx," 2012. [Online]. Available: https://www.symantec.com/security-center/writeup/2012-052807-4439-99. [Accessed: 20-Oct-2018].

[203] F-Secure, “Mobile Threat Report Q4 2012,” 2012.

[204] FortiGuard Labs, “Android/SendPay.G!tr,” 2012. [Online]. Available: https://fortiguard.com/encyclopedia/virus/3682676/android-sendpay-g-tr. [Accessed: 20-Oct-2018].

[205] Huawei, “2013 Botnets and DDoS Attacks Report,” 2013.

[206] C. Wueest, “The continued rise of DDoS attacks,” pp. 1–31, 2014.

[207] FortiGuard Labs, “Android/Pletor.A!tr,” 2014. [Online]. Available: https://fortiguard.com/encyclopedia/virus/6235399. [Accessed: 20-Oct-2018].

[208] B. Krebs, “Android Botnet Targets Middle East Banks,” 2014. [Online]. Available: https://krebsonsecurity.com/tag/sandroid/. [Accessed: 20-Oct-2018].

[209] FortiGuard Labs, “Android/Wroba.AP!tr,” 2014. [Online]. Available: https://www.fortiguard.com/encyclopedia/virus/6495604. [Accessed: 20-Oct-2018].

[210] A. Karim, R. Salleh, and S. A. A. Shah, “DeDroid: A Mobile Botnet Detection Approach Based on Static Analysis,” *2015 IEEE 12th Intl Conf Ubiquitous Intell. Comput. 2015 IEEE 12th Intl Conf Auton. Trust. Comput. 2015 IEEE 15th Intl Conf Scalable Comput. Commun. Its Assoc. Work.*, pp. 1327–1332, 2015.

[211] A. J. Alzahrani and A. a Ghorbani, “SMS Mobile Botnet Detection Using A Multi-Agent System : Research in Progress Categories and Subject Descriptors,” 2014.

[212] M. Parkour, “Contagio Mobile,” 2016. [Online]. Available: http://contagiominidump.blogspot.it/. [Accessed: 15-Nov-2016].

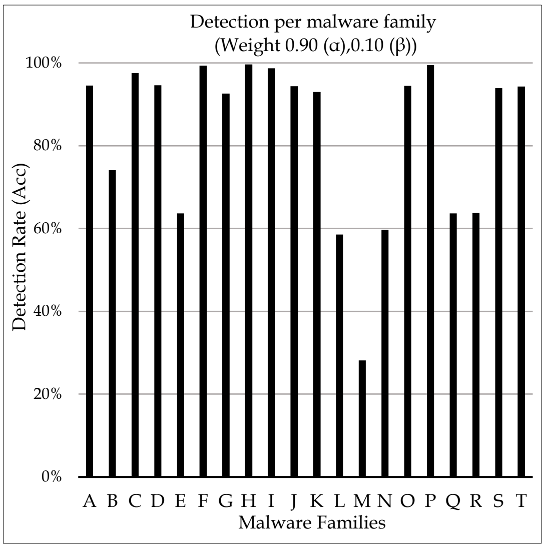
[213] “Ransomware Samples,” 2016. [Online]. Available: http://ransom.mobi/. [Accessed: 15-Nov-2016].

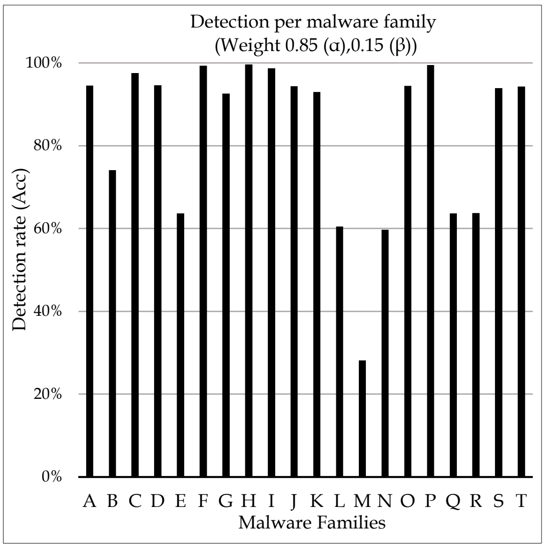
[214] L. H. Yeo, X. Che, and S. Lakkaraju, “Understanding Modern Intrusion Detection Systems : A survey,” 2017.

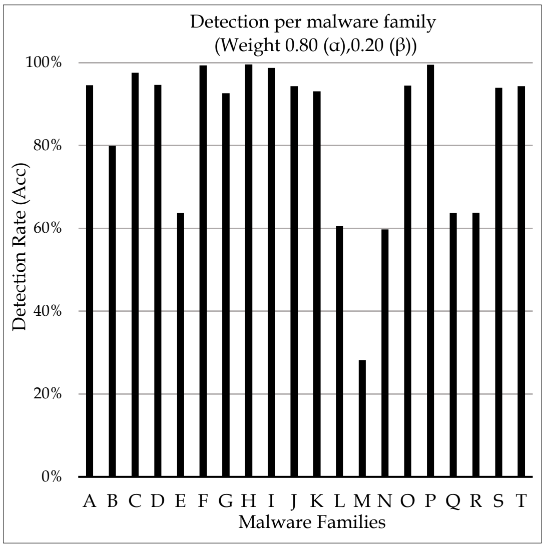
[215] D. Price, “How to remove a virus from an iPhone or iPad,” 2018. [Online]. Available: https://www.macworld.co.uk/how-to/iphone/remove-virus-iphone-ipad-3658975/. [Accessed: 07-Nov-2018].

