An SMS-based Mobile Botnet Detection Framework
Using Intelligent Agents

by

Abdullah J. Alzahrani

Previous degrees
M.Sc. in Computer Security, LTU, 2008
B.Eng. in Computer Engineering, UQU, 2005

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

Doctor of Philosophy

In the Graduate Academic Unit of Faculty of Computer Science

Supervisor(s): Ali A. Ghorbani, Ph.D., Computer Science

Examining Board: Virendrakumar C. Bhavsar, Ph.D., Computer Science,
Natalia Stakhanova, Ph.D., Computer Science,
Brent Petersen, Ph.D., Electrical and Computer Engineering, Chair

External Examiner: Paul S. Pang, Ph.D., Computing,
Unitec Institute of Technology, New Zealand

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK
October, 2016
©Abdullah J. Alzahrani, 2017
Abstract

Along with increasing security measures in Android platforms, the amount of Android malware that use remote exploits has grown significantly. Using mobile botnets, attackers concentrate on reliable attack vectors such as SMS messages. Short Message Service (SMS) has been increasingly targeted by a number of malicious applications ("apps") that have the ability to abuse SMS features in order to send spam, to transfer command and control (C&C) instructions, to distribute malicious applications via URLs embedded in text messages, to send text messages to premium-rate numbers, and to exploit smartphones. Efficient detection and defence techniques that use filtering and blocking methods for SMS botnets is therefore an urgent necessity. Unfortunately, most botnet detection solutions proposed so far are reactive; that is, they require a large amount of data in order to effectively generate signatures and filtering rules to differentiate between normal and malicious SMS messages. By using proactive approaches such as a multi-agent system, agents can monitor certain environments and report abnormal behaviour in order to protect user data.

In this thesis, we propose an SMS-based botnet detection framework using intelligent agents that are used to detect malicious SMS messages and monitor smartphone resources which are typically targeted by SMS botnet attacks. The proposed detection framework is based on a multi-layer model which consists of three modules and intelligent agents. The first is an SMS signature-based detection module which can be used to combat SMS botnets, in which we first apply pattern-matching detection approaches for incoming and
outgoing SMS text messages, and then use rule-based techniques to label unknown SMS messages as suspicious or normal. The second module, an anomaly-based detection module, employs unsupervised learning techniques, using clustering algorithms to group SMS messages into four class labels and to classify reported text messages to one of those four classes. The module also uses a robust and efficient behavioural profiling analysis to detect whether there are any correlations between classification results and alerts from profiling analysis. Rule-based correlations are used to label SMS messages as either normal or malicious. The third module is a defence module that can be used as a more proactive approach which directly generates signatures and rules in order to protect Android smartphones from abuse by SMS botnets. This module is used to generate signatures of malicious SMS messages, to update phone number blacklists, to analyze malicious applications and to send feedback to Android smartphones so that the user can take action. Finally, a multi-agent system that can be used to observe Android mobile devices and to interact with service provider agents in order to detect malicious applications and SMS botnet activities on Android mobile devices. We have developed an intelligent and proactive framework that scans incoming and outgoing text messages, monitors Android resources and observes user usage that includes user connectivity time. The framework creates a user profile that is used to perform behavioural profiling analysis in order to identity malicious SMS and cut the C&C Channel.

The proposed framework has been implemented using JADE agents. We demonstrate the capability of the multi-agent system, signature-based detection, anomaly-based detection module, and defence module in accurately detecting SMS botnets, we conduct different experiments in three phases. In the first phase, we focus on evaluating the efficiency of the SMS signature detection module in Android devices. This module was evaluated using over 12,000 test messages. It was able to detect all 747 malicious SMS messages in the dataset (100% detection rate with no false negatives). It also flagged 351 SMS messages as suspicious. A comprehensive performance analysis of the anomaly-based detection module
is conducted in the second phase. The detection performance of the anomaly-based detection module has an average accuracy of 95% and an average of false negative rate is 3.95% on applied datasets. After having studied the performance of each module individually, in the last phase, we analyzed the overall performance of the proposed framework and provided a thorough analysis of JADE agents monitoring mechanism after demonstrating the capability of each module individually. We used approximately 60,000 test messages to evaluate the proposed framework. The signature detection agents reported 165 malicious SMS messages and 3,081 suspicious SMS messages. The anomaly-based detection module labelled 941 SMS messages as malicious.
Dedication

To my mother Mosferah,
who taught me to trust in Allah

To my father Jamaan,
who inspired me to believe in myself

To my wife Fawzyah,
who believed in me and cared for me

To my children Aleen and Anas,
who trusted me and made me happy

To my sisters Norah and Fawzyah,
who supported me

To my brothers Ali, Bader and Hussain,
who motivated me

To my brother-in-law Mohmmed and his family,
who cheered for me

To my brother-in-law Mazen,
who I hope gets better
Acknowledgements

First of all, I would like to praise God for granting me the strength and patience to complete my PhD thesis. I have been able to do it only due to his abundant blessings. Then, I graciously acknowledge the funding from the University of Hail and Saudi Arabia’s Ministry of Higher Education and thank them for their financial support and for granting me this opportunity to pursue higher education. In addition, I would like to express my deepest gratitude to my supervisor, Dr. Ali A. Ghorbani, for his guidance and support during the preparation of this thesis. His tremendous knowledge of Network Security has provided me with countless insights, and guided me throughout my research years. I also wish to express my sincere appreciation to Dr. Natalia Stakhanova, Mr. Ehsan Mokhtari and Mr. Hugo Francisco Gonzalez Robledo for their valuable comments, discussions and guidance during the course of this thesis. I also acknowledge the help and support of my friends and colleagues at the Canadian Institute for Cybersecurity, who always helped to keep my spirits high during any hard times. I would like to thank my committee members and examining board, especially Prof. Rodney H. Cooper, for taking the time to provide me with constructive comments. Lastly, I am thankful and indebted to my family and cannot thank them enough - my wife, respected parents, lovely sisters, adorable brothers, and my brother-in-law Mohammed. They have supported and motivated me throughout my journey.
Table of Contents

Abstract ii

Dedication v

Acknowledgments vi

Table of Contents xii

List of Tables xiv

List of Figures xvi

1 Introduction 1

1.1 Research Motivation ........................................ 5

1.2 Contributions of this Thesis ................................ 7

1.3 Thesis Outline ................................................ 9

2 Literature Review 10

2.1 Smartphone Security ........................................ 10

2.1.1 Mobile operating system (OS) .......................... 11

2.1.2 Android Platform .......................................... 12

2.1.2.1 Android Features ...................................... 13

2.1.2.2 Features of Android Application .................... 13

2.1.3 Short Messages Service (SMS) .......................... 14

2.1.4 Android Security ........................................ 15
List of Tables

3.1 Central service agents and their responsibilities. ........................................ 49
3.2 Android smartphone agents and their responsibilities. .............................. 51
3.3 Collected features for Android user profiling. ........................................... 56
3.4 Features selected for SMS signature-based detection ................................. 59
3.5 Suspicious SMS rules. ........................................................................... 62
3.6 Reported SMS features. ......................................................................... 66
3.7 Android user profiling features. ............................................................... 73
3.8 List of dangerous permissions. ................................................................. 77
3.9 Correlation rules. .................................................................................... 80
4.1 Agent - permission mapping tables. ......................................................... 131
5.1 Details of the datasets used for experiments. ............................................. 141
5.2 Overview of the extracted C&C and URLs. .............................................. 142
5.3 The confusion matrix ............................................................................ 144
5.4 Summary statistics of the extracted features. .......................................... 147
5.5 SMS spam collection dataset experimental results. ................................. 148
5.6 NUS SMS corpus experimental results. .................................................... 149
5.7 The experiment A confusion matrix. ....................................................... 153
5.8 The experiment B confusion matrix. ....................................................... 153
5.9 First experiments anomaly-based detection module performance ............. 154
5.10 The detection performance of the proposed anomaly-based detection mod-
    ule .............................................................................................................. 156
5.11 Third experiment findings. .................................................. 157
5.12 Comparison with other results. .......................................... 157
5.13 The number of signatures records. ..................................... 160
5.14 The NUS experiments datasets. ......................................... 161
5.15 Monitored permissions. ...................................................... 162
5.16 The proposed framework experimental results. ...................... 165
5.17 The proposed framework experimental results. ...................... 166
5.18 The SMS clustering and classification results. ....................... 167
5.19 The anomaly-based detection module results for NUS dataset. .......................... 168
List of Figures

3.1 Overview of the proposed SMS botnet detection framework. . . . . . . . . 42
3.2 Agents interaction design. . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
3.3 Use-case diagram for the proposed framework. . . . . . . . . . . . . . . . 47
3.4 Initial agent diagram for the proposed framework. . . . . . . . . . . . . . 48
3.5 Full agent interaction diagram for the proposed framework. . . . . . . . . 52
3.6 Structure of SMS signature-based detection module. . . . . . . . . . . . . 58
3.7 The architecture of anomaly-based detection module. . . . . . . . . . . . . 63
3.8 Profile behaviour analysis diagram. . . . . . . . . . . . . . . . . . . . . . 74
3.9 Overview of SMS botnet defence module. . . . . . . . . . . . . . . . . . . . 84

4.1 JADE agent architecture [17]. . . . . . . . . . . . . . . . . . . . . . . . . . 92
4.2 The architecture of the Android profiling framework using the JADE platform. 97
4.3 Architecture of the proposed multi-agent system. . . . . . . . . . . . . . . 99
4.4 Overall design of the central agent. . . . . . . . . . . . . . . . . . . . . . 103
4.5 Generated signatures database. . . . . . . . . . . . . . . . . . . . . . . . . 105
4.6 Overall design of the SMS profiling agent. . . . . . . . . . . . . . . . . . . 106
4.7 Android profiles database. . . . . . . . . . . . . . . . . . . . . . . . . . . . 109
4.8 Overall design of the Android profiling agent. . . . . . . . . . . . . . . . 110
4.9 Interaction between Android device agents and service provider agents. . . 113
4.10 Overall design of the Android agent. . . . . . . . . . . . . . . . . . . . . . 114
4.11 Signatures update database. . . . . . . . . . . . . . . . . . . . . . . . . . . 116
4.12 Overall design of the signature detection agent. . . . . . . . . . . . . . . 117
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.13 Overall design of the app-profile agent</td>
<td>120</td>
</tr>
<tr>
<td>4.14 Overall design of the user-profile agent</td>
<td>123</td>
</tr>
<tr>
<td>4.15 Agent deployment diagram for SMS botnet detection framework</td>
<td>137</td>
</tr>
<tr>
<td>5.1 First experiment results comparison</td>
<td>154</td>
</tr>
<tr>
<td>5.2 The first experiment: precision and recall</td>
<td>155</td>
</tr>
<tr>
<td>5.3 First and second experiments results comparison</td>
<td>158</td>
</tr>
<tr>
<td>5.4 The experiments distribution result of signature detection</td>
<td>166</td>
</tr>
<tr>
<td>5.5 CPU usages for obtaining signatures update</td>
<td>168</td>
</tr>
<tr>
<td>5.6 Memory usages for obtaining signatures update</td>
<td>170</td>
</tr>
<tr>
<td>5.7 CPU usages for signature detection performance</td>
<td>171</td>
</tr>
<tr>
<td>5.8 Memory usages for signature detection performance</td>
<td>171</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

One of the most serious threats to Internet security is the proliferation of botnets. Recently, there has been a dramatic rise in the use of botnets. Most recently, this growing trend has increased tremendously and become an increasing trend that compromises Internet users. Botnet attack and defence research is always the hot issue in the field of security. The etymological concept of botnet comes from the term “bot”, which means that victims are controlled by an attacker. The attacker, also known as the bot master, has the capability of controlling large-scale networks of bots from various locations in order to carry out attacks. In 1998, GTBot appeared as the first botnet followed by HTTP protocol, simple P2P protocol and Fast-flux mechanisms. These mechanisms have appeared as the representative of the Botnet in succession [75]. In recent years, the botnet development enters the phase of confrontation that can be used to advance the robustness of the botnet by using Domain Flux [125], URL Flux [136] mechanisms, Hybrid P2P [50]. The characteristics of C&C channels have evolved from IRC-based to HTTP-based, FTP-based, DNS-based, Twitter-based, and SMS-based, and from the centralized structure to P2P and Fast Flux Network Services [75]. Botnet has steadily developed into the cross-domain, multiple protocol fusion zombie network [141]. However, another wave of botnets has become a wider threat during this technology-driven era: mobile botnets. This has
become a serious issue because of the increasing worldwide trend in the use of mobile devices.

There have been recent improvements in the Android platform security, especially in the Telephony content provider (SMS provider), which allows applications (apps) to read and write short Message Service (SMS) and Multimedia Messaging Service (MMS) messages on a device [41]. In previous versions of Android, an app could easily intercept and abort an SMS message, but in Android version 4.4 device settings allow users to select a “default SMS app”. The default app can have full control of SMS features with the ability to write to the SMS provider and receive the delivery broadcast. Other apps can only read the SMS provider component and be notified when an SMS arrives by checking with the SMS RECEIVED_ACTION broadcast. The main intention of using the SMS RECEIVED_ACTION broadcast is for the app to read special incoming messages, and to perform an action, for example performing phone number verification by non-default SMS apps [41]. There are two trends that make our work important: 1) The user has to choose to select the default SMS app. In the meantime many apps have been developed that can provide full control to the SMS provider; some of these have more than 500,000 downloads, which raises a security issue. 2) Android smartphone users are still using previous versions of the Android platform that do not support new SMS provider security measures. During a seven-day period ending on February 2, 2015, the percentage of Android devices that were using Android version 4.4 was only 39.7%, while 60.3% of the Android devices were using an older version of the Android platform. This statistic is based on the number of devices that were accessing Google Play Store [101].

This lack of security puts devices at risk of attack through mobile botnets. A mobile botnet is a network of compromised smartphones that share the same command and control (C&C) infrastructure, controlled by a bot master to perform a variety of malicious attacks [52]. During a real-life attack, the owners of the infected phones are usually not aware that their phones are part of a botnet [29]. The first SMS mobile botnet is said
to have been assembled in early 2009 right after the SMS worm known as “Sexy Space” cascaded across cell-phone networks. The iKee.A worm infected “jailbroken” iPhones, and a developer added command and control mechanisms to convert iKee.A to a botnet, called iKee.B [100]. Most common mobile botnets utilize SMS-based architecture, which allows attacks to be performed through SMS without user knowledge of such activity on their phones [29]. Mobile phones have advanced at a fast pace over the past few years and society is striving to get the most out of the capabilities and functionalities that the new phones provide. Due to the nature of smartphones [133], security issues have arisen, such as a lack of IP addresses, problems with communication costs, changes in connectivity and new stealth mechanisms. Development of Intrusion Detection Systems (IDS) in traditional networks is one of the most well defined and extensively studied fields. Intrusion detection in mobile devices generally falls under the umbrella of this broader domain, and the core foundation of intrusion detection generally follows defined principles. At the same time, there are several specificities that make traditional IDS unsuitable for mobile devices. Some of these factors are constrained resources, mobility, a deployment environment, exposure and privacy [9].

One of the main components of a botnet is the C&C channel, which is used by attackers to carry out C&C communication. With the availability of SMS on smartphones, SMS messages are used to transfer C&C commands, send SMS spam, send premium-rate SMS messages without user knowledge [107] and distribute malware as propagation vectors. SMS based C&C presents some advantages for the bot master. The first advantage is that the attacker can communicate with the root node, given that communication utilizes a tree topology. The second advantage is that it makes detection of bot communications very difficult. On the other hand, SMS based C&C has disadvantages that pose challenges for bot masters. Initially, the bot master must assess whether the tree is still intact, and that there are no breaks in the tree or missing nodes that might compromise the communication process. An additional disadvantage is that SMS-based C&C requires a node list to be
operated on infected phones [84]. In the botnet, a C&C channel is considered to be the weakest link to the botnet, so losing a C&C channel means, the attacker will not be able to manage the botnet [55].

In order to address these issues, we propose a framework with two components, intrusion detection and a multi-agent system. The main component is an intrusion detection system with signature-based and anomaly-based detection modules. Signature-based detection is also called misuse detection or knowledge-based detection. Traditionally, the signature-based approach extracts the features from traffic and detects malicious activities by comparing incoming traffic to the signatures of the attacks. Signatures are patterns or sets of rules that can uniquely identify an attack. The advantages and disadvantages of this approach are extensively discussed by Debar et al. [39]. Signature-based detection has two main advantages, exceptional accuracy, and production of very few false alarms because this approach requires frequent updating of attack signatures. The main disadvantage is that signature detection is unable to detect an unknown attack. Additionally, this approach does not have the ability to detect every type of attack because these attacks do not all have corresponding signatures.

The anomaly-based detection approach is also be referred to as behavior-based detection. This approach builds models of normal data and then attempts to detect the deviation from the norm in observed data; this deviation is considered an anomaly. The key advantage of this approach is that it can detect new types of threats when there are deviations from normal data [22]. A shortcoming is the cost of dealing with the large number of false alarms. To improve the capability of intrusion detection, researchers have proposed hybrid detection, which is a combination of signature-base detection and anomaly-based detection. This hybrid approach will detect known attacks with high accuracy using signature detection, while also being able to detect unknown attacks through the use of anomaly detection.

The second main component of this research involves a multi-agent system composed of
multiple interacting agents. To provide robustness against failure, a multi-agent must be embedded to share the responsibilities among the different agents [124]. Agents have the ability to adjust their behavior in order to trigger alerts to resource constraints, such as network bandwidth and battery capacity. The concept of adapting a multi-agent approach to Smartphones is a very new area of research that focuses on the Android platform. Currently, several multi-agent platforms are being developed for Smartphones, and research is still ongoing. Multi-agent systems for botnet detection require a generation of agents set with some structure and functionality. Based on the work by Savenko et al. [114], agents are utilized when their results are transmitted to the effectors as a means of influence on the computer system. If malware is detected, an agent, through effectors, blocks the processes that are responsible for performance of some malware and then notifies the user about the infection. The proposed framework uses multi-agent system technology to aid in performing detection without exhausting smartphone resources such as battery and memory.