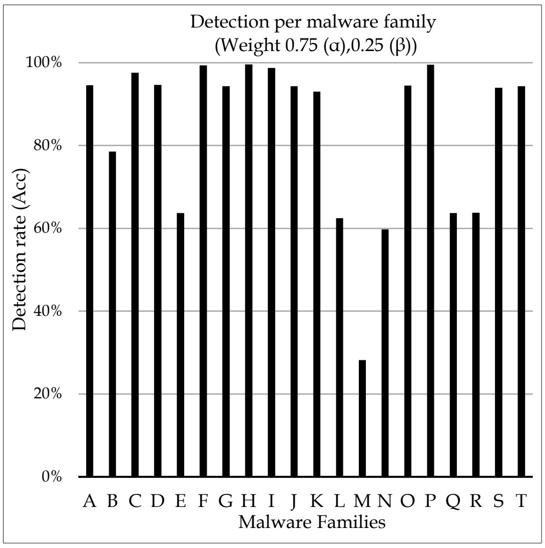
[216] S. Khandelwal, "New Man-in-the-Disk attack leaves millions of Android phones vulnerable," 2018. [Online]. Available: https://thehackernews.com/2018/08/man-in-the-disk-android-hack.html. [Accessed: 07-Nov-2018].

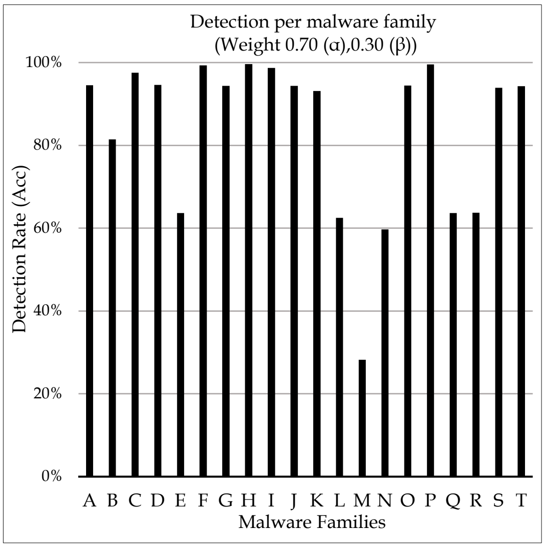
APPENDIX 1 –OTHER RESULTS FOR USING OPTIMAL PARAMETER CHOICE

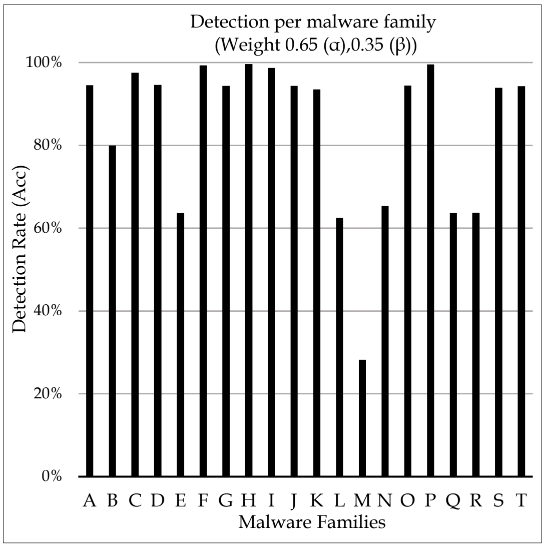




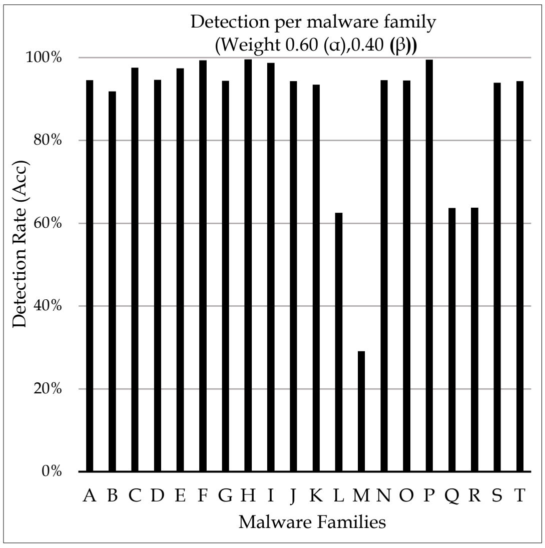


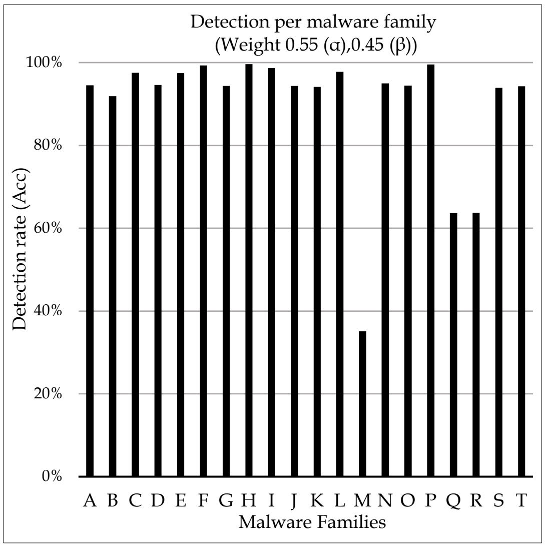


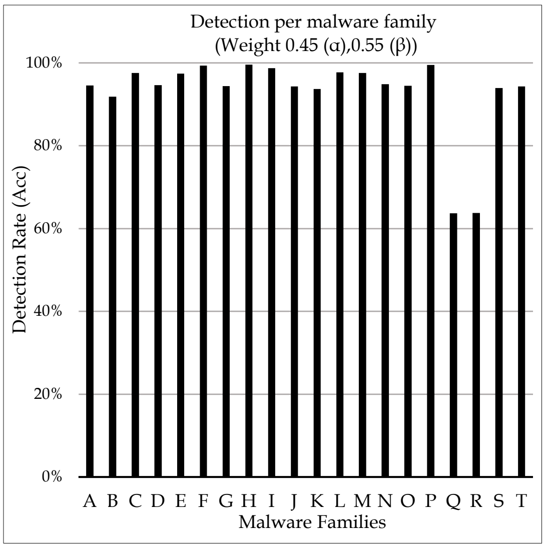


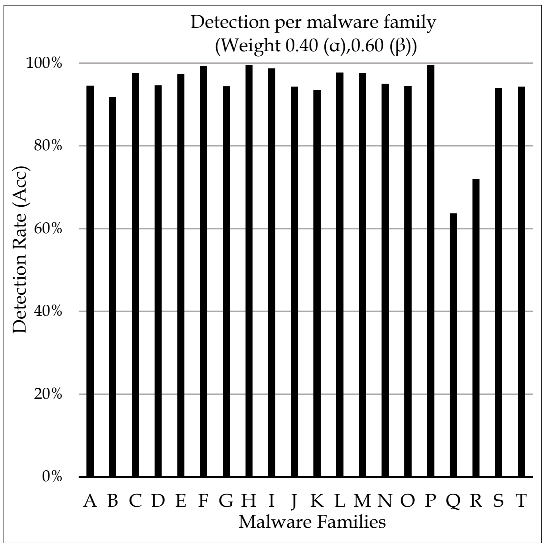


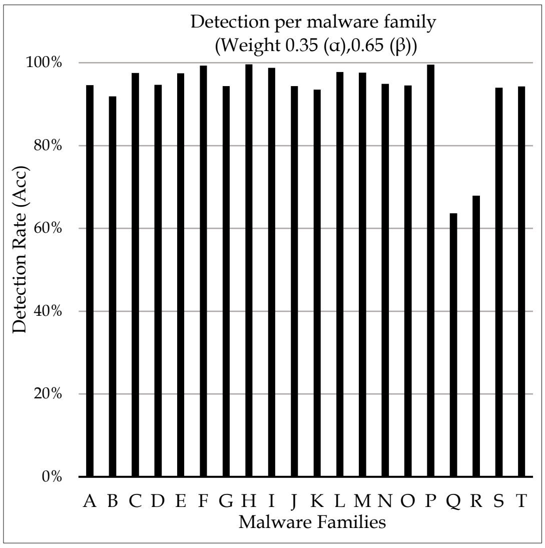
APPENDIX 1 - OTHER RESULTS FOR USING OPTIMAL PARAMETER CHOICE