In this thesis, we propose an SMS-based botnet detection formwork that uses multi-agent technology based on observations of SMS and Android smartphone features. This formwork has the ability to detect SMS botnets and identify ways to block the attacks in order to prevent damage caused by botnet attacks. We developed an adaptive hybrid model of SMS botnet detectors by using a combination of signature-based and anomaly-based algorithms. These components utilize multi-agent technology to recognize malicious SMS and prevent users from opening the messages that infect smartphones. This framework includes a defence module that employs defence strategies.

1.1 Research Motivation

In spite of the fact that intrusion detection systems have been around for a long time, they have not been totally effective for critical applications special in smartphone platforms,
particular when it comes to detecting new and unknown attacks. While anomaly detection has been the major focus of research for identifying unknown threats, the number of commercial products in the market that have been developed based on anomaly detection methods are limited. In other words, the most effective IDSs are still signature-based products that most of anti-virus vendors are relying on. However, they offer limited capability in protecting a system or smartphone against the new attacks.

Taking on a worldwide and ominous presence, botnets are an effective malware-launching platform in which a new worm or virus is sent out instantaneously by numerous bots. Research has proven that the attacks that target mobile platforms such as spam, phishing, click fraud, key logging, key cracking, copyright violations, and denial-of-service (DoS) attacks are a common tactic used by black hats intent on bringing down a high-profile website.

On the other hand, we have conducted research to discover the most misused features in Android mobile devices. Our results show that outgoing SMS is the topmost misused feature, and that incoming SMS is also a highly misused feature. Based on these findings, protection of the SMS services is required. Mobile botnets are now using the SMS feature of smartphones as a C&C channel to conduct attacks. Almost every mobile phone in current use has SMS capabilities. This is of particular concern for Android smartphones, which are leading the market in mobile phones. With the rapid growth of smartphones, especially Androids, it is quite understandable that such growth will be followed by numerous attacks, in which attackers take advantage of many of the features of mobile devices. One of the most costly features is SMS messaging. Some cellular companies provide free SMS or low-rate SMS messaging. However, many people pay high rates to mobile phone providers for SMS services. Our research is aimed at protecting users against attacks or threats coming through SMS services, through the design of an adaptable detection system.

The continuous development of botnets and other malware reveals dynamic growth in its quantity [114]. The vast expansion of interconnectivity within the Internet and the rapid
evolution of highly-capable but largely insecure mobile devices threaten cellular networks [33]. In particular, mobile botnets have become a greater trend following the growth of traditional botnets in the internet world. Mobile botnet is a network of compromised smartphones that share the same command and control (C&C) infrastructure and are controlled by a bot master to perform a variety of malicious attacks [52]. Because of the features of smartphones, which include limitation of power resources, problems relating to communication costs, connectivity changes, lack of IP addresses, and new stealth mechanisms; mobile botnets are harder to detect.

In particular, SMS has been abused by attackers using a new threat to smartphones called SMS botnet. Since SMS is available to every smartphone, compromised SMS makes the smartphone easy to break into. SMS botnet can be used for sending SMS spam, transferring all command and control (C&C) instructions, launching denial-of-service (DoS) attacks to send premium-rate SMS messages without user permission, and propagating malware via URLs sent in SMS messages [153].

1.2 Contributions of this Thesis

The proposed framework employs hybrid detection approaches in order to counteract botnet attacks, by investigating damaging SMS botnet activities through the examination of smartphone behaviours. The most important contribution of this research is the development of a framework using detection approaches and a multi-agent system. This framework includes a multi-agent system for collecting and monitoring data and detection modules that together have the ability to detect known and unknown SMS botnets, along with a defence module that can make a decision and send an action to be performed. The contributions of this thesis can be summarized as follows:

- Proposing an effective framework to detect and deal with SMS botnets including its architectural components and their interrelationships: We
have developed a framework that enables the detection and blocking of malicious SMS messages and applications. This framework includes signature-based detection, a multi-agent system, an intrusion detection module with active components, and a defence module.

- **Designing and implementing a module that can be easily implemented using intelligent agents:** In order to protect, monitor, and collect data from Android devices, we developed a multi-agent system using JADE agents [18] consisting of four agents with different functionalities in each smartphone. The collected data is then sent to the server. To handle and manage the data, three agents are established in the server to provide services to other agents.

- **Proposing a SMS signature-based detection module:** In order to protect smartphone users, we developed a signature-based module to perform light-weight scans on incoming and outgoing SMS messages to detect malicious SMS and to intelligently handle the uncertainty in the intrusion detection module.

- **Proposing an intrusion detection module:** We developed a module to detect malicious SMS using anomaly detection approaches. The detection module is composed of five components as follows: SMS profiling collector, SMS clustering, SMS classification, behavioural profiling analysis, and SMS correlation. In addition, we have proposed an efficient correlation algorithm that is able to model the uncertainty of attacks.

- **Developing a defence module:** This module can be used to make informal decisions, generate signature updates, create new rules, update phone number blacklists, and carry out malicious application analysis. This module sends the actions to Android device to be performed by Android users or specific agents.

- **Evaluating the proposed architecture using the different types of datasets:** We have prepared a data set of SMS botnets by merging existing SMS spam datasets,
collecting SMS C&C commands from other papers, and running malware that generate SMS messages which are used to send unauthorized SMS and distributed malware. In addition, we statistically analyze Android malware samples from different datasets to extract phone numbers, C&C instructions and URLs from a variety of botnet families.

1.3 Thesis Outline

In Chapter 2, we present background information on the Android platform and intrusion detection. Different techniques for intrusion detection and mobile botnet are briefly explained. We also briefly review correlation methods and multi-agent systems. Chapter 3 provides the details of the proposed framework for SMS botnet detection using a multi-agent system. First, a high-level overview of the framework is presented. Second, we explain the design of multi-agent systems and define agents goals. Third, we describe our signature detection module on smartphones. Fourth, we illustrate the intrusion detection module in which we provide details of SMS clustering, SMS classification, Android user profiling analysis and SMS correlation methods and algorithms. Finally, the defence module is described that includes signature generation, phone number blacklists, malicious applications analysis, and response action approaches. We present a multi-agent system implementation of the proposed framework in Chapter 4. The multi-agent is implemented by using transducer [51] agents and belief-desire-intention (BDI) [104] agents. Chapter 5 provides details about the data used in our experiments and evaluation criteria. We report the experiments that are conducted for evaluating the proposed framework. Finally, the conclusion and future directions for our research are given in Chapter 6.
Chapter 2

Literature Review

In this chapter, important fields of work are reviewed that have a significant impact on our SMS botnet detection framework. First, we review Android mobile security and the mobile botnet field in more detail. In Section 3, we explain Intrusion Detection Systems. In Section 4, we describe Alert Correlation Techniques in more detail. Finally, we discuss the multi-agent System approach.

2.1 Smartphone Security

As smartphones are becoming more and more convenient in terms of their functionality, mobile security threats have considerably grown into a serious issue for mobile users, particularly those using smartphones and tablets [142 69]. The growing popularity and fairly lax security of smartphones are what made them eye-catching targets for attackers. Smartphone users could easily access the Internet and other applications, but, unlike PCs (personal computers), application of security measures like firewalls, antivirus and encryption are often scarce in smartphones. More so, the majority of smartphone users are prone to discarding these security issues, as they tend to perceive that surfing the Internet using their smartphones is ‘as safe as or safer than surfing on their computers’ [108].
2.1.1 Mobile operating system (OS)

Mobile phones and handheld communication devices have become a major aspect of people’s lives, as they do not only provide services for messaging and voice calls, but also provide other services such as camera functionalities, Internet surfing and many others. All mobile phones and handheld devices like smartphones need mobile operating systems (Mobile OS) in order to be able to realize their purpose and offer services to their users. By definition, a mobile operating system or Mobile OS refers to the software platform installed on mobile devices such as smartphones, tablets, and PDAs that enables application programs to run on the device. In the world of smartphones, there are several mobile operating systems that smartphone manufacturers are integrating into their smartphone products. These include Android OS, iOS, Windows Phone, Blackberry OS, Symbian OS, and webOS.

Mobile platforms operate on top of baseband or a real time operating system that interacts with hardware features of the phone. The three most well-known platforms are Android, iOS, and Windows Phone platforms. According to IDC Analyze the Future’s estimation in 2015, Android dominated the smartphone market share of 82.8%, iOS has the second highest share at 13.9%, Windows Phone has the lowest share among the three platform at 2.6%. Other platforms have a share in the market with 0.7% [63].

First, Android OS is most common and has the largest-installed base compared to other mobile Operating System as of 2013. Android OS for mobile devices is developed by the Open Handset Alliance, led by one of the largest tech companies in the world, Google. In addition, mobile devices running in Android OS sell more compared to those mobile devices installed with Windows, iOS and Mac OS devices combined. On the other hand, iOS (iPhone OS) is the operating system for Apple mobile devices including iPhone, iPad, iPod Touch and its Apple TV. iOS is developed based on the Mac OS X, and can be regarded as a modification or variation of UNIX. It consists of four abstraction layers - Core OS, Core Services, Media and Cocoa Touch6 [92].
Windows Phone is a proprietary smartphone operating system that is developed by Microsoft and was first launched in 2010, with major mobile manufacturers such as HTC, Samsung, LG and Nokia installed Windows OS for their mobile devices. Conversely, BlackBerry OS is developed by RIM (Research in Motion) intended for BlackBerry smartphones and tablet devices.

2.1.2 Android Platform

Among the most targeted platforms is the Android platform, mostly due to the ease of use of malicious apps, and the lack of proper defence. According to Kaspersky’s estimation, the number of mobile malware targeting the Android platform tripled in 2012, reaching 99% of all mobile malware. Also, Kaspersky stated that in 2013 there were more than 148,427 mobile malware modifications in 777 families and 98.05% of mobile malware targeted the Android platform.

Android is one of the fastest developing operating system available today. Although its fame is in the mobile world, it is also trying to become an accepted platform in personal computers. In this section, we will discuss the basics of the Android Platform. First, we include in the discussion a brief history of the development of this platform, noticing how each of the versions has its unique name and features. Later on, we also include the description of the Android features and then moving further to the describing the features of the mobile applications, specifically.

Android’s operating system has a unique way of naming its different versions from Donut 1.6 released in 2008 until that of the latest Marshmallows 6.0. Each of the first letters of these names follows the alphabet, starting with letter D: Donut (2008), Eclair (2009), Froyo (May 2010), Gingerbread (December 2010), Honeycomb (2011), Ice Cream Sandwich (October 2011), Jelly Bean (October 2012), Kitkat (September 2013), Lollipop (March 2015), and Marshmallows (October 2015). The official website of Android cites that its users should remember the android operating system as “something sweet”. The following
discussions provide a further description of the Android features and its applications.

2.1.2.1 Android Features

Direct manipulation is the prime feature of the Android interface. This meant that human users can integrate their movements while using a device designed with this platform. These movements can include swiping, tapping, pinching, and reverse pinching in order to manipulate a screen.

Development in the Android interface have made its features become widely “personal”. Later developed Android versions allow printing, starting with the Kitkat version; OK Google voice, with Google Lollipop; and the Fingerprint support finger print sensor equipped devices, as does the Marshmallow version. In general, Android platforms include features for booting the home screen, a wide selection of third party apps available through Google Play and other app stores, and display a status bar showing information about the device and its connectivity. They even allow the user the ability to change the launcher in order to change the look of the screen. All these and more are provided to allow the users to decide for themselves what activities they wish to use with the device packed with the Android platform.

2.1.2.2 Features of Android Application

As mentioned, devices that are packed with the Android platforms allow the user to download and install applications available through Google Play and other app stores. Applications are essential tools that have been developed through Java programming languages, with complete access to the Android APIs (application program interface). These applications add the functionality of the Android interface, from web surfing, e-mailing, reading, playing games, and even organizing.

The Android OS has a distinguishing structure. Android applications do not have a unique entry that programs usually have on other OS [67]. To develop an application
in Android, requires four main components: activity, service, broadcast receiver, and content provider. These four components work separately; consequently an element for transporting messages is required, namely, intent. Intent transfers messages from activity to activity, containing specific information concerning the operation to be performed. Activity is a user interface component that provides a screen with which the user interacts in order to do something. Service is a component run as a background process to perform long-running operations which are invisible to the user. Broadcast receiver responds to broadcast messages (events or intents) from other applications or from the system itself and wakes up the proper actions after receiving. The content provider component supplies and shares data between other applications on request.

The API (application programming interface) is a set of functions and classes provided to control the principal actions of Android OS. It is much more efficient to consider that malicious applications mostly use certain APIs, rather than to extract all the APIs from the source code of an application [67]. Seo et al. [115] determined suspicious APIs often used by malware by analyzing malware samples. 122 permissions are provided in the Android platform to notify the user what activities will be carried out and which resources will be accessed by an application. [67] From the perspective of Android malware, malware applications can request more permissions than normal applications. Moreover malware can often request permissions that have risks related to: user privacy and device security such as collecting user data, collecting device information, accessing Internet, or sending and deleting SMS without the user’s permission. Sarma et al. [112] analyzed malware and an application dataset to study the distribution of requested permissions for each dataset. They found 26 critical permissions related Android security.

### 2.1.3 Short Messages Service (SMS)

Since the early 90s, people can send and receive messages faster than they expect. The invention of the Internet paved the way to send and receive messages using a personal
computer. With the development of the Short Messages Services (SMS), messages can be sent, stored, or received using a mobile phone. This has made receiving, storing, and sending messages even faster since messages can be available through the users’ personal cellular phones that they can bring anywhere, anytime.

The development of the Android allowed the evolution of SMS. Today, short messages can be received, stored, and sent using Android applications. Moreover, the Android platform can also allow mobile chatting, which means that users can exchange messages once the mobile phone has an Internet connection. Today mobile chatting is among the most popular features of the SMS. It has also allowed much cheaper access to the exchange of short messages. Despite some limitations of the SMS such as the difficulty of being able to input more than 160 characters and some devices’ limitations on multimedia content, the popularity of the SMS remains unbeatable. Gupta et al. [56] notes the reason for SMS’ popularity as a “mechanism of sending and receiving messages [that] not only saves time but costs less as well. In many situations, one is relatively much more comfortable sending a message via SMS than talking over phone.” The innovation of the Android has even empowered the SMS, which can add to better value of both the SMS and the Android technology these days.

2.1.4 Android Security

At the core of the open and free access to third party stores for mobile applications, Android empowered mobile devices to face serious security issues. While different from the Apple store, Google’s Play Store had been a hub for mobile applications that could contain viruses and malware; thus, resulting in privacy invasion and worse, infecting the devices leading to a failure to function. The problem with Google’s desire to win the operating system war by entering the mobile business with an open-to-all Android is that there’s no way to completely ensure and control security [120]. Even if Google wanted to, certain issues concerning the design of the Android platform led to the company’s failure.
to provide solutions such as patching older versions of Android to provide protection from malware. Google has released various solutions in order to provide protection for Android users. In August this year, the company released a patch that served as a solution to devices affected by a malware called Stagefright. This has affected over 950 million Android devices beginning with the 2.2 version. Earlier than this, the company also acknowledged some issues concerning the malware vulnerability of Android devices and so they welcomed developers who could help them protect users against security bugs. However, Smith believes that Google still fails to protect its market against security threats “because it has no say whatsoever when it comes to update releases. Google can only issue timely updates to Nexus handsets and tablets whenever significant issues are discovered. But code belonging to OEMs might take longer to fix - and that’s even if carriers don’t hinder the entire update process”.

2.1.4.1 Android Mobile Security Threats

Security threats are very prevalent among Android-empowered mobile phones. Mobile phones are now smartphones, which allow users to share their personal information to the virtual world. While users enjoy the very open and easiness of access to mobile applications that could be essential to them, such openness allows unofficial applications that can possibly infect a device and threaten its security. According to a famous Internet security provider, the Mobile Threat Center, “In 2015, the Mobile Threat Center determined a significant growth in Android malware, which currently consists of 97 percent of all mobile malware developed. In 2014 alone, there were 1,268 known families of Android malware, which is an increase of 464 from 2013 and 1,030 from 2012”. There is also an additional concern citing that more than 35,000 of the applications available for the Android platform are malicious.

The mobile security threat is real in the Android platform. With the growing popularity of its use as (more than 70% of the mobile phone market share are Android users), there is a
greater possibility for hackers and other cybercriminals to introduce certain bugs in order to steal information and even use this stolen information against the user. The openness of the Android platform is one of the most considered reasons as the vulnerability of viruses is high amongst these devices.

2.1.4.2 Threats from Android Malware

Three major groups of malware infect mobile devices that contain Android platforms. These include the SMS Trojan viruses, advertising modules, and exploits. Malware that have C&C mechanisms are considered botnet. In this section, we describe each of these malware and describe the threats they impose to mobile devices with Android platforms. Trojan viruses have been known to target personal computers. However, the growing popularity of the mobile phones has also led to the growing prevalence of platforms affected with Trojan viruses. In mobile phones, these viruses can spread out through an SMS. Web links and even applications can contain malicious components that can load and execute Trojans into the system of the device. Once activated, Trojans can enable cyber-criminals to spy on users, steal their sensitive data, and gain backdoor access to the system [71].

Included among threatening malware in the Android platform are advertising modules. They can come in the form of scareware, where the user sees a warning message about an upgrade of a battery or a cleaning application to maintain optimum performance. Spring [122] writes about nasty Battery warning that could lead to the user downloading the Battery app that contains malicious components potentially infecting the device. Dangers imposed by such malware can include “harvests the phone’s address book, the phone number, the user’s name and email address, and the phone’s unique identifying IMEI number. With a phone user’s name, IMEI, and wireless account information, an attacker could clone the phone and intercept calls and SMS messages, or siphon money from a user by initiating premium calls and SMS services” [122].