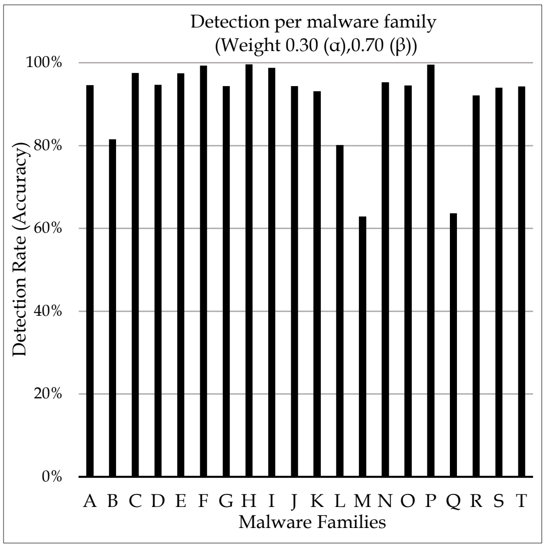




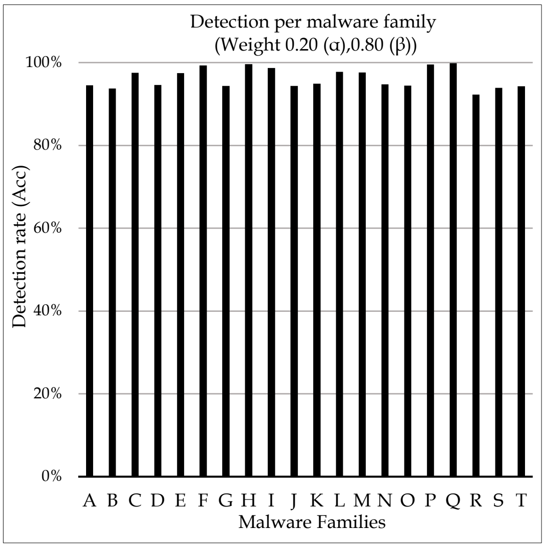


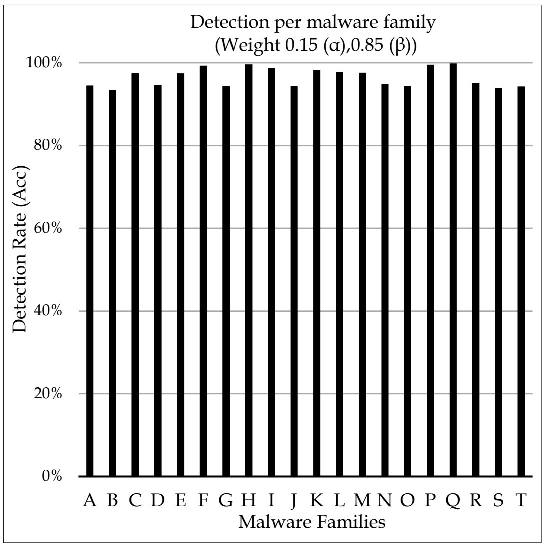


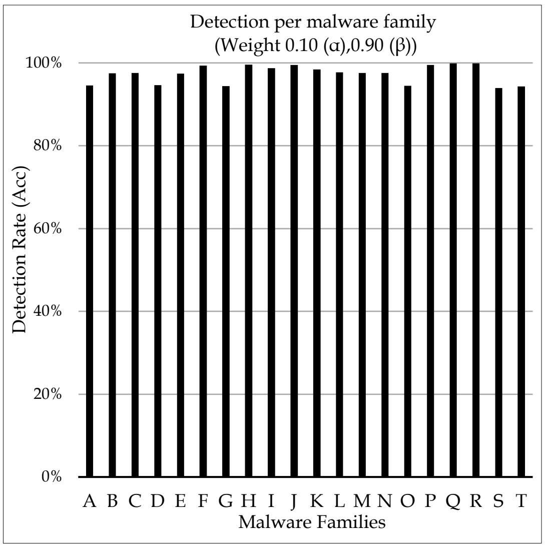




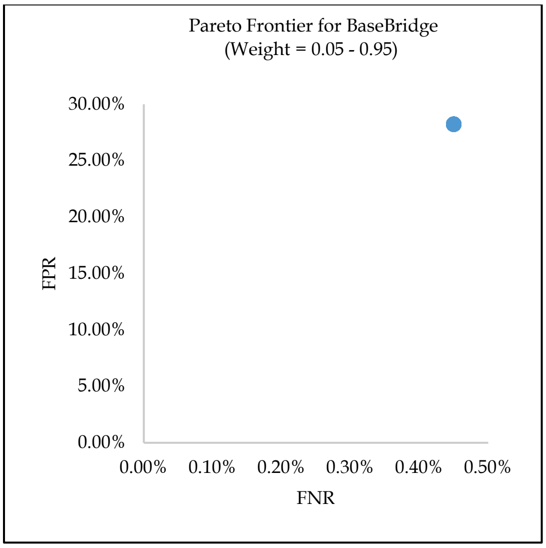
APPENDIX 1 - OTHER RESULTS FOR USING OPTIMAL PARAMETER CHOICE

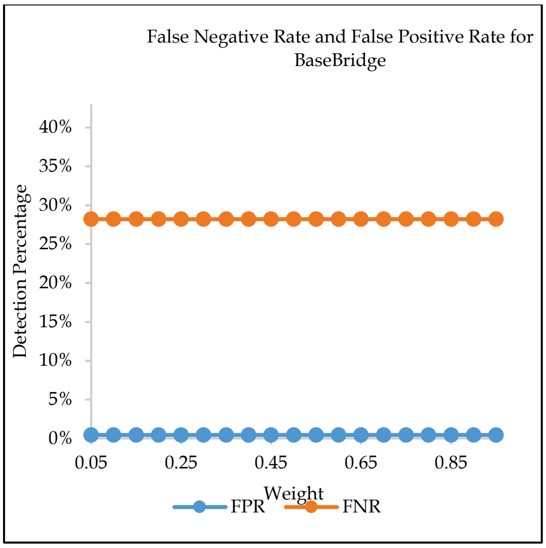


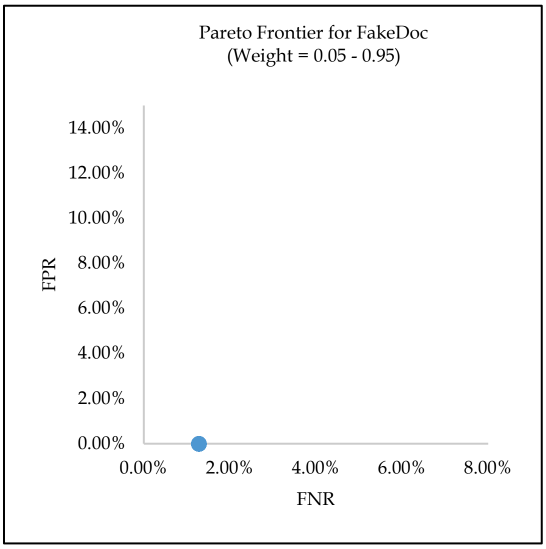


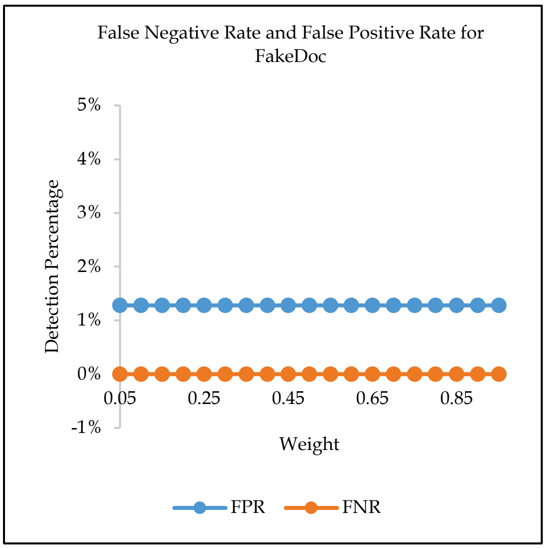


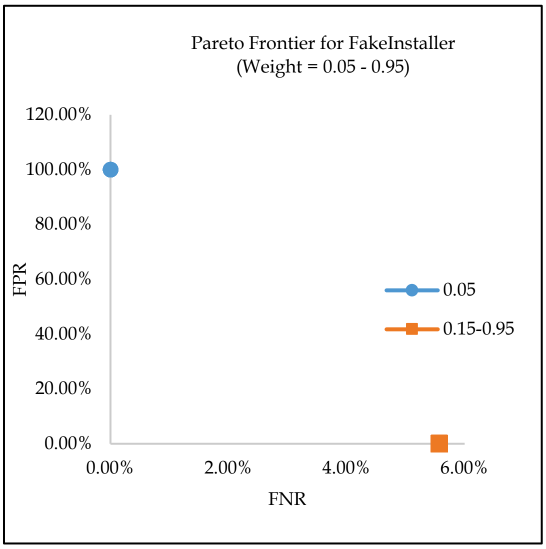
APPENDIX 2 – OTHER RESULTS FOR PARETO FRONTIER