Finally, exploits are dangerous malware that can threaten Android devices. In 2015,
Stagefright was among the most dangerous exploit malware that affected thousands of Android mobile device users. In August 2015, Stagefright presented a danger when Google failed to provide a patch that could give a solution to such a malware problem. Exploits are dangerous because they are rooted to the Android system, which can lead to further damage later on.

2.1.5 Android Limitations

Despite the great advantage of the Android platforms among its users, there are also limitations in the design of the platform. This section will specifically discuss the limitations present in the resources and specific characteristics of the Android platform. The platform is also limited in its technology due to some permission limitations. Additionally, the Android platforms has been facing challenges in regards to its security. We next discuss how this vulnerability could further hamper the advantages it now enjoys.

2.1.5.1 Resource Limitation and Specific Characteristics

An Android platform has a specific environment that is different from the desktop computer. This creates some issues in developing a wider spectrum of security solutions that can work in a variety of devices that the platform can use. There is also an issue regarding the speed of developing applications. With the limited resources within the Android environment, there is a little challenge in maintaining efficiency and speed in the design process.

2.1.5.2 Self Protected Technology

The Android Platform also has its self-protected technology through its native code processing, which was intended for self-containment and exclusivity on CPU-intensive operations. This, however, becomes a limitation in a way that it can restrict some apps for protecting security and “contain exploits against system security flaws” [47].
2.1.5.3 Lack of Current Security Measurement

Another important limitation that Google and Android developers should focus on is the way to maintain security on this device platform. In their technical paper, Fedler and colleague [47] wrote about the security threats occurring among Android devices as well as the further threats that could arise in the future. In their paper, the authors describe how certain flaws are evident in the manner of measuring security for users and their devices. It reveals how the file system permissions of Android OS are not sufficient. Finer permission distinction is needed, as an open app market is not a strongly monitored and trustworthy software source [47]. The Google Play Store has also become a hub of free illegitimate copies of paid content, which could also mean open access to malicious applications that could damage the device or even threaten the user’s privacy. The platform’s long process of patching and its extensive procedure could generally affect the security even further. The insufficiency of file system permissions allow the Android platform to vulnerabilities in security. Furthermore, there is also a problem with how Android platforms’ open access to third party apps can generally became the “main channel of malware distribution” [47]. Despite the efforts of Google in managing risk through the 2012 Bouncer, which can “examine apps submitted to the Android Market automatically by execution inside a virtual Android environment in Google’s cloud infrastructure” [47], this still proved ineffective. The same is true with the use of patching as a way to improve security. The executive procedure proved to create further problems with security.

2.2 Botnet

Botnets are generally considered as an army of compromised computers that could transmit spam or viruses to other computers on the Internet. Botnets can allow further damage caused by malware and thus lead to the security issues faced by android devices. These are composed of bots, “a type of malware that allows an attacker to take control over an
affected computer” [89].

In mobile application, typical botnet functionality includes spam message delivery, stealing credentials and performing denial of service attacks [47]. The prevalence of botnet can even go further when devices become vehicles to relay spam and even providers “with high quality contact details from various services’ address books” [47]. With the open access feature of this platform, the vulnerability of the Android device is even higher.

Botnets are known to have existed in Android mobile beginning in 2009. Later on, however, this led to furthering the developing of botnets, as reported in Chickowski [35]: The botnet and the mobile malware family that drive it turn mobile devices into TCP proxies that can be used in any number of creative ways, including sending spam, committing click fraud, employing brute-force passwords, and initiating fraudulent ticket purchases.

2.2.1 Principles of a Botnet

The continuous growth, development and expansion of the Internet comes with the increasing pervasiveness of cyber attacks and intrusions [149]. Existing literature has long been acknowledging the existence of various forms of security threats in the virtual environment, yet some of these forms are little known by academics and the general public. One security threat is botnet. By definition, botnet is the term used to define the networks of infected end-host, called bots, that are under the control of a human operator, commonly known as a botmaster, through command and control (C&C) channels [3]. Botnets can basically be used as a platform for cyber attackers that launch several extensive and well-coordinated cyber attacks including spamming, information theft, DDoS or distributed denial of service, phishing, search engine abuse and click fraud [147, 26, 139, 106, 59].

Drawing from the above-stated definition of a botnet, there are two important phases in the lifecycle of a botnet: (1) Infection phase and (2) Control phase. The infection phase involves the clean host being infected by a bot binary, which then becomes a member of a botnet. These can be done through the conventional scanning-and-exploiting approach
(e.g. worm propagation, IRC-based bot propagation), spam-based infection (e.g. spam email that install bot binary) and the web-based malware infection (e.g. social-engineering-based infection and drive-by download attacks). The control phase begins once the bot-infected hosts are controlled by botmasters through command and control (C&C) channels in order to launch the cyber attacks \[151\]. In mobile devices, botmaster utilized the SMS service as a C&C channel in order to gain control of the infected smartphones.

2.2.2 Mobile Botnet

Smartphones and mobile devices have become an important part of everyone’s life; however, security threats are evident. As smartphones and other mobile devices are becoming more and more complex and sophisticated in terms of their capabilities similar to that of personal computers, mobile threats are increasing \[108\], including threats from mobile botnets. By definition, mobile botnets refers to the collection of compromised nodes due to mobile malware, which are able to perform coordinated attacks \[76\]. Unlike its traditional counterparts, mobile botnet attacks are performed in mobile devices as its primary platform. In a traditional botnet, the botmasters and the slave run on the same platform (e.g. PC), and the mobile botnet attack is performed by the botmaster using a PC or a mobile phone, controlling the slave is on a mobile phone \[86\].

The increasing prevalence of mobile botnet attacks is driven by factors including the connectivity of the mobile phone that make communication with a C&C server easier, as well as mobile devices being lucrative attack platforms for attackers \[86\]. There could be various types or forms of botnet attacks within the mobile platform, including unwanted sending of emails and SMS/MMS, information theft using spyware, privacy issues, and many others \[68\]. However, there are certain challenges that botmasters face when conducting a mobile botnet attack. First, the battery power is limited compared to that of a PC, which can result in easy detection by a user as battery power consumption speed exceeds the normal measure. Second, if the C&C consumes an abnormal amount of network
traffic, the abnormality is likely to be noticed. Lastly, the absence of public IP addresses as well as the regular change in network connectively would likely make the P2P-based C&C in PC-based botnets unfeasible [146].

2.2.3 SMS-Based Botnet

SMS-based mobile botnet has become an evident trend in the field of cybercrime forensics, considering that SMS, unlike Internet access, has always been a primary service provided by all mobile devices, including smartphones and even tablets [57]. In 2013, SMS-based mobile botnet became evident as cybercriminals were pointed out as the developer of the then recently-discovered Android malware that masqueraded itself as a Google application in order to steal messages from mobile phone users, particularly Android smartphone users [134].

A lot of studies have provided some attestation regarding the feasibility of an SMS-based mobile botnet and its potential damage to smartphone or mobile phone users. The research has proven that SMS messages can be used to transfer command and control (C&C) instructions, distribute SMS spam, launch denial-of-service (DoS) attacks to send premium-rate SMS messages without user permission, and propagate malware via URLs sent within SMS messages. One prominent piece of research, about mobile botnets that use SMS as a propagation vector, was conducted by Hua et al. [60]. They proposed an Android-based mobile botnet which utilizes the SMS as its platform to work. In their proposed botnet, Hua et al. [60] made use of an Internet server in order to create and establish the botnets’ topologies, as well as to control the infected phone or nodes to send an SMS to its other neighboring nodes. In order to make the malicious behavior invisible, Hua et al. [60] developed a rootkit to manage the encrypted SMS message exchanges. Their findings of their proposed SMS-based Android botnet indicated that the botnet was able to pull off 90 per cent coverage of the 20,000 attacked mobile phones in just a small fraction of time through the use of Edros-Renyi random graphs. Basing on the ability of the developed botnet in
inflicting damage to potential smartphone users in just a small period of time, they pro-
posed two defense approach or strategies - (1) a hardware-level alarm of SMS exchange
and, (2) a honeypot that easily captures the malware installed in a mobile device.

In light of this, the detection of mobile botnets is also a major problem because they are
also hard to detect [50]. A review of the literature has identified various strategies and
detection mechanisms to address the challenges of detecting mobile botnets. These strate-
gies purport many implications, most particularly for OS vendors and software providers,
as mobile botnets have imposed themselves on the operating systems of mobile phones
in their design. One probable countermeasure that would aid in the detection of mobile
botnets is outlined in the study by Zhen et al. [150]. They proposed an SMS commanded-
and-controlled and P2P-structured mobile botnet. The principle behind the propagation
of the botnet involves involvement of users, as well as exploitation of any mobile vulnerabil-
ity. Here, a mobile botnet makes use of an SMS command and control channel, acting as a
communication mode between the mobile bots and the botmaster. To facilitate reception
of commands through SMS messages, each of the mobile bots has an 8-byte pass-code.
More so, the SMS messages are designed as spam message in order to avoid detecting
the commands being transmitted through SMS messages. To deliver and propagate the
command-contained SMS messages, the botmaster will exploit the SMS services that are
being provided by the Internet to send the message. Both structured and unstructured
P2P topologies were utilized in their simulation. Based on the findings, their proposed
mobile botnet using a modified structured P2P typology could only make use of SMS as
its C2C channel. They also noted that SMS-based transmission of commands is a costly
way or channel for botmasters to lunch a mobile botnet attack [150].

Another research paper about the SMS botnet was a remarkable study by Mulliner et
al. [83], wherein the researchers studied three forms of botnets that could work on Apple
iPhone devices - P2P, SMS-based and SMS-HTTP botnets. Taking into consideration
the fact in the SMS-based botnet, every single command is generated and transmitted
through SMS, and a tree of nodes (phones) is then created in order to limit the number of messages being exchanged. On the other hand, SMS-HTTP botnet makes use of an URL of a certain website link that is being sent to some randomly-selected bots through SMS. These bots will download and carry out the encrypted messages posted on a URL [83]. Nguyen et al. [85] propose methods to detect SMS C&C botnets from infected Android devices. Their approach is based on Android radio logs, whereby logs are required to be read and the radio activities associated with SMS-based botnet activities identified.

2.2.4 Malicious Activity of a Mobile Botnet

A recent study by a software company, Kaspersky Lab, reported that the most known and prevalent attacks that target the Android platform include RiskTool, Adware, Trojan, Trojan-SMS, Trojan-Spy, with RiskTool leading the pack [46]. In addition, mobile botnets are also becoming a prevalent and dangerous security threat that attack mobile platforms, perform malicious activities at the instructions of the botmaster. There are a wide range of malicious activities that are performed by mobile botnets. Karim et al. [68] identified some of the different Android mobile botnets and their categories and summarized their type of malicious activity and attacks, the mobile platform they run on, the target audience, the dissemination or propagation techniques, and their operational impact.

2.3 Intrusion Detection Systems

SANS Institute [113] defines intrusion detection systems (IDS) as a security system that monitors computer systems and network traffic and analyzes that traffic for possible hostile attacks originating from outside the organization and also for system misuse or attacks originating from inside the organization. The principle behind the intrusion detection system is similar to that of a burglar alarm of a lock system of a car. If someone breaks the lock system and attempts to steal the car, then the burglar alarm detects the intrusion
and alerts the owner by raising an alarm \[113\]. Most of the IDSs that are used for detecting botnets are classified into any one of the three major categories: signature-based (misused detection), anomaly-based detection and behavior-based detection.

### 2.3.1 Signature-Based Detection

Signature-based detection has also been a widely-used technique in identifying and detecting malware in the internet world. Many of the anti-virus applications utilize signature-based detection since it is the easiest approach to detection. Traditional detection applications perform scanning, where the content of computer files is scanned. The content is then cross-referenced with code signatures that are associated with malware or viruses. If the scan and comparative action reveals a matching signature, then the content can therefore be considered an infected file. These code signatures are contained in a library, which is continuously updated by software providers and vendors. The same concept of signature detection in anti-virus software applications is applied in detecting mobile or traditional botnets. In the work undertaken by Behal et al. \[129\], botnet attacks are detected and prevented by analyzing the outbound traffic. For their signature-based botnet detection, the outbound traffic is examined through their N-EDPS (Network exclusion detection and prevention system). The authors stress that signature-based N-EDPS is more effective than N-IDPS (network inclusion detection and prevention system) since a smaller database of rules or signatures is needed.

Signature-based detection (also known as misuse detection), is regarded as a complementary detection mechanism to anomaly detection, in which it assumes that known attack patterns could easily and effectively be detected and identified through the use of explicit knowledge. It is complementary to the anomaly detection in such a way that it searches for and detects well-defined patterns of a specific or pre-defined attack or vulnerability that anomaly detection techniques can not distinguish \[21\].

Despite misuse or signature-based detection being efficient in detecting known attacks or
vulnerabilities, one key limitation of this detection category is that it is impractical to use when new types of attacks with no signatures available yet to detect occur. Nonetheless, its effectiveness and easy to understand concepts are what makes misuse or signature-based detection systems common to security experts.

2.3.1.1 Patterning Matching

The pattern matching-based intrusion detection approach is often the common choice among security experts especially when applied to network-based intrusion detection systems. The detection approach makes use of attack patterns that are modeled, matched, and identified based on the packet header, packet content, or both. Attack patterns could also be established in host-based intrusion detection systems through concatenating the words representing the system calls in a system audit trail. With the continual of new types and varied forms of attacks the number of signatures is constantly growing, thus making the pattern matching more expensive in terms of the computational cost [53].

A lot of studies in the past have explored the viability of pattern matching as a primary technique for intrusion detection systems. For example, Norton [89] presented an optimized version of the Aho-Corasick multi-pattern search engine used in Snort. Snort is an open source intrusion detection system, that utilizes algorithm that enables the system to scan the network traffic packets search for any intruders just by looking for specific values in the network headers and through searching for any known patterns in the application layer data. The enhanced version of the system takes into consideration various aspects increase the performance of pattern searching and matching capabilities of the system, which includes the pattern character case sensitivity, pattern sizes, pattern group size, search text size, frequency of searches and many others [89]. There are also other studies that contributed to the literature exploring the use of pattern matching techniques for intrusion detection (e.g. Abbes et al. [1]; Dharmapurikar et al. [43]; Chen et al. [33]; Zheng et al. [152]).
2.3.1.2 Rule-Based Techniques

Rule-based techniques are one of the popular approaches used for misuse or signature-based detection. Here, expert systems program intrusive scenarios that function as a set of rules, in which these rules are matched against network traffic data. If the matching process finds any deviation in the traffic data against the rules, then the system recognizes an intrusion [53]. Multics Intrusion Detection and Alerting System (MIDAS), Intrusion Detection Expert System (IDES) and Next-generation intrusion detection expert system (NIDES) are some of the examples of rule-based misuse detection systems [53].

The IDES framework involves mathematical codification of intrusion detection mechanisms, based on two key assumptions: (1) that normal interactions between subjects (e.g. users) and objects (e.g. files, programs, or devices) can be characterized; and (2) that users always behave in a consistent manner when they perform operations on the computer system. These usages can be characterized by computing various statistics and correlated with established profiles of normal behaviors. New audit records are verified by matching known profiles for both subjects and their corresponding groups. Deviations are then flagged as intrusions [53].

2.3.2 Anomaly-Based Detection

Anomaly-based intrusion detection systems differ from signature-based systems in that instead of identifying and flagging misuses, they generate the pattern of normalcy for the system and flag deviations from the normal. The basis for this approach is that intrusive network traffic and user activity are inherently different from legitimate network traffic and activity, respectively. In line with this fact, any activity that displays behavior that is deemed abnormal by the IDS is then flagged as intrusive. In order to ascertain what is normal, an anomaly-based system uses sensors to collect and store network and user data. Using data mining techniques, the informative characteristics of the data that best describe normal behavior are selected. Finally, machine learning techniques are used to
learn patterns, using the selected parameters over a period of time, to create a suitable baseline profile [91].

The main advantage that anomaly-based intrusion detection systems have over signature-based systems is that they are capable of detecting new attacks or variants of existing one. Whereas signature based systems are limited to detecting attacks stored within their signature database, anomaly-based systems are, theoretically, boundless as to what they can detect. An additional characteristic of anomaly-based systems is their flexibility in training, which can be supervised or unsupervised. In supervised training, the learning process is rule-based. Sensor data is collected and explicitly labelled as normal or intrusive by an expert to let the system learn what kinds of traffic to allow. Rules regarding how much leeway to allow when applications digress from standard protocol, as well as how to deal with sudden spikes in activity, could be set. Activity such as multiple access to sensitive data, or bursts of SYN packets, which could signify a port scan, or bursts of activity to a port, which could signify the onset of an attack or a worm like Blaster. Unsupervised training on the other hand uses statistical methods and thresholds to automatically detect and adjust itself to the current environment. The learning process starts by collecting, over a period of time, necessary statistics and information on the distributions of file accesses, log on attempts, flows of certain kinds of network traffic, and other facets of the system. When a suitable model is garnered, thresholds are set and any activity that crosses the thresholds is flagged as anomalous [91].

2.3.2.1 Advanced Statistical Methods

In 1987, Dorothy Denning published seminal work about an intrusion-detection model that has the capability of sensing and distinguishing any intrusion, penetrations and other types of computer attacks. Her intrusion detection model is grounded on the assumption that security violations could be perceived and detected through checking and monitoring a system’s audit record for any abnormal patterns of system usage.
The basic conception on Denning’s detection model, with slight alterations, is evident in various intrusion detection systems, primarily statistical-based model for anomaly detection. For example, the Haystack system is one of the earliest anomaly detection systems that is based on statistical models [53]. Another classic example of an anomaly detection system based on statistical models is the EMERALD or the Event Monitoring Enabling Responses to Anomalous Live Disturbances. It consists of a signature analysis component and a statistical profile-based anomaly detection component. The anomaly detector is based on a statistical approach in which the events are labelled as intrusive if they are largely deviant from the expected behavior [53].

2.3.2.2 Rule-Based Techniques

Besides misuse detection, there are also rule-based techniques applied for anomaly detection. Some classic examples of rule-based techniques for anomaly detection include Wisdom & Sense (W&S), Network Security Monitor (NSM), Time-based inductive machine (TIM) and many others.

First, Wisdom & Sensor is a distinctive technique for anomaly detection that involves two elements - wisdom and sense. The wisdom element basically involves a set of rules that describe the normal behaviors of a particular system basing them on the historical audit data. On the other hand, the sense element is an expert system which is grounded on the basis of previous rules, which tests and validates whether or not the succeeding audit data contravenes the set of base rules. An alert will be performed by the sense component once it detects any anomalous behaviors [53]. On the other hand, the NSM is a rule-based intrusion detection system, which is the first to directly utilize network traffic as its primary data source. Conversely, the Time-based Inductive Machine, which was developed by Teng et al (1990) to capture a snapshot of a user’s behavior pattern. Generally, it distinguishes any temporal sequential patterns in a sequence of events [21].
2.3.2.3 Learning Models

Learning models that forms the basis of anomaly detection integrate learning capabilities into the overall intrusion detection procedure. This is made possible through the use of artificial learning techniques. There are two detection scheme categories in a learning-based detection namely supervised and unsupervised.

In supervised learning, it is assumed that the training dataset is available, in which the dataset consists of instances that are labelled as normal or as belonging to known anomaly classes. The common approach to this is the process of generating a class predictive model for normal versus anomalous behavior. The new data instances are then tested against this model to create their connection or membership to a certain class [20].

On the other hand, there is no training data necessary for unsupervised anomaly detection, as it involves an unlabelled dataset taken as input for the intrusion detection. The detected intrusion instances can then be utilized for training of misuse-based or supervised method. There are various unsupervised anomaly detection methods, which include clustering-based anomaly detection methods, detection using outlier mining, detection using association mining, detection using Hidden Markov Model and many others [20].

2.3.3 Behavior-Based Detection

It is evident that there is an increased sophistication in the various types of security threats including malware and botnets, as well as an increased inefficiency of binary signature matching anti-virus software. More so, modern malware uses increasingly advanced schemes to avoid detection. Nowadays some malware is crafted for one time use against high value targets. This type of malware is difficult to detect mainly because it employs advanced mimicry and hiding techniques. Moreover, it often requires insider knowledge for them to hide and spread efficiently. The attack usually evolves slowly over time with a small footprint making it even harder to notice using statistical methods [44].

This technique uses the concept of clustering, which is utilized to find representative beh-
haviour clusters. In the work of Chang et al. [31], the detection system makes use of the process of attribute selection and behaviour clustering in which attributes are chosen based on observation of the action sequence of normal users. Their cluster-based profiling approach successfully captures the node behaviour clusters in a network, and then two detection schemes are facilitated through formal statistical tests – behaviour proportion-based tests (BPT) and behavior mean distance-based tests (BMDT). Behavior-based detection functions just like any other detection technique, wherein the detector first characterizes the normal process of sending SMS messages by a system-call state-diagram and then keeps monitoring the system calls that generate outgoing messages to see if there is any deviation from the norm. Such an approach utilizes a detecting algorithm that identifies the encoding scheme using binary analysis so that it is able to intervene and delete malicious messages before giving access to any application in the mobile phones [150].

2.3.4 Hybrid Detection

There were a lot of studies in the past that exemplified the approach of using both signature-based/misuse detection and anomaly detection in order to improve intrusion detection capabilities ([96]; [118]; [56]). In a study, Tombini et al. [132] used an anomaly detection method in order to establish a list of suspicious items. Then, a misuse detection method was utilized in order to categorize these suspicions into three different categories - false alarms, attacks and unknown attacks. Their approach was grounded on the hypothesis that a high detection capability could be achieved through the anomaly detection component since missed or neglected intrusions might occur as the signature detection component failed to detect them. Although the proposed hybrid system missed certain types of attacks, it did reduce the false alarm rate and increased the likelihood of examining most of the alerts.

The study by Tavallaee et al. [53] proposed a hybrid IDS combining both misuse detection and anomaly detection components, in which a random forests algorithm was applied
firstly in the misuse detection module to detect known intrusions. The outlier detection provided by the random forests algorithm was then utilized to detect unknown intrusions. Evaluations with the part of 1999 KDDCUP data set showed that their misuse detection module generated a high detection rate with a low false positive rate and at the same time the anomaly detection component has the potential to find novel intrusions. The whole hybrid system achieved an overall 94.7% detection rate with 2% false positive rate.

2.4 Alert Correlation

Alert correlation is an analysis process that occurs when the alerts being generated by the intrusion detection systems are monitored and generates compact reports on the security status of the network under surveillance. Basically, the primary objective of the correlation process is to create a detailed overview of security-related activity on the network. This process incorporates a series of components that transform intrusion detection sensor alerts into intrusion report [136].

In a much simpler definition, alert correlation starts with receiving a stream of alerts produced by the intrusion detection system as input. In each component of the process, alerts are merged in to high-level intrusion reports or tagged as irrelevant if they do not represent successful attacks. Finally, the alerts are prioritized according to the site’s security policy, and eventually the results are reported [137].

Normalized alerts have a standardized name and are in a format that is understood by the other components of the correlation process. However, additional pre-processing may be required, since some sensors omit fields that are necessary for other components of the correlation process (i.e., time, source, and target). The goal of the pre-processing component is to supply, as accurately as possible, missing alert attributes that are important for later correlation components [137].

Basically, the preprocessing phase transforms alerts from different sources into a normal-
ized format and then amalgamates multiple alerts into a single alert; hence removing any duplicates and significantly minimizing the amount of time to process and evaluate. First, a normalization step is conducted, in which alerts are converted into a generic format and reduces the number of alerts to be correlated. There are various methods for normalizing data, but the most common is the IDMEF or the intrusion detection message exchange framework. Correlation is too difficult to be attempted in a single stage. Research work conducted on these correlations can be divided into four main categories, namely statistical correlation, scenario-based correlation, rule-based correlation, and temporal correlation [109]. We apply rule-based correlation in the SMS botnet detection framework. One advantage of using rule-based detection is that the set of rules helps to produce fewer false positive alarms [66]; it also has the ability to label unknown attacks. Many researchers have studied the use of requested permission to check the risk of an application, but it is hard to guarantee high detection rates in the permission-based detection approaches. This framework will require all alerts to conform to different attributes including analyzer, create time, classification, detection time, analyzer time, source, target, assessment and additional data. On the other hand, data reduction is conducted which involved removing redundant alerts from the processing chain. Alerts could either be aggregated or filtered.

2.4.1 Correlation Techniques

Correlation often makes use of various techniques in order to create logical associations or relationships between alerts or to determine attacks that happen in stages. One of the most common correlation technique is the feature similarity technique, which involves clustering alert based on their similarity in parameters (e.g. source IP, port number, target IP, etc). The features for the correlation engine to assess include: (1) similarity between source IP, (2) similarity between target IP, (3) similarity in target port numbers, (4) similarity between target IP and subsequent source IP, (5) backward correlation and (6) frequency of alert correlation [97].
Another common method is the known attack scenario, in which known attack scenarios are programmed and written using either expert rules or machine-learned training rules. This reveals the causal connection between the alerts but can only detect and monitor known intrusion. Features in use in this approach include (1) alert type, (2) time between alerts, (3) similarity of consecutive bit of destination IP, (4) similarity of consecutive bit of source IP, and (5) similarity of consecutive bit of last destination IP vs. new source IP [97].

### 2.5 Multi-Agent System

There has been increased attention on using multiple agents incorporated with intrusion detection systems. From a broad definition, an agent is a program module that functions continuously in a particular environment. It is able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment. An agent is able to learn from its experiences. The agent is autonomous. It takes actions based on its built-in knowledge and its past experiences [58].

Agents can be classified into four categories: 1) simple reflex agents, 2) agents that keep track of the world, 3) goal-based agents and 4) utility-based agents. The simple reflex agents perceive the input from their environment and interpret it as a state that matches their rules. The agents that keep track of the world maintain an internal state of past inputs because their actions need to occur in correlation with the past states and the new state. The goal-based agents need to know some information about their goals because the percepts (impression of the object obtained using the senses) do not provide enough information to determine the action to be taken. Sometimes knowing the goals is not sufficient for the agents to take the right action, especially when there are conflicting goals. As a result, the utility based agents map the percept states into numbers that determine how closely the goals were achieved [58].
The multi-agent system is a system that consists of multiple agents that can interact together to learn or to exchange experiences. To be flexible, the agents should achieve reactivity, pro-activeness and social ability. Reactivity means the agent can perceive its environment and respond, in a timely fashion, to changes that occur to satisfy its design objectives. Pro-activeness means that the agent is able to exhibit goal-directed behavior by taking the initiative in order to satisfy its design objectives. Social ability means that the agent can interact with other agents in order to satisfy its design objectives. The IDS consists of several agents working together. Since the attacks change everyday, the signatures also change. Thus, the agents must be able to learn the new signatures or detect the abnormal traffic resulting from the new attacks [58].

Botnet detection has been said to be most effective when it utilizes multi-agent systems that allow for botnet diagnosis to be made more logical through an agent’s communication within the network. Multi-agent systems for botnet detection require a generation of agents set with some structure and functionality. Based on the work by Savenko et al. [114], agents are utilized when their results are transmitted to the effectors as a means of influence on the computer system. If malware is detected, an agent, through effectors, blocks the processes that are responsible for performance of some malware and then notifies the user about the infection. The agent model ensures the integrity of the agent’s structure. It is realized by implementation of system checkpoints to provide the serviceability of the agent. Also, after each check of the agent, critical elements are stored for later restoration in case of virus attack on an antivirus multi-agent system, or possible failures in the computer system. Each agent can activate the recheck for the selected number of sensors to refine the results. In a situation when an agent cannot communicate with another agent it behaves as an autonomous unit and is able to detect different malware, relying on knowledge of the latest updates and corrections in the trusted software base. It is advisable to keep all the given values.

The use of a multi-agent approach in smartphone security with a focus on the Android
platform is a new area of research. Several multi-agent platforms have been developed for smartphones. According to the work by Savenko et al. [114], agents are utilized when their results are transmitted to the effectors as a means of influence on the computer system. If malware is detected, an agent, through effectors, blocks the processes that are responsible for performance of some malware and then notifies the user about the infection. Ágüero et al. [5] present the Agent Platform Independent Model (APIM) for mobile devices. Their proposed approach has been implemented and tested on the Android platform with some limitations. Carabelea et al. [30] present an overview of several multi-agent platforms that have been created for use in small devices that include mobile devices, and identify characteristics of these platforms. JaCa-Android [111] presents the use of Agent-Oriented Programming (AOP) technologies to develop smartphone applications for the Android platform. However, the approach has some weaknesses that need to be addressed, such as devising a notion of type for agents and artifacts, improving modularity in agent definition and improving integration with the Object-Oriented (OO) layer. Moreno et al. [82] explain the use of JADE-LEAP to implement personal agents on a mobile device. They developed a taxi service application for PDAs on which the personal agent communicates via Bluetooth with the rest of the multi-agent system.

In Context-Aware for Android, ConUCON proposed by Bai et al. [12] is used to enhance data protection and resource usage control on Android devices by leveraging context information. They also implemented a policy enforcement framework as a security mechanism. Frantz et al. [49] proposed a micro-agent framework for the Android platform that has the ability to interface with Android platform facilities in order to allow agent-based applications to access Android functionality. Alam et al. [7] presented a context-aware multi-agent based framework for securing Android devices by collecting Android data which is then analyzed in the central server. Cheng [34] proposed a multi-agent security system for the Android platform that uses agents to collect data from Android devices, and then send it to agent service providers to make a decision. The study shows that multi-agent systems
can be adapted to the Android platform with its inherent resource limitations.

2.6 Concluding Remarks

In this chapter, we provided a brief introduction to Android smartphone security, mobile botnets, intrusion detection systems, correlation techniques, and multi-agent systems that will be helpful in understanding our proposed framework to be presented in the next chapter. Also, as preparation for chapter 3, botnet detection techniques in related literature have been discussed; the rationale behind different techniques, their strengths and limitations. The previous research that used behaviour profile analysis, alert correlation, and multi-agent system in intrusion detection were also discussed and it was briefly explained how we propose to solve the identified problems or extend current studies in our proposed solution. The next chapter, chapter 3, presents the proposed solution as an SMS-based botnet detection framework.
Chapter 3

Proposed Framework

One of the main components of a botnet is the C&C channel, which is used to carry out communication with bots. With the availability of SMS on smartphones, SMS messages are used to send spam messages, to transfer command and control (C&C) instructions, to distribute malicious applications via links embedded in text messages, to send text messages to premium-rate numbers without user knowledge [107], and to exploit smartphones. SMS-based C&C presents some advantages for the bot master. The first advantage is that the attacker can communicate with the root node, given that communication utilizes a tree topology. The second advantage is that it makes detection of bot communications very difficult. On the other hand, SMS-based C&C poses some challenges for bot-masters. One of these is that the bot master must first assess whether a tree is still intact, ensuring that there are no breaks in the tree or missing nodes that might compromise the communication process. An additional challenge is that SMS-based C&C requires a node list to be operated on infected phones [84]. In the botnet, a C&C channel is considered to be the weakest link to the botnet, so losing a C&C channel means that the attacker will not be able to manage the botnet [55].

In order to address these issues and make it easier to detect SMS botnets, we propose a framework with two main components, a multi-agent system and an intrusion detection
The first component of this research involves a multi-agent system composed of multiple interacting agents. To reduce the chance of failure, a multi-agent system must be embedded, in which responsibilities are shared among several agents [124]. Agents have the ability to adjust their behavior in order to be alerted to resource constraints, such as network bandwidth and battery capacity [124]. Botnet detection is believed to be most effective when it utilizes a multi-agent system that makes botnet diagnosis more logical by means of an agent’s communication within the network. The proposed framework uses multi-agent system technology to aid in performing detection without exhausting smartphone resources such as battery and memory.

The second main component is an intrusion detection system with signature-based and anomaly-based detection modules. Traditionally, the signature-based approach extracts the features from traffic and detects malicious activities by comparing incoming traffic to the signatures of the attacks. Signatures are patterns or sets of rules that can uniquely identify an attack. This approach does not have the ability to detect all types of attack because these attacks do not all have corresponding signatures. The anomaly-based detection approach can also be referred to as behavior-based detection. This approach builds models of normal data and then attempts to detect the deviation from the norm in observed data; this deviation is considered an anomaly. The key advantage of this approach is that it can detect new types of threats when there are deviations from normal data [22]. A shortcoming is the cost of dealing with the large number of false alarms. To improve the capability of intrusion detection, researchers have proposed hybrid detection, which is a combination of signature-based detection and anomaly-based detection. This hybrid approach will detect known attacks with high accuracy using signature detection, while also being able to detect unknown attacks through the use of anomaly-based detection.

Intrusion detection in networks is one of the most well defined and extensively studied fields. Intrusion detection in mobile networks generally falls under the umbrella of this broader domain, and thus the core foundation of intrusion detection generally follows the
defined principles. At the same time, there are several specificities that make traditional IDSs not suitable for mobile devices:

- **Constrained resources**: the resource-constrained environment of smartphones puts strict limitations on the usage of the available time, power, and memory resources, essentially dictating what actions the detection system can and cannot afford. As such many of the approaches that require heavy computational operations (e.g., malware static analysis) are avoided.

- **Mobility**: As opposed to traditional IDSs where an IDS system is permanently stationed on a known network or a host, mobile device IDSs are generally located on a mobile device with some more resource intensive functionality residing on a “cloud”. Since a mobile device goes through a variety of networks with often unknown configurations and differing security measures, mobile IDS faces challenges to provide a comprehensive defense for the wide range of threats and conditions.

- **Deployment environment**: One of the security features characterizing modern mobile platforms is the use of a sandbox environment that allows them to constrain unwanted activity. Since a sandbox is meant to execute untrusted code, trusted native code is generally run directly on a platform. Although sandboxing is generally seen as a desirable intrusion detection technique, it has limitations. Sandboxing is usually less effective in detecting non-generic targeted attacks, e.g., malware designed to be activated upon specific user action or to trigger malicious behavior after a period of normal activity.

  Sandboxing is also largely ineffective against another practice, namely, the use of external code, that has been gaining popularity in mobile apps. This mechanism allows attackers to use legitimate applications to load malicious functionality without requiring any modifications to the existing legitimate code. As such the original bytecode remains intact, allowing an app to evade detection. Poeplau et al. [99]
defined several techniques to load external code onto a device: with the help of class
loaders that allow extraction of classes from files in arbitrary locations, through the
package context that makes it possible to access resources from other apps, through
the use of native code, with the help of runtime method execution that gives access
to a system shell, and through less stealthy installation of .apk files requested by a
main .apk file.

- **Exposure:** The typical attack vectors seen in traditional platforms also exist in mo-
bile environments. However, the specificity of mobile devices has opened up new
avenues for compromise. As such infection methods usually not monitored by tradi-
tional IDSs, e.g. through SMS, MMS, and app’ markets, have recently gained wide
popularity.

- **Privacy:** As opposed to traditional networks where privacy leakage typically con-
stitutes a small portion of potential threats, private information theft is rapidly
becoming one of the major concerns for mobile devices [70, 116].

In this thesis, we propose an SMS botnet detection system that uses multi-agent technol-
yogy, based on the observations of SMS and Android smartphone features. This system
will detect SMS botnets and identify ways to block the attacks in order to prevent damage
caused by these attacks. We have developed an adaptive hybrid model of SMS botnet de-
tectors by using a combination of signature-based and anomaly-based algorithms. These
components utilize multi-agent technology to recognize malicious SMS and prevent users
from opening these messages.

The architecture of the proposed detection system consists of four different components: a
multi-agent system, SMS signature-based detection, an anomaly-based detection module,
and a defence module. These components are divided into two tiers: Android mobile
devices and a central server. Figure 3.1 shows the proposed model design that will serve
as a comprehensive SMS botnet-detection mechanism. In the rest of this section, we
explain the multi-agent system in more detail, discuss the intrusion detection system that includes the SMS signature-based detection module, describe the main components of the anomaly-based detection module, and finally, illustrate the functions of the defence module.