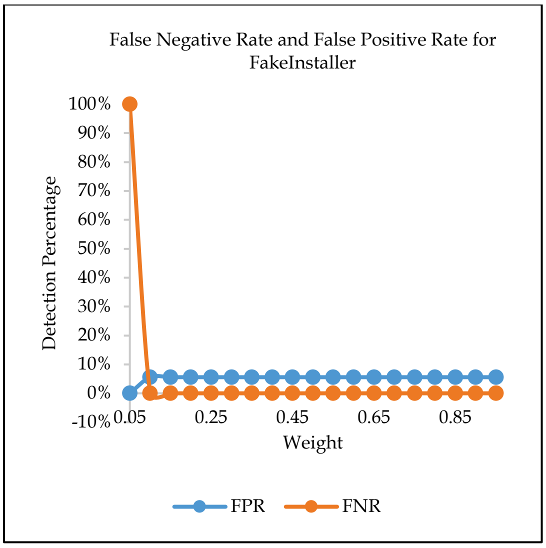




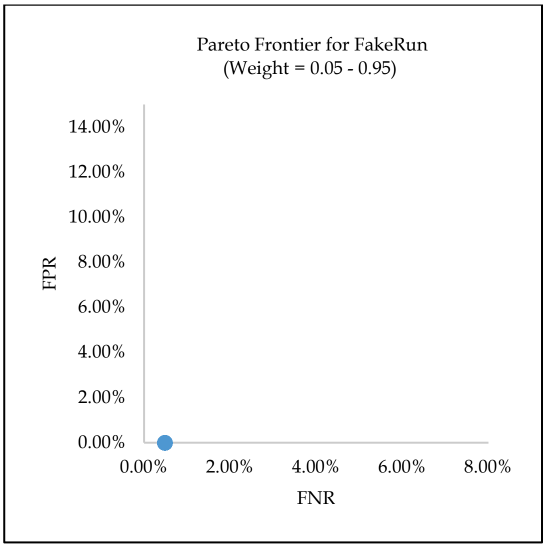


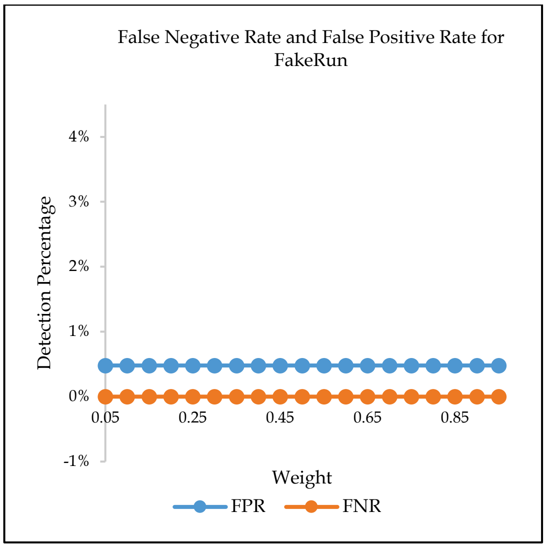


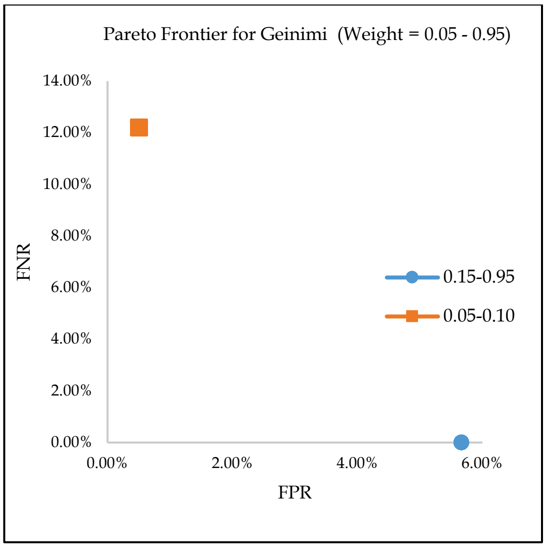


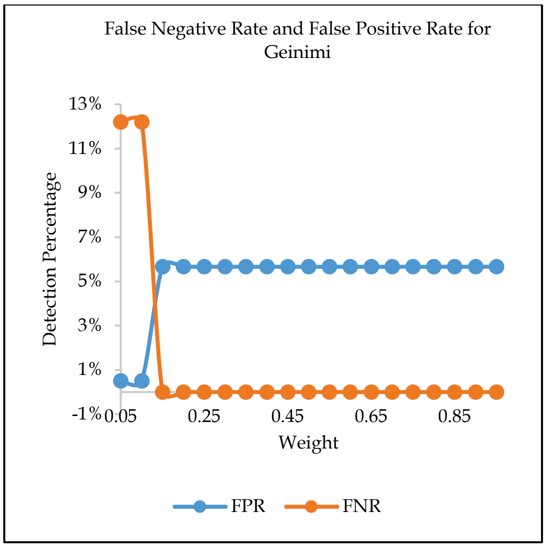


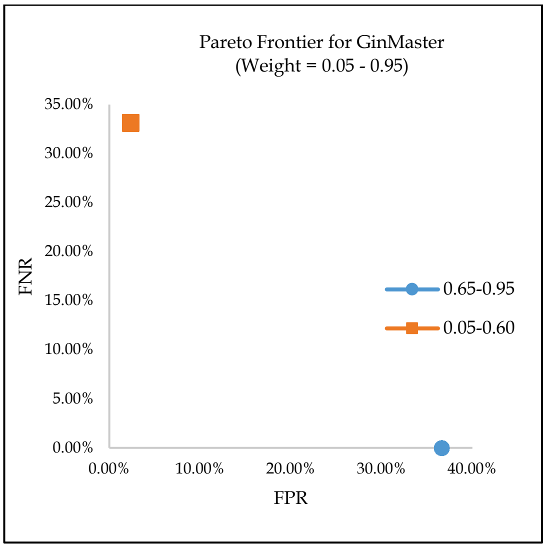
APPENDIX 2 - OTHER RESULTS FOR PARETO FRONTIER