3.1 Multi-Agent System Analysis

The SMS mobile botnet detection framework consists of two main systems: a multi-agent system and an intrusion detection system. The framework incorporates two components: Android mobile devices and a server that offers services. A multi-agent system has different agents with related responsibilities and goals to achieve. These agents are distributed between the two tiers. The intrusion detection system consists of three modules: an SMS signature detection module in Android mobile devices, an anomaly-based detection module
and a defence module based in the server. Figure 3.1 shows our complete framework design that functions as a comprehensive SMS botnet detection mechanism, and illustrates the interaction between agents and other modules.

A multi-agent system is a system composed of multiple interacting agents [144]. Our proposed framework requires a multi-agent system with extensive knowledge about distributed systems and required agent interactions in order to observe, monitor, and handle the data exchange. There are several multi-agent system frameworks in existence. Among the platforms in use, such as Agent-Builder [62], JACK [25], and Cougaar [156], one of the most well-known is the JADE platform [15]. The advantage of this platform is the provision of ready-to-use and easy-to-customize core functionalities. We selected the JADE platform for our multi-agent system on account of its features. The Foundation for Intelligent, Physical Agents (FIPA) has defined a list of specifications for agents [48]. Agent management and agent communication are at the core of this list, and need to be included in any multi-agent system framework in order to develop an agent platform [18]. An analysis of the multi-agent system is illustrated in the subsections that follow. The design and implementation of JADE agents will be described in chapter 4. This thesis follows the JADE methodology [87] for developing JADE agents.

The proposed framework, composed of multiple agents, is distributed on two levels: four types of agents in each subscribed Android mobile device and three types of agents in the service provider as shown in Figure 3.2. The basic idea behind the proposed multi-agent system is that JADE platform acts as a framework in which the main container is reside in the service provider. Agents live in containers which are the Java process that provides the Jade run-time and all services needed for hosting and executing agents. The JADE-based multi-agent system has the ability to perform the following tasks. The SMS mobile botnet detection framework allows an Android device user to install our application (app) in order to obtain all services. The agents monitor and observe Android device activities, and then capture and report suspicious behaviours to the detection part of the central
server. We also define each agent’s tasks and responsibilities in more detail below.

### 3.1.1 Scenario Analysis

The proposed framework includes the following task functions:

1. An Android user can install the SMS botnet detection application in order to protect his smartphone against SMS botnets.

2. An Android user must register with a service provider that can provide all the services and keep a list of all registered devices.

3. The service provider is responsible for maintaining the list of Android devices and offering the services to the subscribed Android devices.

4. An Android mobile device must check its phone status, including Internet connection, battery level, and network status, and report them to the service provider.
5. An Android device runs a signature detection algorithm to check the incoming and outgoing text messages, and then reports the suspicious and malicious SMS messages to the service provider. If a message is detected as malicious, a copy will be sent to the service provider, the message will be deleted from the Android device, and the Android user will be notified. If an SMS message is suspicious, it will be sent to the service provider, and the Android user will be notified. If the SMS message is normal, the SMS will be displayed to Android user.

6. The service provider must frequently update the signatures and send the update to the subscribed Android device in order to detect malicious SMS messages.

7. An Android device updates its current profile and sends information to the service provider with the user’s permission. The service provider keeps updated the user profiles for each Android device in order to perform further analysis.

8. The service provider has an anomaly-based detection module that will perform anomaly detection and profile analysis on the reported data. It also finds any correlations between the data, and accordingly makes recommendations to the Android device. The recommendations include but are not limited to updating the signature, maintaining the phone number blacklist, submitting the Android profile, removing a certain application, and reporting back information about a specific application. It is up to the mobile device user to take action based on the recommendations.

3.1.2 Use Case

The proposed framework monitors and observes Android device activities, and then captures and reports suspicious behaviours to the detection part of the server. Use cases illustrate a way of capturing all the possible functional requirements of a system, and depict the way a user interacts with our framework or other modules in order to achieve a specific goal. Figure 3.3 shows how the framework use case achieves a specific goal,
by interacting with the end user and other agents or systems, using the most popular specification, Unified Modeling Language (UML) [87].

### 3.1.3 Initial Agent Types Identification

Here, we identify the main agent types and subsequent formation of a first draft of the agent diagram design. All Android devices and service providers, modules make explicit the interaction between humans and external systems by a multi-agent system that is described in the agent diagram. In the agent diagram, a circle denotes an agent, squares represent resources and the use case actor represents the user. By applying JADE methodology rules [87] to the proposed framework, the initial diagram shown in Figure 3.4 is obtained.

In the proposed framework, the agents act as transducers that serve an interface between external/legacy systems and other agents in the framework [87]. Genesereth et al. [51] define three techniques used to account for external systems in the multi-agent system, namely, the use of a transducer agent, the insertion of a wrapper, and the rewriting of the code. The details of these techniques are extensively discussed in [51] [87]. Since the multi-agent system has to interact with the Android platform and other modules, the transducer approach is the most suitable for the SMS botnet detection framework.

### 3.1.4 Agent Responsibilities

The Agent responsibilities are defined from the use-case and the way the agents achieve their goals. The following two sections describe in details all agents and their responsibilities.

#### 3.1.4.1 Service Provider Agent Responsibilities

As shown in Figure 3.2, the service provider has three agents and two modules that are used to process the data, in order to detect SMS botnets and malicious SMS, to make intelligent
Figure 3.3: Use-case diagram for the proposed framework.
decisions, and to perform actions. The three major agents that perform the majority of the activities of the detection system are the central agent, the Android profiling agent and the SMS profiling agent, as outlined below. Based on the results of profiling analysis, these agents provide service and offer further analysis to achieve a high detection rate and make intelligent decisions, in order to detect SMS botnet activities. Table 3.1 illustrates all agent types with their relevant functionalities.

**Central Agent:** The central agent is in charge of handling and responding to smartphone requests and adding them to the subscriber list. It also performs activities that are relevant to Android mobile device agents, such as managing, updating, blocking, deleting and controlling. After an SMS message has been verified by the anomaly-based detection module, it is the central agent that sends the updated signature database to the Android
agents, and then sends commands to the Android agents, detailing the decision obtained in the defence module. The central agent manages all the local agents situated in the server, and obtains profile updates which it forwards on to the Android profiling agent. It also has to obtain copies of SMS messages and logs, and send them to the SMS profiling agent, which manages all suspicious SMS activity.

**SMS Profiling Agent:** The SMS profiling agent handles all the incoming and outgoing SMS that are considered to be suspicious, and maintains the SMS logs. It receives reports on suspicious SMS data and logs from the SMS detection agent, and then forwards them to the anomaly-based detection module to verify whether they are indeed deemed as botnets.

**Android Profiling Agent:** Once the profile updates have been received by other agents, this particular agent will then maintain and update the profile for all subscribing smartphones. Additionally, this agent makes updates based on the information it receives from the anomaly-based detection module and the other agents. It responds to anomaly-based detection module requests, which consist of findings and actions that need to be acted upon. Finally, this agent can request more information from the app-profile and user-profile agents on subscribed smartphones if required.
3.1.4.2 Android Smartphone Agents Responsibilities

An Android mobile user must subscribe to the central agent in order to obtain all the defined services and to maintain the interaction between local agents. The agents monitor incoming and outgoing SMS messages and send them to the SMS signature-based detection module. They also observe smartphones behaviour and resources. The function of the SMS signature-based detection module will be described extensively in the next section. There are four major agents within Android mobile devices, namely, an Android agent, an SMS signature detection agent, an app-profile agent, and a user-profile agent.

**Android Agent:** This is the agent that establishes a connection with the service provider by subscription. The Android agent plays a critical role in the system as it creates a channel that allows communication from the actual phone to the server. The Android agent obtains the agent identification from the service provider and establishes the interaction. It reads the status of the smartphone, including the Internet connection, SMS delivery, and battery power level. Basically, as the name states, the Android agent supervises the interactions between the local agents for Android mobile devices and is in charge of responding to any request received from the service provider. In addition, the agent can exchange data with the Android profiling agent. The Android mobile user can unsubscribe from the service provider at any time. The Android agent notifies the user when a new threat is detected.

**Signature Detection Agent:** The main task of this agent is to monitor incoming and outgoing SMS messages, then send requests to the SMS signature-based detection module to perform detection for malicious and normal SMS. This agent first registers itself with the SMS profiling agent, and then obtains updates on the SMS signature database. Additionally, this agent is responsible for monitoring SMS logs and reporting to the SMS profiling agent. This agent receives the results from the SMS signature-based detection module, and performs one of the following actions: deliver, delete, or send the suspicious
SMS to the SMS profiling agent.

**App-Profile Agent:** The app-profile agent registers with the Android profiling agent. This agent is responsible for observing phone settings and phone activities. The app-profile agent also plays a role in reporting accessibility in the browser and other installed applications. In addition, it checks the Internet connection since connectivity is an important factor that likely contributes to the realization of SMS botnet attacks. Reports regarding all the agent activities are then passed through to the Android agent.

**User-Profile Agent:** Malicious activities usually wait until the smartphone is in an ideal mode, or after reboot. The user-profile agent registers with the Android profiling agent. This agent is in charge of monitoring user connectivity time, maintaining the blacklist of phone numbers, reporting daily usage of the mobile phone and responding to the Android agent as required.

Table 3.2: Android smartphone agents and their responsibilities.

<table>
<thead>
<tr>
<th>Agent Type</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Android Agent</strong></td>
<td>1. Smartphone user must have subscribed to the central service provider.</td>
</tr>
<tr>
<td></td>
<td>2. Read smartphone status.</td>
</tr>
<tr>
<td></td>
<td>3. Obtain the agent identification from central service provider to establish the interaction.</td>
</tr>
<tr>
<td></td>
<td>4. Respond to requests from central agent.</td>
</tr>
<tr>
<td></td>
<td>5. Manage the interaction communication between local agents.</td>
</tr>
<tr>
<td></td>
<td>6. Send data to Android profiling agent.</td>
</tr>
<tr>
<td></td>
<td>7. Unsubscribe from central service provider.</td>
</tr>
<tr>
<td></td>
<td>8. Notify the user when new threat is detected.</td>
</tr>
<tr>
<td><strong>Signature Detection Agent</strong></td>
<td>1. Register with SMS profiling service in central server.</td>
</tr>
<tr>
<td></td>
<td>2. Obtain SMS signature update.</td>
</tr>
<tr>
<td></td>
<td>3. Read incoming and outgoing SMS.</td>
</tr>
<tr>
<td></td>
<td>4. Receive the result from SMS signature-based detection module.</td>
</tr>
<tr>
<td></td>
<td>5. Monitor SMS logs.</td>
</tr>
<tr>
<td></td>
<td>6. If SMS is normal, deliver it to SMS application.</td>
</tr>
<tr>
<td></td>
<td>7. If SMS is malicious, delete SMS and notify the user.</td>
</tr>
<tr>
<td></td>
<td>8. If SMS is suspicious, send a copy of suspicious SMS to SMS profiling agent.</td>
</tr>
<tr>
<td><strong>App-Profile Agent</strong></td>
<td>1. Register with Android profiling service in central server.</td>
</tr>
<tr>
<td></td>
<td>2. Report any access to browser or other apps when SMS application tries to access.</td>
</tr>
<tr>
<td></td>
<td>3. Check WiFi status and Internet access.</td>
</tr>
<tr>
<td></td>
<td>4. Monitor smartphone status including battery usage, apps that are running, memory usage, etc.</td>
</tr>
<tr>
<td></td>
<td>5. Spot any setting changes.</td>
</tr>
<tr>
<td><strong>User-Profile Agent</strong></td>
<td>1. Register with Android profiling service in central server.</td>
</tr>
<tr>
<td></td>
<td>2. Observe user connectivity time.</td>
</tr>
<tr>
<td></td>
<td>3. Maintain the whitelist and blacklist.</td>
</tr>
</tbody>
</table>
3.1.5 Agent-Resource Interactions

The seven types of agents in the proposed framework interact with each other, with external resources, the security administrator, and Android users as shown in Figure 3.5. Agents exchange data and send and receive requests among themselves. Agents that interact with external resources have been briefly introduced in Subsection 3.1.3, and are also illustrated in the agent diagram. These resources are classified into two main categories: inactive, and active resources, based on their status changing behaviour. The interactions between agents, and the interaction of agents with inactive and active resources, are outlined below.

3.1.5.1 Service Provider Agent-Resource Interactions

Three types of agents reside in the service provider: the central agent, the SMS profiling agent, and the Android profiling agent. These agents interact with active resources, i.e.,
with the anomaly-based detection module and defence module. The following interaction
design focuses on service provider resources, and gives more details of the design and
implementation provided in Chapter 4:

- The modules have the ability to query, insert, and modify data in the databases;
- The modules provide a listener-based interface so that the designated agents can
  immediately detect changes inside the resource;
- The modules provide the notification event that can be used with the listener im-
  plemented in order to inform the agents of new changes; and
- The agents receive “events” from the modules, which tell the agents to perform a
certain behaviour.

3.1.5.2 Smartphone Agent-Resource Interactions

Four type of agents run in smartphones, namely, the Android agent, the signature detec-
tion agent, the app-profile agent, and the user-profile agent. These agents interact with
inactive resources including smartphone status (such as device ID) and external storage
(such as the SD card). They also interact with active resources including SMS service,
the contact list, battery status, network status, setting changes, wake-lock status, and the
application manager. The Android agents interact with a remote server, with smartphone
resources, such as battery and phone status, and with an Android user. The signature
detection agents communicate with SMS applications and the SMS signature-based detec-
tion module. The app-profile agents interact with other applications, Internet access and
the Android agent, as well as with other agents and smartphone’s user. The user-profile
agents interact with smartphone resources, identifying smartphone status and the phone’s
wake-up time, for example, and observe user interaction with the smartphone.

The following features are common to all four smartphone agents:

- All the agents must keep a record of from where an application’s context was received.
• All agents have the ability to query the updated information via application context in the Android platform.

• Intent filters component added to the application activity in order to receive broadcasts of relevant system and other application intents such as SMSBroadcast.

• In the application activity, there are listener functions for obtaining Android system services such as ConnectionListener, TaskManager, SMSSManager, and ConnectivityManager.

• The application’s intent receives update information from the listeners once system status updates are detected.

• BroadcastReceiver in the application will process all the system events and updates from listeners, then send them to the agents via application intent broadcasting.

• Agents register their own BroadcastReceiver and intent filters to retrieve updates.

3.1.5.3 Agent-User Interactions

In many cases, the proposed framework requires that agents interact with users (Android users and security administrators). Agents interacting with users have been briefly introduced in Subsection 3.1.3 and are expressed in the agent diagram by an associate relation with an actor element. The graphical user interface (GUI) is a common type of user interface, and is the type that a JADE agent uses. A GUI can be viewed as an active resource. In the Android and the service provider, a GUI implemented using Swing displays the recommended action that the Android user should consider. The agent is used to achieve its goal, and the GUI to display results to the user. Both work with the same data, but they use this data in different ways. The GUI and the agent interactions will be discussed in more detail in Subsections 4.5.1 and 4.5.2 respectively.
3.1.6 Android User Profile Approach

In order to detect malicious applications that abuse SMS service, an Android user profile must perform profiling analysis, because the detection of malicious SMS messages is not enough. The main idea of this multi-agent approach is that the app-profile and user-profile agents stay proactive and monitor Android resources and application behaviour. Once a malicious or suspicious SMS message is identified by the signature detection agent, this agent will notify the Android agent, which will send a request to app-profile and user-profile agents to respond with an update of the current profiles. These profile agents will in turn send the collected data to the service provider agents. The profile analysis step involves obtaining data from the Android profiling and SMS profiling agents for a given period of time, collecting data for another period of time, and then comparing user usage based on the two. Also, it takes each Android user profile and compares the profile with other reported profiles. The system is looking for any deviation that can be recognized as abnormal behaviour. This abnormal behaviour is referred to as malicious activity. It also detects outgoing SMS messages that are sent without the user’s permission by comparing user connectivity time with the outgoing SMS time stamps and investigating the logs reported by SMS detection agents. The collected data used for our analysis is illustrated in Table 3.3. The profiling analysis process is discussed in more detail in Subsections 3.3.4.

3.2 SMS Signature-Based Detection Module

Focusing on incoming and outgoing SMS messages, the proposed design for Android mobile devices uses a signature-based detection algorithm to identify SMS botnets. As illustrated in Figure 3.6, SMS signatures are obtained and copied from an SMS signature database where signatures of known botnets and malware are stored.

We employ a content-based mechanism using a signature-based approach. Signatures are
Table 3.3: Collected features for Android user profiling.

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious SMS</td>
<td>Detected malicious SMS are added with time stamp of the detection time.</td>
</tr>
<tr>
<td>Suspicious SMS</td>
<td>Detected suspicious SMS are added with time stamp of the detection time.</td>
</tr>
<tr>
<td>SMS Logs</td>
<td>Monitor SMS log and spot any changes.</td>
</tr>
<tr>
<td>Installed Applications</td>
<td>List of installed applications on Android mobile device.</td>
</tr>
<tr>
<td>Running Applications</td>
<td>List of current running applications.</td>
</tr>
<tr>
<td>Granted Permissions</td>
<td>List of permissions for running application.</td>
</tr>
<tr>
<td>Running Services</td>
<td>List of running services.</td>
</tr>
<tr>
<td>Network Information</td>
<td>List of all network information including available, connected or connecting, state, detailed state, and other information.</td>
</tr>
<tr>
<td>Browser Accessibility</td>
<td>Keep track of browser usages.</td>
</tr>
<tr>
<td>Application Interactions</td>
<td>Monitor the communication between the applications.</td>
</tr>
<tr>
<td>Battery Usage</td>
<td>Monitor battery usage, including daily battery usage.</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>Monitor memory usage, including daily memory usage.</td>
</tr>
<tr>
<td>CPU Usage</td>
<td>Monitor CPU usage, including the CPU daily usage.</td>
</tr>
<tr>
<td>Setting Changes</td>
<td>Keep track of phone setting changes.</td>
</tr>
<tr>
<td>Connectivity Times</td>
<td>Observe user connectivity time.</td>
</tr>
<tr>
<td>User Usages</td>
<td>Track user’s use of phone, including time user usually uses the phone to call and send SMS.</td>
</tr>
<tr>
<td>Contact Blacklist</td>
<td>Keep track of contact blacklist.</td>
</tr>
<tr>
<td>Contact Whitelist</td>
<td>Keep track of contact whitelist.</td>
</tr>
</tbody>
</table>

patterns or sets of rules that can uniquely identify an attack. Traditionally, the signature-based approach extracts the features from traffic and detects malicious activity by comparing incoming traffic to the signatures of attacks. The advantages and disadvantages of this approach are extensively discussed by Debar et al. [39]. Because this approach involves frequent updating of attack signatures, signature-based detection has two main advantages: exceptional accuracy and production of very few false alarms. The main disadvantage of signature detection is its inability to detect an unknown attack, for example an attack that has not been seen before, or an attack that does not have a corresponding signature. In order to address this issue, our approach labels an unknown attack as either suspicious or normal using rule-based techniques. We use a real-time content-based signature detection to differentiate between normal, suspicious and malicious SMS based on the content of the SMS, and then display the result of the suspicious SMS to the Android user, who may then choose whether to remove the SMS or not.

Once SMS signatures are obtained, the SMS signature-based detection module scans for
malicious messages. If scanning reveals detection of a known botnet, then the detection algorithm performs signature detection on the obtained SMS, cross-referencing it with the known SMS signatures before the message is passed to the SMS application. If the message is determined to be malicious, as found by the SMS detection process, the mobile user is notified and the message is automatically deleted. If the SMS is found to be suspicious, the user has the option to remove the SMS. If the SMS is determined to be normal, the SMS message is shown to the user. Our approach provides extra information along with notification of suspicious SMS, to aid the user in making the right decision.

3.2.1 Signature-Based Detection

As shown in Figure 3.6, the first step to effectively spot malicious SMS is to extract SMS features that have the potential to distinguish the behaviour of SMS text messages. The names and descriptions of these features are given in Table 3.4. All the selected features share three significant characteristics:

1. They have been shown to be effective in distinguishing between types of SMS messages, whether normal or malicious;

2. They can be used in real time, and impose no delay on the anomaly-based detection module; and

3. They keep our detection approach simple and fast.

In order to process the SMS message, we have implemented an SMS Feature Extractor to extract the selected features of the incoming and outgoing SMS messages, and then pass these features for detection by the detection algorithms. If the SMS is malicious or suspicious, a time stamp is added to the results from the SMS signature-based algorithm, which then reconstructs the SMS. The result will be shown to the user and the user can make a decision whether to delete the suspicious SMS or not. Four techniques have been used to implement signature-based detection, namely: pattern-matching, data-mining,
Figure 3.6: Structure of SMS signature-based detection module.
rule-based and statistical-based techniques. The following is an explanation of the pattern-matching and rule-based techniques used in our approach.

Table 3.4: Features selected for SMS signature-based detection

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromPhone#</td>
<td>sender phone number</td>
</tr>
<tr>
<td>ToPhone#</td>
<td>recipient phone number</td>
</tr>
<tr>
<td>URLs</td>
<td>links within SMS message</td>
</tr>
<tr>
<td>Command</td>
<td>specific words</td>
</tr>
<tr>
<td>Phones#</td>
<td>phone number in SMS content</td>
</tr>
<tr>
<td>Content</td>
<td>SMS text</td>
</tr>
<tr>
<td>Phishing Words</td>
<td>words used with malicious content</td>
</tr>
</tbody>
</table>

3.2.1.1 Pattern-Matching Techniques

Pattern-matching techniques are used to identify and match a perceived sequence of text with its signature. Researchers have used this approach in both network-based and host-based intrusion detection. In host-based detection systems, pattern-matching can check the words represented in the system call. We have adapted this approach to detect malicious SMS by looking for specific patterns of selected features as shown in Figure 3.6. The main reason for choosing pattern matching is its ability to reduce search space, resulting in faster searches \[53\]. A negative side of the technique is that a growing number of signatures has caused there to be high costs associated with pattern-matching. This issue has been addressed in Abbes et al. \[1\].

3.2.1.2 Rule-Based Techniques

Rule-based techniques are a defined set of rules that can be applied against text. Any match of the rule is reported as abnormal. Rule-based expert systems are one of the early techniques used for signature detection. After applying pattern-matching techniques, rule-based techniques are utilized to spot suspicious behaviour, by applying a set of rules to ascertain whether the incoming or outgoing SMS is abnormal. For instance, the set of rules
may define an attack scenario involving some unusual SMS content. This will determine whether the SMS has abnormal content. This also explains that unknown behaviour does not necessarily represent abnormal behaviour \[53\]. One advantage of using rule-based detection is that the set of rules helps to produce fewer false positive alarms \[66\]; it also has the ability to label unknown attacks.

3.2.2 Signature-Based Detection Algorithm

Signature-based detection needs to be provided with up-to-date signatures of known botnets and malicious SMS. In order to develop an effective signature-based detection approach to combat malicious malware, we extract sender phone numbers and SMS content from our dataset; then, from the SMS content, we extract embedded URLs, commands, phone numbers, and phishing words as signatures for our approach. The signature-based detection algorithm is illustrated in Algorithm 1. The signature-based detection algorithm initially compares the selected features of a given SMS message (FromPhone#, ToPhone#, and Content) with provided signatures, and, if there is a match, the SMS is blocked. However, if the selected features do not match any of the signatures, the algorithm goes deeper and analyzes the body of the SMS. We extracted URLs, phone numbers, and commands by finding token strings in each SMS text body and matching them against the defined signatures.

As some attackers use obfuscation techniques to avoid detection, we define three variable patterns to match obfuscated URLs, phone numbers, and commands using regular expression. If the SMS text has a URL, command, or phone number, the algorithm matches it against provided signatures. If there is a match, the SMS is blocked, but if there is no match, more evidence is sought to classify the SMS by applying rule-based techniques. A set of rules is then applied for unknown SMS as defined in Table 3.5. If the SMS matches a rule, it is classified as suspicious. Otherwise, it is considered normal. Finally, the output from the SMS signature-based detection algorithm labels the SMS as normal, suspicious,
Algorithm 1 : SMS signature-based detection algorithm

Inputs

\[ \text{sms} \leftarrow \text{sms messages}, \ spw \leftarrow \text{phishing words list} \]
\[ \text{fb} \leftarrow \text{SMS body}, \ \text{ffnum} \leftarrow \text{FromPhone number}, \ \text{ftnum} \leftarrow \text{ToPhone number} \]
\[ \text{sc} \leftarrow \text{sms content signatures}, \ \text{sn} \leftarrow \text{phone numbers signatures} \]
\[ \text{surls} \leftarrow \text{urls signatures}, \ \text{scmd} \leftarrow \text{commands signatures}, \]
\[ \text{pnum} \leftarrow \text{phone numbers patterns, purls} \leftarrow \text{urls patterns} \]
\[ \text{pcmd} \leftarrow \text{commands patterns} \]

Outputs

\[ \text{malicious} \leftarrow \emptyset, \ \text{suspicious} \leftarrow \emptyset, \ \text{normal} \leftarrow \emptyset \]

for each \( \text{sms} \in n \) do

\[ \text{if} \ \text{fb} = \text{sc}, \ \text{then} \]
\[ \text{malicious} \leftarrow \text{sms} \]

\[ \text{else if} \ \text{ffnum} = \text{sn} \lor \text{ftnum} = \text{sn} \] then
\[ \text{malicious} \leftarrow \text{sms} \]

\[ \text{else} \]
\[ \text{L(sms)} \leftarrow \text{length of sms} \]
\[ \text{for each } \text{t} \in \text{sms} \text{ do} \quad \text{▷ split sms content into tokens t} \]
\[ \text{if } \text{t} = \text{surls} \text{ then} \]
\[ \text{malicious} \leftarrow \text{t} \]

\[ \text{else if } \text{t} = \text{sn} \text{ then} \]
\[ \text{malicious} \leftarrow \text{t} \]

\[ \text{else if } \text{t} = \text{scmd} \text{ then} \]
\[ \text{malicious} \leftarrow \text{t} \]

\[ \text{else if } \text{L(sms)} = 1 \text{ then} \]
\[ \text{suspicious} \leftarrow \text{sms} \]
\[ \text{if } \text{t} = \text{spw} \text{ then} \quad \text{▷ call phishing_words function} \]
\[ \text{suspicious} \leftarrow \text{sms} \]
\[ \text{suspicious} \leftarrow \text{sms} \]
\[ \text{if } \text{sms} = \text{purls} \text{ then} \]
\[ \text{has-url} \leftarrow \text{purls} \]
\[ \text{if } \text{has-url} = \text{surls} \text{ then} \]
\[ \text{malicious} \leftarrow \text{sms} \]
\[ \text{else} \]
\[ \text{suspicious} \leftarrow \text{sms} \quad \text{▷ call phishing_words function} \]

\[ \text{else if } \text{sms} = \text{pnum} \text{ then} \]
\[ \text{has-num} \leftarrow \text{pnum} \]
\[ \text{if } \text{has-num} = \text{sn} \text{ then} \]
\[ \text{malicious} \leftarrow \text{sms} \]
\[ \text{else} \]
\[ \text{suspicious} \leftarrow \text{sms} \quad \text{▷ call phishing_words function} \]

\[ \text{else if } \text{sms} = \text{pcmd} \text{ then} \]
\[ \text{has-cmd} \leftarrow \text{pcmd} \]
\[ \text{if } \text{has-cmd} = \text{scmd} \text{ then} \]
\[ \text{malicious} \leftarrow \text{sms} \]
\[ \text{else} \]
\[ \text{suspicious} \leftarrow \text{sms} \quad \text{▷ call phishing_words function} \]

\[ \text{else} \]
\[ \text{normal} \leftarrow \text{sms} \]
or malicious. If the SMS message is malicious, it will be deleted; however, if the SMS message is suspicious, it will be shown to the user with some information related to the detected SMS. If the SMS message is normal, a message to this effect will be displayed for the user.

Table 3.5: Suspicious SMS rules.

| R1   | SMS has a link to APK file             |
| R2   | SMS has a URL and the sender is not in the contact list |
| R3   | SMS has a command and the sender is not in the contact list |
| R4   | SMS has a phone# and the sender is not in the contact list |
| R5   | SMS has a phishing word and the sender is not in the contact list |
| R6   | SMS has a URL and a phishing word      |
| R7   | SMS has a URL and a phishing word and the sender is not in the contact list |
| R8   | SMS has a command and a phishing word  |
| R9   | SMS has a command, a phishing word and the sender is not in the contact list |
| R10  | SMS has a phone# and a phishing word   |
| R11  | SMS has a phone#, a phishing word and the sender is not in the contact list |
| R12  | SMS has a URL, a phishing word and other malicious app running |
| R13  | SMS has a command and other malicious app running |
| R14  | SMS has a phone number and other malicious application running |

3.3 SMS Anomaly-Based Detection Module

Although a large number of content and non-content features have been proposed and employed for the detection of a variety of SMS spam, an effective approach for detecting SMS botnets has not yet been found because analyzing SMS messages alone is not enough for this purpose. Our approach requires that an application be installed on user smartphones, to perform real-time signature detection and monitor smartphones’ behaviour, in order to build Android profiles. The signature detection results are then sent to our anomaly-based detection module. In this section, we describe our proposed anomaly-based detection module, which employs a new technique to recognize SMS botnets and malicious text messages, by considering Android profile behaviours.

The anomaly-based detection module consists of five components. The architecture of the
Figure 3.7: The architecture of anomaly-based detection module.
anomaly-based detection module is shown in Figure 3.7. The first component is an SMS and profile collection, which is responsible for receiving, combining, storing and retrieving data. The second component is SMS clustering, which groups SMS messages based on their similarity. The third component is SMS classification that classifies an SMS text message to one of the class labels. The fourth component employs profile analysis on each of the reported Android profiles. The final component is SMS correlation that applies rule-based correlation techniques to identify whether there are correlations between outputs from the class labels and any abnormal activities in Android profiles.

In summary, the anomaly-based detection module is where SMS collection, anomaly detection and behaviour-profiling analysis are conducted. Once the anomaly-based detection module receives the reported SMS from the android device, it performs anomaly detection through specifically-created and manipulated algorithms. Once SMS messages are deemed malicious, content analysis is performed in order to identify the type of attack. All the Android mobile profiles that contain the same SMS are grouped together. Details of these processes are outlined below.

The basic steps of our anomaly detection approach are as follows: The pre-process step takes the labelled datasets and applies the “stop words removal” and “stem words” functions. Afterward, the clustering step uses an X-Mean algorithm that assigns SMS messages to a number of clusters, such that each cluster has similar distances between the instances. The clustering analysis step then analyzes the output of the clusters, and groups them into four class labels. In the SMS classification step, only the reported SMS messages from the Android devices are input into the classifier. The classifier assigns each reported SMS message to one of the four class labels. In this step, we verify each class label to confirm that the messages are correctly classified. This step is repeated until all SMS messages have been classified. Next, the profile analysis step carries out profiling analysis on the reported profiles. Finally, the SMS correlation step is employed to draw rule-based correlations between the four class labels and profile outputs, in order to label the SMS
messages as either normal or malicious and to identify SMS botnets. The details are outlined below.

1. **Pre-process Step:** Take the labelled datasets and apply the “stop words removal” and “stem words” functions.

2.1. **Clustering Step:** Use an X-Mean algorithm to assign SMS messages to clusters, such that each cluster has the same distances between the instances.

2.2. **Clustering Analysis Step:** Analyze the output of the clusters and group them into four label classes: ‘all malicious’, ‘majority malicious’, ‘all normal’, and ‘majority normal’.

3 **Classification Step:** Here the input to the classifier is only the reported SMS messages from Android devices. Classify the reported SMS messages into one of the four class labels, and verify each class label to confirm that the message is correctly classified.

3.2 **SMS Classification Step:** Repeat step 3 until all SMS messages have been classified.

4. **Profile Analysis Step:** Carry out profiling analysis on the reported profiles and produce profiles outputs.

5. **SMS Correlation Step:** Apply rule-based correlations to the four class labels, to label SMS messages either “normal” or “malicious”.

3.3.1 **SMS and Profiles Collection**

The SMS and profiles collection is where input data is stored. There are three types of input sources, namely, labelled SMS datasets, reported text messages, and reported Android profiles. The SMS profiles collection is responsible for collecting, combining, storing, retrieving and managing this data, to allow for more robust detection. There
Table 3.6: Reported SMS features.

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeofSMS</td>
<td>label SMS as incoming and outgoing</td>
</tr>
<tr>
<td>FromPhone#</td>
<td>sender phone number</td>
</tr>
<tr>
<td>ToPhone#</td>
<td>receiver phone number</td>
</tr>
<tr>
<td>Text</td>
<td>SMS text</td>
</tr>
<tr>
<td>URLs</td>
<td>links within SMS message</td>
</tr>
<tr>
<td>Command</td>
<td>botnet C&amp;C instructions</td>
</tr>
<tr>
<td>Phone#</td>
<td>phone number in SMS content</td>
</tr>
<tr>
<td>AgentID</td>
<td>identify an agent who delivered SMS</td>
</tr>
<tr>
<td>Timestamp</td>
<td>time when profile received</td>
</tr>
<tr>
<td>In_contact_list</td>
<td>is the FromPhone# in contact list</td>
</tr>
</tbody>
</table>

are no other existing tools used to capture SMS messages and smartphones’ behaviour. We have implemented a real-time signature detection method in Section 3.2 to detect SMS botnets in smartphones, and to report SMS messages and profiles that need further analysis. A set of the features, as listed in Table 3.6 is each reported SMS messages, along with a time stamp. The third input source to our detection, Android profiles have a set of features as listed in Table 3.7. The Android profiles are collected in smartphone devices and are reported whenever a suspicious or malicious SMS is detected.

3.3.1.1 Term Frequency-Inverse Document Frequency

The TF-IDF is a statistical-based approach that measures the score of the important words in a document, based upon how frequently the words occur across multiple documents. We have adopted the TF-IDF approach in order to improve the quality of the anomaly-based detection module. The TF-IDF value is a weight that has been used in systems such as search engines, information retrieving systems, and text mining applications. Before starting the clustering steps, SMS messages require a preprocessing step that has to be applied to all text messages. The first part of the pre-processing step is called tokenization, in which each SMS message is treated as a string and partitioned into a list of tokens. The second part is to “remove stop words”, which are a set of words commonly used in
a given language. The third part is to stem the words, a process used in indexing and search techniques. The primary goal of stemming is to reduce the words to their word root, by removing any attached suffixes and prefixes (affixes) from index words. This pre-processing step produces the “terms” that are used to calculate the TF-IDF score. In order to compute the TF-IDF score

\[ TFIDF = TF \times IDF \quad (3.1) \]

two measurements are required:

1. Calculate the Term Frequency

\[ TF = \frac{C}{T} \quad (3.2) \]

where C is the number of occurrences of the term in a document, divided by T, which is the total number of terms in that document.

2. Calculate the Inverse Document Frequency

\[ IDF = \frac{D}{DF} \quad (3.3) \]

where D is number of the documents in the corpus divided by DF, the number of documents in which the specific word or term occurs.

The score of tf-idf is used to produce a vector that represents documents in which each item of a vector corresponds to the tf-idf value of an individual term in a dataset [80]. The terms that do not occur in a document are scored zero. This representation as a common vector space is called vector space model [110].