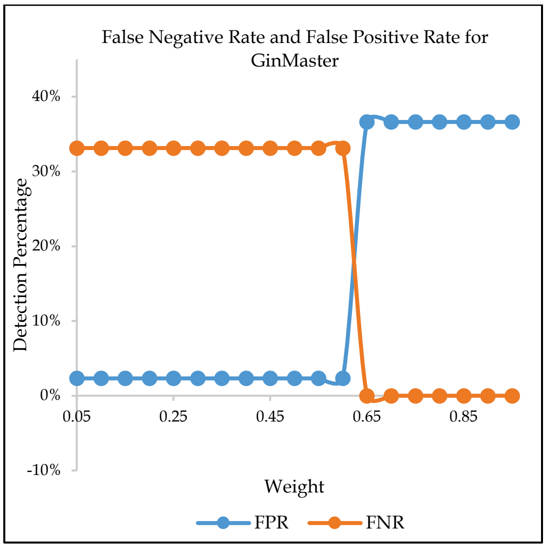




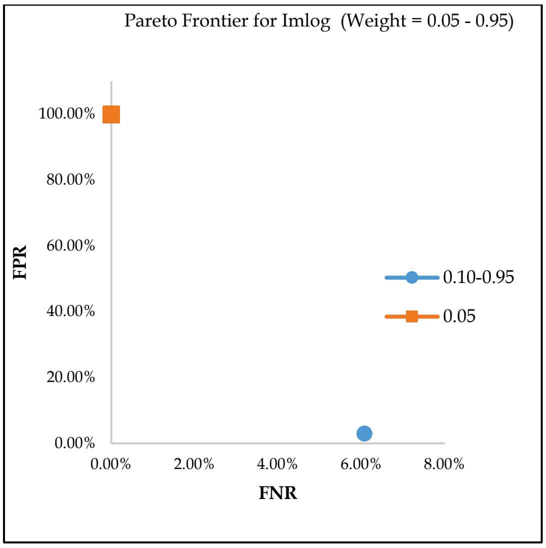


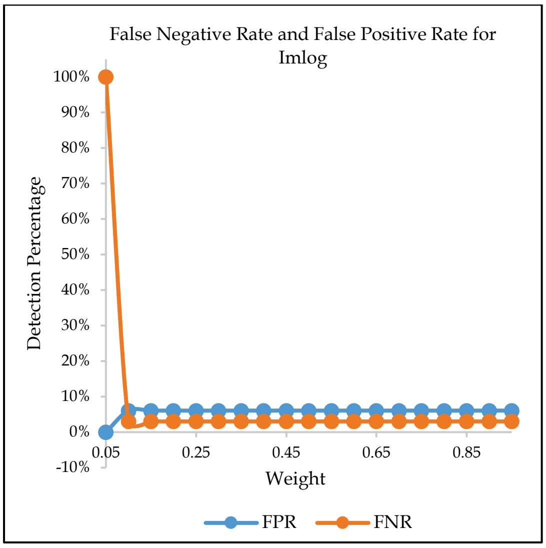


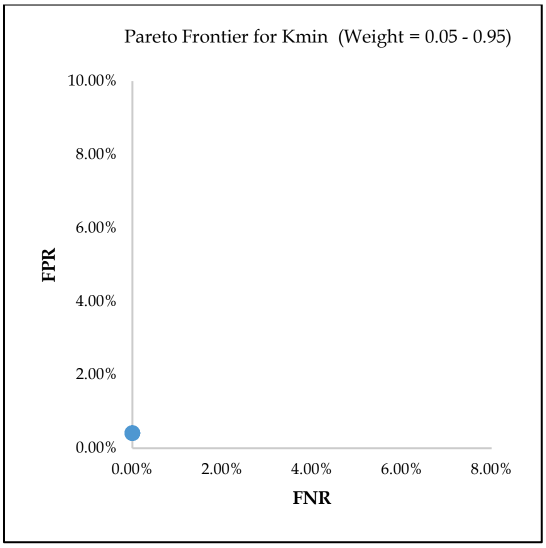


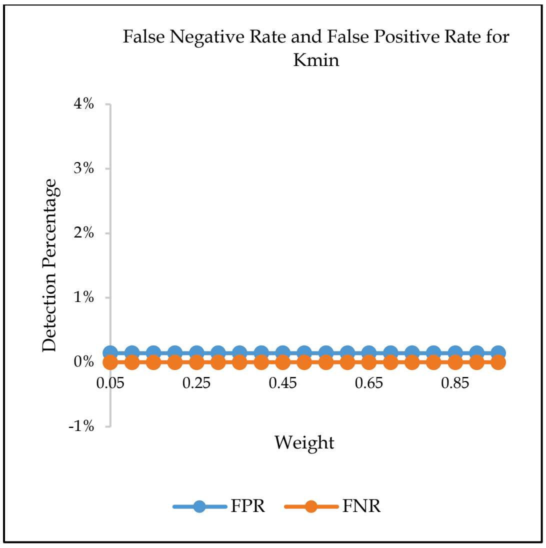


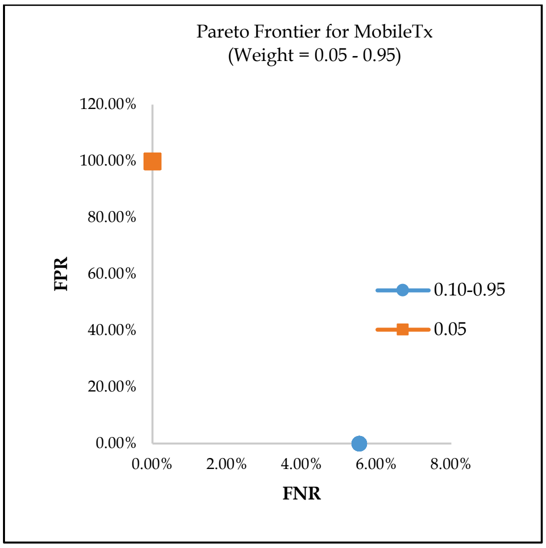
APPENDIX 2 - OTHER RESULTS FOR PARETO FRONTIER