### 3.3.1.2 Measuring similarity between two text messages

There are two main types of measures used to determine whether two objects are similar in text documents. The first method is a clustering technique that uses a variety of distance
metrics to determine the similarity or dissimilarity between any two strings of a given type, such as Euclidean, Manhattan, Canberra, and Tchebyche, to name a few. These metrics and similarity measurements are extensively discussed in Pandit et al. [93]. The second most popular similarity measuring technique that is applied to text documents is the Cosine Metric, which is used in a variety of information retrieval applications [73]. We use the clustering technique in our SMS clustering components, and the cosine similarity metric in SMS classification components.

When SMS messages are represented as term vectors, the similarity of two messages matches the relationship between two vectors. This is called cosine similarity. Cosine similarity measures the similarity between two vectors by computing the cosine of the angle between them. When the angle is 0, the cosine is equal to 1, and the cosine of any other angle is less than 1. As the angle between two vectors shortens, the cosine similarity approaches 1, meaning that the vectors get closer and the similarity of whatever is represented by the vectors increases. Assuming the features of two vectors, $A$ and $B$, the cosine similarity, $\theta$, is denoted by means of a dot product and magnitude as

$$\text{sim}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \quad (3.4)$$

where $A$ and $B$ are the TF vectors over the term set $T = t_1, ..., t_m$. Each vector represents a term with its weight in the message, which is non-negative. As a result, the cosine similarity uses the positive value bounded in $[0,1]$. An important characteristic of cosine similarity is its ability to normalize the SMS text message length during comparison.

### 3.3.2 SMS Clustering

The advantage of using the clustering technique is that it provides a logical summary of the collected data in terms of text-clusters [73]. It can be used to offer a summary understanding of the complete content of the underlying dataset.
3.3.2.1 SMS Clustering Algorithm

Unsupervised learning has benefited from significant efforts in pattern recognition and machine learning. Take, for instance, clustering. The main idea of clustering is to take a set of data and group its contents based on their similarities. The cluster method does not require class labelling of the data. There are several types of clustering methods, including hierarchical clustering, density-based clustering, and partition-based methods. One of the clustering approaches, the partition-based method, comprises two different algorithms, the K-means algorithm and the X-means algorithm.

The unsupervised clustering algorithm that is used for SMS botnet detection is X-means clustering. The X-means clustering algorithm is based on K-means, but the number of clusters is found dynamically. The X-means algorithm consists of the following two steps recurring until completion: step one is to improve parameters by applying K-means to the initial dataset, with k alternating from kMin to kMax. Step two is to improve the structure by applying k-means with k = 2 to each cluster obtained in step one (“parents”) and to calculate the BIC score for parent (k=1) and child clusters. The situation with the highest BIC score is considered best [98]. Algorithm 2 shows the details of our SMS clustering algorithm that includes a summary of X-means proposed by [64], as well as with our own cluster analysis method.

Given a set of SMS messages, S, to be clustered into X number of groups and an N x N distance matrix, we must begin by randomly initializing the first cluster center that is selected from among the data points that are being clustered. This cluster center, called the centroid, assigns each SMS to the closest center cluster, and then follows an iteration of the following steps until it gets to its stable status: first, determine the centroid coordinate, then determine the distance of each object from the centroid, and finally, group the object based on minimum distance (i.e., find the closest centroid). After all input SMS messages have been assigned, the new mean value is updated by calculating each centroid and then the SMS are re-assigned to clusters. This reassigning and updating continues until each
Algorithm 2 : SMS clustering algorithm

Inputs
$S \leftarrow \{S_1, ..., S_n\}$ sms labelled datasets

Outputs
$n \leftarrow \emptyset$, $mn \leftarrow \emptyset$, $m \leftarrow \emptyset$, $mm \leftarrow \emptyset$

Phase A: Apply X-means algorithm

Step 0: pre-process N-dimensional data whose sample size is $n$
Step 1: set an initial number of clusters to be $k_0$
Step 2: apply K-means to all data with setting $K = K_0$
Step 3: repeat the following procedure from step 4 to step 8 by setting $i = 1, 2, ..., k_0$
for a cluster of $C_i$ do
  step 4: apply k-means by setting $k = 2$
  step 5: calculate the BIC
  if $BIC > BIC'$ then
    step 6: push the cluster centers, and the BIC onto the stack
  if $BIC = BIC'$ then
    step 7: clusters are no longer divided
    extract the stacked data in step 6, and Return to step 3
    if the stack is empty, go to step 6
  step 8: the procedure for $C_i$ is completed
  step 9: the procedure for initial $k_0$ divided clusters is completed
step 10: output the cluster identification number to which each instance is allocated
step 11: correspond $S_i \in C_i$ to its ordinal label

Phase B: Cluster results analyzer

for a cluster of $C_i$ do
  if $S_i \in C_i$ are normal then
    $n \leftarrow C_i$
    if # of normal $S \in x_i >$ # of malicious $S \in C_i$, then
      $mn \leftarrow C_i$
  if $S \in C_i$ are malicious then
    $m \leftarrow C_i$
    if # of malicious $S \in C_i >$ # of normal $S \in C_i$, then
      $mm \leftarrow C_i$
    if # of malicious $S \in C_i ==$ # of normal $S \in C_i$, then
      if $S \in C_i$ has url, phone#, or command, then
        $mm \leftarrow C_i$
      else
        $mn \leftarrow C_i$

SMS message is in the correct cluster. For a given data set and the initial seeds, the generated clusters are locally optimal. Different final partitions can be a result of how initial seeds are decided \cite{61}. Approaches for selecting good starting points have been addressed by Arthur at el. \cite{11}. The main focus of this thesis, however, is not to optimize the clustering. The X-means cluster works by means of a distance measurement that aims to minimize the cluster distances. Therefore, similarity measures do not directly fit into
the algorithm, because small distances mean dissimilarity. Because cosine similarity is
bounded in [0,1], we convert the similarity measures to distance values by taking \( D = 1 - \text{sim} \) as corresponding distance values.

The data set that is used for the cluster is a combination of malicious and legitimate SMS
messages. As an output of the clustering, a number of clusters will have different kinds of
messages. We analyze the result of clusters and group them into four class labels:

1. The first class is called ‘\textbf{All Malicious}’, which consist of only malicious SMS;

2. The second is ‘\textbf{Majority Malicious}’, in which the majority of the text messages
   are malicious messages;

3. The third is ‘\textbf{All Normal}’, that contain all legitimate text messages; and

4. The fourth is ‘\textbf{Majority Normal}’, where the majority of text messages are legiti-
   mate.

In the case of a cluster where 50\% of the SMS messages are legitimate SMS messages and
50\% of the SMS messages are malicious SMS messages, we analyze the content of the SMS
messages. If the SMS messages have embedded URL, phone number, or command, the
cluster will be grouped as ‘\textbf{Majority Malicious}’. Otherwise, the cluster will be grouped
as ‘\textbf{Majority Normal}’. At this point, we can label all SMS messages that belong to the
‘\textbf{All Malicious}’ class as malicious text messages but ‘\textbf{Majority Malicious}’, ‘\textbf{All Normal}’
and ‘\textbf{Majority Normal}’ will require further analysis.

\textbf{3.3.3 SMS Classification}

Although clustering is fundamentally an unsupervised learning approach, the clustering
 technique can also be used to increase the classification accuracy of supervised detection [4].

In order to classify the reported SMS to one of the four class labels list in part 3.3.2, the
following method is used.
3.3.3.1 SMS Classification Algorithm

Algorithm 3 describes the SMS classification algorithm. In this stage, when new suspicious SMS messages are reported to the anomaly-based detection module, we begin by pre-processing each one. After that, we take each text message and add it to all the class labels. For each class label, we calculate the TF-IDF weight, then apply the cosine similarity method to measure the similarity of the text message to each group by calculating the mean of each group. We find the minimum mean vector among the four class labels, and assign the text message to that class. After that, we remove the SMS message from other class labels, and update the class labels. We consider that all the SMS messages in ‘All Malicious’ and ‘Majority Malicious’ classes are malicious SMS, and we check the majority malicious class to look for any misclassification. If no misclassification is found, we then give a reason why the message is labelled as malicious. In order to verify the SMS messages in ‘All Normal’ and ‘Majority Normal’ classes, a further analysis is required, with additional information, to make a decision about the reported messages. The four class labels will then be sent to SMS correlation components.

3.3.4 Android Profiling Analysis

In order to provide accurate detection of SMS botnets, certain detection techniques are commonly used in Intrusion Detection Systems (IDSs). One of the techniques that contributes to detection is a behaviour-based analysis technique that is employed to find proof of compromise rather than any specific attack. Android profiling analysis is used to analyze the reported profiles from Android smartphone devices based on the features shown in Table 3.6 to detect outgoing SMS sent without user knowledge, and to perform further investigation of reported SMS to find the similarity between the reported profiles. The definition of the Android profiles here is the combination of the SMS-profile, app-profile, and user-profile for each subscribed smartphone. To build an Android profile, we extracted the features that are related to SMS botnet behaviours. The profiles are collected
Algorithm 3: SMS classification algorithm

**Inputs**

\( \text{rsms} \leftarrow \{ \text{rsms}_1, \ldots, \text{rsms}_n \} \) \text{ reported sms messages}  
\( \text{cl} \leftarrow \{ n, mn, m, mm \} \)  
\( \text{Initialize(Scores[rsms \in cl])} \)

**Outputs**

\( \text{cl}(\text{cn}, \text{clmn}, \text{clm}, \text{clmm}) \leftarrow \text{rsms}^k \)

**Method**

\( \text{pre-process rsms}^k \)  
for each \( \text{rsms}_i \in \text{rsms} \) do  
    add \( \text{rsms}_i \) to \( \text{cl}_i^3 \)  
    for class \( i \) in \( \text{cl} \) : \( \text{cl}_i \in \text{cl} \) do  
        for each term \( t \in \text{cl}_i \) do  
            \( \text{cl} \leftarrow \text{fetchpositionlist}(t) \)  
            \( \text{df}_i \leftarrow \text{getsmplist}(\text{cl}) \)  
            \( \text{wt}_{rsm} \leftarrow \text{computeclassestlabel}(t, \text{rsms}, \text{df}_i) \)  
            for each \( \text{rsms}, tf_{rsm} \in \text{cl}_i \) do  
                \( \text{Scores[rsms]} + = \text{wt}_{rsm} \times \text{weightincl}(t, \text{rsms}, \text{df}_i) \)  
        calculate mean of the \( \text{Scores[rsms]} \)  
        \( \text{d[cl]} = 1 - \text{mean(Scores[rsms])} \)  
        find min(\( \text{d[cl]} \))  
        min(\( \text{d[cl]} \)) \leftarrow \text{rsms}  
        remove \( \text{rsm} \) from other \( \text{cl} \)  
    Repeat the procedure until all \( \text{rsm} \) are assigned to the right \( \text{cl} \).

In smartphones and then reported these profiles to service provider by the app-profile and the user-profile agents.

In an Android platform, an SMS botnet wishing to infect a smartphone must trick an Android user into installing a malicious application that can receive commands from a C&C server through SMS text messages. To detect the SMS botnet, it is important to monitor all installed applications, running applications, granted permissions for the running applications, and running services. We take an extra precaution by keeping track of

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installed Applications</td>
<td>List of current installed applications.</td>
</tr>
<tr>
<td>Running Applications</td>
<td>List of running applications.</td>
</tr>
<tr>
<td>Granted Permissions</td>
<td>List of current application permissions.</td>
</tr>
<tr>
<td>Running Services</td>
<td>List of current running services.</td>
</tr>
<tr>
<td>Browser Accessibility</td>
<td>Keep track of browser usages.</td>
</tr>
<tr>
<td>Connectivity Times</td>
<td>Observe user connectivity time.</td>
</tr>
</tbody>
</table>
browser usages, in order to prevent any communication with the bot through the browser. One botnet behaviour is to send out SMS messages at premium-rate without the user’s knowledge. We are able to spot this behaviour by observing user connectivity times and SMS logs. In Android devices, permissions are used to notify the user of what activities will be carried out and which resources an application will gain access to. The basis of the profile analysis component is a form of profile aggregation which accepts profiles as input and produces high level attack scenarios as output, displaying the results in an Abnormal Profile Table (APT), as shown in Figure 3.8.

![Profile behaviour analysis diagram.](image)

3.3.4.1 Profile Aggregation

A higher level of management is required for analyzing Android profiles before correlating them with SMS messages in the four class labels. This is needed for two main reasons.
First, it is hard to detect botnets by only analyzing and monitoring SMS messages; therefore, it helps to spot abnormal activities by analyzing the behaviour of the Android devices. Second, the reported Android profiles are considered raw profiles; therefore, it is important to find the logical and static connections between the profiles. To overcome these problems, recently there have been significant contributions made in both academic and industrial communities in something called “alert aggregation” a technique that can be adopted to aggregate Android profiles.

The main idea behind profile aggregation is to group all similar profiles together. Studies differ in their criteria for alert aggregation; for example in some studies, alerts are considered similar to each other if they have the same attributes, while in others similarity may be based on several attributes. Alert aggregation has been recognized to be highly effective in decreasing the alert volume. In Qin and Lee’s [102], they measured the similarity of alerts based on the similarity between alert attributes. The aggregation of profiles takes into account the similarity between particular profile features. Similarity between values of each feature (e.g. Android_ID, FromPhone#, ToPhone#, sending_time, received_time, URLs, Command, Phones#, Content, Phishing Words, contact_list, dangerous permissions, services, connectivity time) has been well-defined based on the characteristics of each feature. What alert aggregation is looking for is any deviation that can be recognized as abnormal behaviour. This abnormal behaviour is referred to as malicious activity. For instance, one of the attributes is ‘sender phone number’ (‘FromPhone#’). We would examine in connection with the text messages sent from that number, the percentage of devices that have reported the same suspicious phone number. Similarly, we compare user connectivity time with outgoing SMS time stamps by investigating the logs reported by SMS detection agents.
3.3.4.2 Profile Prioritization

The next phase of profile analysis is to prioritize each profile based on the following two features: dangerous permissions, and user connectivity time. The objective is that, by means of the profile priority rank, an administrator can choose a high risk profile as the selected profile for further correlation and analysis. If the profile has dangerous permissions and connectivity time, it will be considered a high risk profile; otherwise, it will be considered low risk. The profile outputs will be stored in the abnormal profile table.

From the perspective of Android malware, malware can request more permissions than actually legitimate applications. Additionally, it can often request permissions that have risks related to user privacy and device security, such as collecting user data, collecting device information, accessing Internet, or sending and deleting SMS. We have analyzed seven well-known SMS botnet families (MisoSMS - Zitmo - NickySpy - TigerBot - Sandroid - PletorWroba) to study the distribution of requested permissions for each family. We found 20 dangerous permissions that abuse the SMS message service by means of SMS mobile botnets, all of which were missed by the Android security. The list of these dangerous permissions is described in Table 3.8.

Android attackers enable malicious applications to send out SMS messages to premium-rate phone numbers without the user’s knowledge, and to send out text messages while the phone is in sleep mode. The proposed framework is able to detect these malware activities by taking user connectivity time into account, along with dangerous permissions, when prioritizing profiles.

3.3.4.3 Abnormal Profiles Table (APT)

Figure 3.8 contains an illustration of an APT. APTs maintain records of all reported Android profiles. In the anomaly-based detection module, the SMS profiling and Android profiling agents decide about an SMS message and its profile on receipt. The APT divides
Table 3.8: List of dangerous permissions.

<table>
<thead>
<tr>
<th>No.</th>
<th>Dangerous Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTERNET</td>
</tr>
<tr>
<td>2</td>
<td>SEND_SMS</td>
</tr>
<tr>
<td>3</td>
<td>RECEIVE_SMS</td>
</tr>
<tr>
<td>4</td>
<td>READ_SMS</td>
</tr>
<tr>
<td>5</td>
<td>WRITE_SMS</td>
</tr>
<tr>
<td>6</td>
<td>RECEIVE_BOOT_COMPLETED</td>
</tr>
<tr>
<td>7</td>
<td>READ_PHONE_STATE</td>
</tr>
<tr>
<td>8</td>
<td>READ_CONTACTS</td>
</tr>
<tr>
<td>9</td>
<td>WAKE_LOCK</td>
</tr>
<tr>
<td>10</td>
<td>ACCESS_NETWORK_STATE</td>
</tr>
<tr>
<td>11</td>
<td>WRITE_EXTERNAL_STORAGE</td>
</tr>
<tr>
<td>12</td>
<td>ACCESS_WIFI_STATE</td>
</tr>
<tr>
<td>13</td>
<td>CHANGE_WIFI_STATE</td>
</tr>
<tr>
<td>14</td>
<td>MOUNT_UNMOUNT_FILESYSTEMS</td>
</tr>
<tr>
<td>15</td>
<td>ADD_SYSTEM_SERVICE</td>
</tr>
<tr>
<td>16</td>
<td>CALL_PHONE</td>
</tr>
<tr>
<td>17</td>
<td>READ_CALL_LOG</td>
</tr>
<tr>
<td>18</td>
<td>WRITE_CALL_LOG</td>
</tr>
<tr>
<td>19</td>
<td>BROADCAST_STICKY</td>
</tr>
<tr>
<td>20</td>
<td>RECEIVE_MMS</td>
</tr>
</tbody>
</table>

profiles into two categories:

1. Normal profiles, which consist of Android profiles with no indication of suspicious activity;

2. Abnormal profiles, which are recorded in the APT of the profile analysis, and require a suitable response.

The framework uses APTs to analyze new profiles to detect new attacks of SMS botnets. We assume that if there were regularities in the way an attacker usually developed and used the same apps to carry malicious activity, these could also be discovered from APTs. This automatic analysis of APTs is similar to the traffic log analysis conducted by human-network managers when they inspect the common features of harmful traffic. The exposed regularities are used to filter profiles that could be profitably used in the SMS correlation stage.
### 3.3.4.4 Android Profiling Analysis Algorithm

Algorithm 4 describes the procedure of the Android profiling analysis engine. The algorithm combines those profiles that have the same attributes, except “Android_ID”, “sending_time”, and “received_time”, and then flags these as combined profiles. The “Android_ID” is the unique name of the reported device. The algorithm then applies profile prioritization to prioritize the profile outputs based on risk. The suspicious profiles will be stored in the APT that can then be used by the SMS correlation component and the Security Administrator.