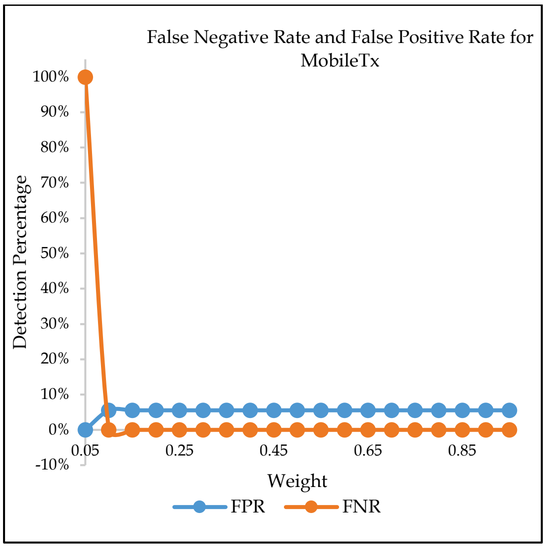




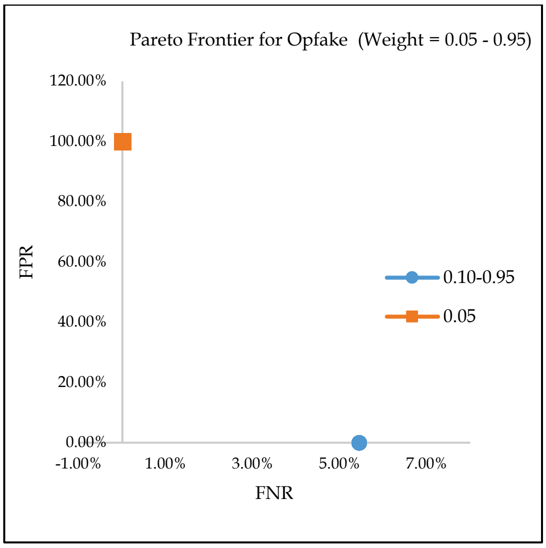


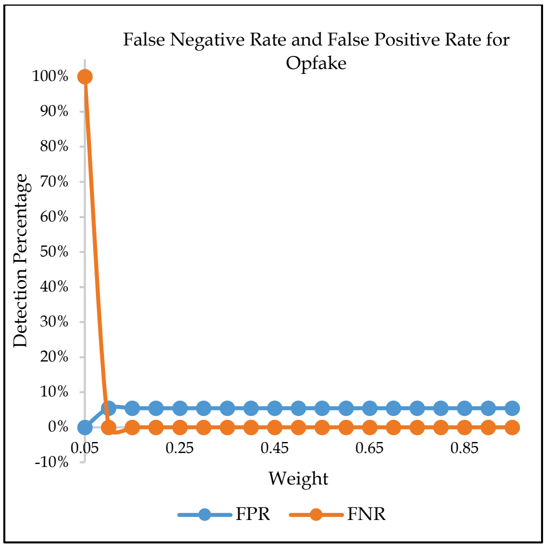


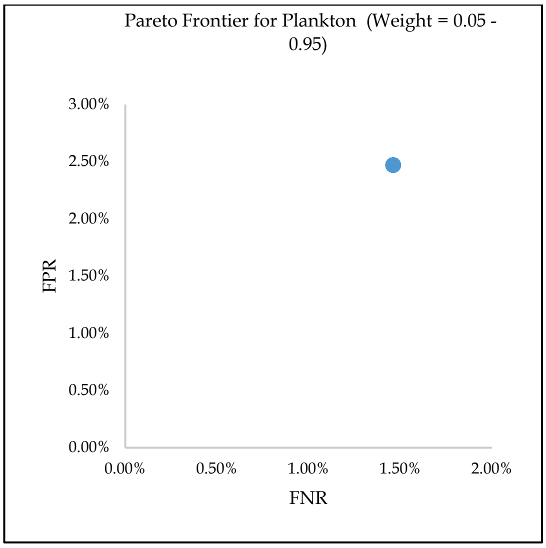


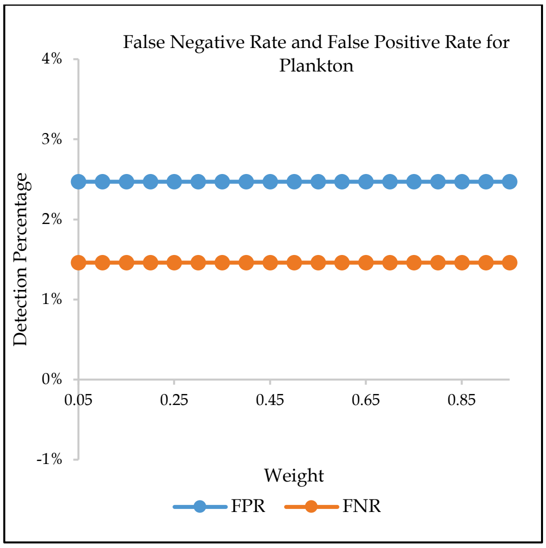


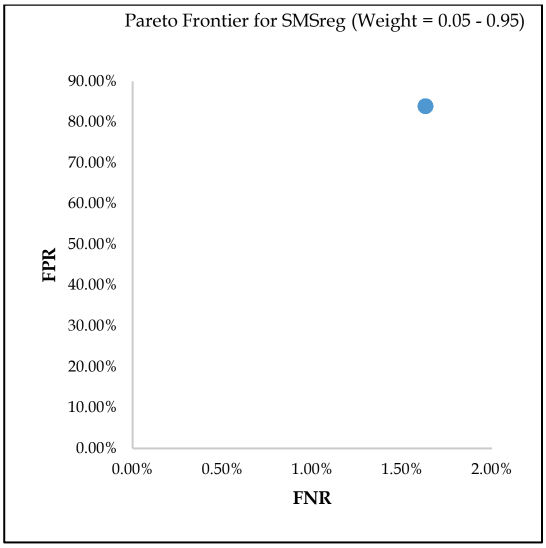
APPENDIX 2 - OTHER RESULTS FOR PARETO FRONTIER

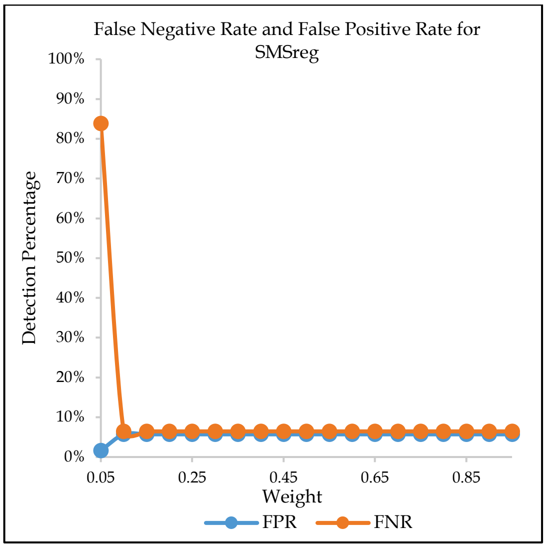

Pareto Frontier for Opfake (Weight = 0.05 - 0.95)


False Negative Rate and False Positive Rate for Opfake


Pareto Frontier for Plankton (Weight = 0.05 - 0.95)


False Negative Rate and False Positive Rate for Plankton


Pareto Frontier for SMSreg (Weight = 0.05 - 0.95)


False Negative Rate and False Positive Rate for SMSreg