#### Algorithm 4 : Android profiling algorithm

**Inputs**
- \( ap \leftarrow \{ap_0, ..., ap_n\} \) (android user profiles)
- \( a \leftarrow \{a_0, ..., a_{13}\} \) (profile attributes)
- \( nump \leftarrow \emptyset \) (\# of combined profiles)
- \( hpp \leftarrow \emptyset \) (high level profile priority)
- \( lpp \leftarrow \emptyset \) (low level profile priority)

**Outputs**
- profile outputs \( PO \)
- \( po \leftarrow \emptyset \) (profile outputs)

**Method**

\[
\begin{align*}
\text{for each } & ap_i \in ap \text{ do} & \quad \triangleright \text{ refer to the table 3.6 and table 3.7} \\
& \text{Step 1 : combine } ap_i \text{ with same } a_{10}^i \text{ except } \text{Android_id, sending_time, and received_time} & \\
& \text{Step 2 : } nump \leftarrow \text{ the number of combined profile} & \\
& \text{if } a_{13}^i = \text{ dangerous permissions or } a_{13}^i = \text{ connectivity time} \quad \text{then} & \\
& \quad \text{Step 3.1 : } tpp \leftarrow ap_i & \\
& \quad \text{else} & \\
& \quad \text{Step 3.2 : } lpp \leftarrow ap_i & \\
& \text{Step 4 : } po \leftarrow tpp \text{ and } lpp \text{ in APT} & 
\end{align*}
\]

### 3.3.5 SMS Correlation

SMS messages have additional attributes that are noted in the anomaly-based detection module. These attributes are used to create profiles. We attach the features to find any correlation between outputs from profiles and the text messages. To reconstruct attack scenarios based on the profiles and the reported SMS in each class label, we use SMS correlation to identify the relationship between the outputs of the profiles and each
detected SMS message. The idea behind SMS correlation is to provide insight into attacks by analyzing raw profile outputs and SMS messages. SMS correlation is too difficult to be attempted in a single stage. Research work conducted on these correlations can be divided into four main categories: statistical correlation, scenario-based correlation, rule-based correlation, and temporal correlation [109]. We apply rule-based correlation in the SMS botnet detection framework. One advantage of using rule-based detection is that the set of rules helps to produce fewer false positive alarms [66]; it also has the ability to label unknown attacks. Many researchers have also studied the use of requested permissions to check the risk of an application, but it is hard to guarantee high detection rates in the permission-based detection approaches.

3.3.5.1 Rule-Based Correlation

The wide range of exploits used to satisfy attack goals has led to huge variations of an attack scenario, so it is difficult to maintain a comprehensive attack database. Rule-based approaches address this issue by generating finer-tuned correlations [88]. The rule-based methods associate alerts by coordinating prerequisites and consequences of attack phases. For instance, consider an attack_correlation (Attack1, Attack2): This predicate says that Attack1 may be correlated with Attack2; that is, Attack1 enables the intruder to then perform Attack2 [38]. In an Android platform, any application that sends or uses SMS service features must have permissions to access the service. An attacker has to define the permissions it needs before starting an attack.

In rule-based correlation, the important fields that need to be considered are as follows: attack prerequisites, or logical conditions that ensure the success of the attack; attack consequences, or logical conditions that identify the influence of the attack when this attack succeeds; and scenarios that describe the combination of events which are necessary to detect an occurrence of the attack [38]. In this approach, alerts are basically a set of logical facts about how Android platforms and SMS botnets work. In order to correlate two
alerts directly using rule-based correlation, one predicate in the consequences condition of the first alert should be connected with one predicate in the prerequisites condition of the second alert. We have studied the characteristics of Android SMS botnets and extracted some features that can aid in the detection of SMS botnets. After these alerts are discovered, correlation rules are applied, which explain the conditions under which an alert may occur, and preparation is made for the second alert, in order to correlate them directly if applicable.

Table 3.9: Correlation rules.

<table>
<thead>
<tr>
<th>Extracted features</th>
<th>F1: Blacklist</th>
<th>F2: In contact list</th>
<th>F3: Dangerous permissions</th>
<th>F4: SMS sent in sleep mode</th>
<th>F5: Reported by other agents</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender_num</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>5%↑</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Check URL</td>
</tr>
<tr>
<td>Has_URL</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>5%↑</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Check URL</td>
</tr>
<tr>
<td>Has_num</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>5%↓</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Check num</td>
</tr>
<tr>
<td>Has_command</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>2%↑</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Check command</td>
</tr>
<tr>
<td>Content</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>40%↑</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>10%↑</td>
</tr>
<tr>
<td>Outgoing SMS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>2%↑</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Check command</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>2%↓</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>2%↓</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>2%↑</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>2%↓</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Malicious</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>0%</td>
</tr>
</tbody>
</table>

80
We have defined a set of correlation rules that can be applied against each SMS message in each class label, along with its reported profile to label SMS messages as either malicious or normal. Table 3.9 shows these correlation rules. Any match of the rule is reported, and the SMS messages labelled as malicious. The set of rules may define an attack scenario involving some unusual SMS content. In this case, the algorithm first checks whether or not the sender phone number of the SMS message is in the user contact list, and also applies permission based methods to check for dangerous permissions. Additionally, we check the percentage of reported text messages that have the same features. These rules are pre-defined, and they label the SMS messages as either “malicious” or “features need to be checked”. We consider all SMS messages that pass the rule-based methods to be normal SMS messages.

For each SMS, the feature extraction technique is used to extract six features from SMS messages. These features are:

1. “Sender_num”, which refers to the sender phone number of the SMS author.
2. “Has URL”, which refers to whether an SMS message contains an embedded URL.
3. “Has_num”, which refers to whether an SMS message contains an embedded number in it.
4. “Has_command”, which refers to whether an SMS message contains an embedded command.
5. “Content”, which represents SMS content.
6. “Outgoing_SMS”, which refers to the type of outgoing SMS message.

### 3.3.5.2 SMS Correlation Algorithm

Algorithm 5 outlines a method for the SMS correlation. At first we apply feature extraction to each SMS message. The results of the feature extraction are called alerts. In the second
step, we will correlate each SMS messages to its profile outputs. If the message has an alert, we apply the correlation rules explained in Table 3.9. If any match of the rules is found, the SMS message will either be labelled as malicious or will require further analysis by an administrator. However, if there is no match, the algorithm will apply the next correlation rules. Take for example the scenario of an SMS that has “has_URLS” as one of the extracted features. That SMS will be compared with corresponding profile outputs that have the features, “Blacklist” (F1), “In contact list” (F2), “Dangerous permissions” (F3), “SMS sent in sleep mode” (F4), and “The percentage of same SMS reported by Android devices” (F5), and with the profile that have at least one of these feature, as shown in Table 3.9.

**Algorithm 5 : SMS correlation algorithm**

**Inputs**

\[
\begin{align*}
cln & \leftarrow \{smscl_1, ..., smscl_n\} \text{ (all normal sms messages)} \\
clmn & \leftarrow \{smscl_1, ..., smscl_n\} \text{ (majority normal sms messages)} \\
clm & \leftarrow \{smscl_1, ..., smscl_n\} \text{ (all malicious sms messages)} \\
clmm & \leftarrow \{smscl_1, ..., smscl_n\} \text{ (majority malicious sms messages)} \\
cl & \leftarrow \{cln, clmn, clm, clmm\} \\
op & \leftarrow \{po_0, ..., po_k\} \quad \triangleright \text{ profiles outputs see android profiling algorithm} \\
r & \leftarrow r_{i=0}^n \text{ correlation rules}
\end{align*}
\]

**Outputs**

\[
\begin{align*}
\text{malicious} & \leftarrow \emptyset, \text{ normal} \leftarrow \emptyset
\end{align*}
\]

**Method**

\[
\begin{align*}
\text{for each} \; smscl \in cl & \; \text{ do} \\
\quad smsp & \leftarrow \text{correspond profile with sms}(smscl, po) \\
\text{for each} \; smsp_i \in smsp & \; \text{ do} \\
\quad fe_{i=0}^5 & \leftarrow \text{feature extraction}(smscl) \\
\quad \text{if} \; fe_i = r_i & \; \text{ then} \\
\qquad \text{malicious} & \leftarrow smsp \quad \triangleright \text{ apply correlation rules} \\
\text{else} \\
\qquad \text{normal} & \leftarrow smsp \\
\end{align*}
\]

Using this approach, the module will be able to detect known and unknown malicious SMS messages, to label SMS messages as malicious or normal, and to help identify malicious apps that misuse SMS service in the Android device. It is up to the Android user to remove the malicious app, based on the result that our system provides back to the user.
along with an explanation of the attack and of the risk involved in using the malicious app. In order to take down the unknown botnet, it will be the administrator’s task to obtain the malicious APK file from the user or another market, and to perform further analysis to understand the functionally of the app. The administrator will carry out static analysis and then extract the features and flag the app as malware.

3.4 SMS Defence Module

Typically, an SMS defence module begins by gaining insight into unknown SMS botnets and then generates signatures and rules. The defence module described in this thesis attempts to protect Android smartphones by introducing a proactive approach to generate signatures and rules. The defence module consists of four components, namely, signature generation, phone number blacklist (PNBL), malicious application analysis, and response action. The signatures identify known SMS botnets and the rules that are used to spot unknown SMS botnets. The defence module uses the output received from an anomaly-based detection module to make logical decisions based on a set of policies that have been established by a human-network manager.

There are four components in this module, as shown in Figure 3.9: signature generation, phone number blacklist, response action, and malware analysis. The primary component, signature generation, includes the signatures generated when new SMS botnet activities are found. The response action component involves the actions taken by an Android user when all the necessary criteria are satisfied.

Each component describes one of four approaches that can be used to respond to an attack. The first approach is to identify the malicious correspondent’s phone number and block it. The second approach identifies the misbehaviour of apps: it detects common features of the malicious applications and prevents those apps from running in the smartphones. The third approach is to identify the similar characteristics of malicious
SMS messages and to group them by their common features. These common features include FromPhone#, ToPhone#, URLs, commands, phone# in SMS, size of SMS, time, and the app names, for example. After getting the result from the anomaly-based detection module, the administrator can interact with an Android user using the fourth approach, that is, by sending feedback to the user. The feedback will explain what the user should know and what actions the user should take.

![Figure 3.9: Overview of SMS botnet defence module.](image)

### 3.4.1 Signature Generation

The idea behind signature generation is to generate signatures that are representative of attack patterns. To ensure acceptable rates of false positives and false negatives during the signature detection process, we consider many exploits, and frequently update the signatures. The central agent sends the signature updates to all Android mobile devices. The first line of defence against SMS botnet activities is the signature-based detection module in Section 3.2, which scans incoming and outgoing SMS messages. Obviously, known SMS botnet attacks would be easily stopped by blocking SMS messages that match the corresponding signatures. Unknown SMS botnet attacks, however, if they match the
defined rules, will be reported to the anomaly-based detection module as suspicious messages in order that we may perform further investigation and label them either malicious or normal. For all messages labelled malicious, signatures will be created based on selected features that are described in Table 3.4.

The anomaly-based detection module will label both the SMS message and its profile. Each message reported malicious will have an explanation noting why it is malicious. We will take the SMS messages and generate signatures based on the defined features. Each SMS messages will have the following attributes: FromPhone#, ToPhone#, URLs, Command, Phones#, Content, and Phishing Words. The Administrator will then be able to add more rules and send them to Android smartphones as well.

3.4.1.1 Signature Generation Algorithm

Signatures are generated using the signature generation algorithm (see Algorithm 6). The defence module receives the malicious SMS messages reported by the anomaly-based detection module. At first we compare the new SMS messages with the existing malicious SMS messages. If an SMS message already has a signature, the algorithm will attempt to match the message’s other features. If there is any match, the SMS message will be ignored. If there is no match, the algorithm will generate a signature of the following features: FromPhone#, ToPhone#, URLs, Command, Phones#, Content, and Phishing Words. It will then repeat the same process until it has generated a signature for each malicious SMS message. The signature updates will be sent to all subscribed Android mobile devices.

3.4.2 Phone Number Blacklist (PNBL)

Blocking malicious SMS is the primary defence against SMS botnet attacks. Clearly, SMS-based attacks would be defendable by filtering if there were regularities in one or more of the attributes of the malicious SMS on Android smartphones. A phone number blacklist
Algorithm 6: Signature generation algorithm

Inputs
\( ms \leftarrow \{ms_1, ..., ms_n\} \) (malicious SMSes)
\( fs \leftarrow f_{s0}, f_{s1}, f_{s2}, f_{s3}, f_{s4}, f_{s5} \) (features signature)

Outputs
\( SMSs \leftarrow SMSs_1, ..., SMSs_n \) (SMS signatures)

Method
\[
\text{for each } ms_i \in ms \text{ do} \\
\quad \text{if } f_{s5_i} = 0 \neq smss \text{ then} \\
\quad \quad SMSs \leftarrow \text{signature generation}(ms) \\
\quad \text{else} \\
\quad \quad \text{remove } ms \\
\quad \quad \text{send a copy of } SMSs \text{ to android smartphones}
\]

(PNBL) contains a list of phone numbers that the SMS botnet detection app should block and should not accept any SMS text messages from. A PNBL can be queried with the signature-based detection module and allows an efficient way to perform lookups. As an example, when detection results report that a set of malicious SMS messages having the same phone number (a common feature of malicious traffic) is initiating harm (sending SMS spam, commands, etc.), we would generate a signature of the phone number and then send it to the signature-based detection module in Android smartphones, so that the module could perform a signature scan and block SMS text messages from this phone number.

3.4.3 Malicious Applications Analysis

Malicious apps are the primary means by which SMS botnets, receiving commands through the SMS service, perform attacks. Analyzing reported apps and extracting their features is therefore a strong method of defence against SMS botnets. The profiling analysis step is done in the anomaly-based detection module, and the outputs are shown to a security administrator, who can perform static and dynamic analysis using common tools such as Androguard [10], and DroidBox [15].

The profile outputs represent the degree of risk presented by an installed application (low,
medium, or high), as gauged by a specific set of security rules. For example, the use of permissions is not as dangerous in some apps as it is in others. The profile outputs can include a normal feature of an attacked smartphone and can be part of a totally legitimate profile. However, a malicious app can also exploit this feature. Research experiments show that, it may not be always possible to confirm the intent of using permission to recognize an attack. Nevertheless, security administrators are be able to use this technique to understand the functionality of the malicious app, and to confirm features and characteristics of malware.

3.4.4 Response

The results of the anomaly-based detection module determine the degree of threat or severity of an attack against the Android smartphone. Although identifying malicious SMS messages will help to block SMS botnets by taking down SMS bots and cutting the C&C channel, it also requires the Android user to cooperate by removing the malicious application.

The security administrator is able to send a request to users, asking them to perform an action, for the protection of their smartphones. We have developed an SMS botnet detection app that runs agents, performs signature detection, and provides an interface to allow the administrator to communicate and interact with the Android user. In the Android platform, users themselves have to uninstall the apps based on the information provided. The administrator provides an extensive explanation about the malicious app, including information about its publisher, and other apps from the same publisher. Also, the administrator indicates what dangerous permissions the app used, and notifies the user that the app is sending out SMS messages without the user’s knowledge.
3.5 Concluding Remarks

In this chapter we have introduced an SMS-based botnet detection framework that improves the traditional detection approaches by utilizing JADE agents. The proposed framework allows integration of different modules in the form of concise yet meaningful behavioural profiles, correlation techniques, and defence strategies. In general, the proposed framework has the following advantages:

- Supports timely monitoring and protecting of subscribed smartphones using JADE agents.

- Provides a real-time content-based signature detection of SMS messages as a first-line defence in smartphones.

- Provides a comprehensive and high level overview of SMS clustering, SMS classification, Android user behaviour analysis and SMS correlation in the SMS botnet detection. These comprehensive reports can further be visualized to make pairwise comparison of SMS message and its profile easier in order to label it as malicious or normal.

- Improves the existing content-based detection or classification approaches by utilizing alert correlation techniques to find any correlation between a suspicious SMS message and its profile.

- Provides a defence module that offers response Actions by signature generation, creates phone number blacklist, and performs malicious applications analysis. Allows future extension by adding other analysis tools as plug-ins and fully automating the analysis process.

- Makes it possible to extend rules when more information about malware becomes available.
Chapter 4

Multi-Agent System Implementation

Although having the theoretical basis for an intelligent mechanism is important, more important still are the contributions that have been made, by researchers in the artificial intelligence (AI) community, to evolve agent technology from theory to practice. Several multi-agent platforms have been developed for smartphones. The use of a multi-agent approach in smartphone security with a focus on the Android platform is a new area of research. Using an intelligent mechanism will enable discovery of SMS botnets and malicious applications by building user Android profiles and correlating the results with suspicious SMS.

Multi-agent systems for botnet detection require a lot of agents with differing structures and functionalities. Past research has proven that a multi-agent system can be successfully applied for intrusion detection [114]. therefore, this is the type of agent system we use. The main advantages of using a multi-agent system are pro-activity, autonomy, and self-awareness. These advantages have been extensively discussed by Carabelea et al. [30].

4.1 Agent Architecture

Agents have the ability to autonomously perform tasks and to decide how an action is to be taken. In this section, we discuss the development of autonomous objects that can interact
with each other to solve application issues via cooperation or negotiation. Another aspect
of agents, besides autonomy, is their productivity which means that agents can have their
own goals, and can behave freely in order to achieve these goals.
Several agent architectures exist, each of which proposes different kinds of mental atti-
tude. One prominent architecture is the belief-desire-intention (BDI) model. This model
is based on philosophical human reasoning, including the following concepts: beliefs, de-
sires and intentions. The theory of the practical reasoning was developed by Bratman at
el. [24]. Bratman develops an intention planning theory, in which intentions are considered
elements of partial plans of action. In practical reasoning, these plans play simple roles,
which accomplish the organization of activities over time and socially.
The belief-desire-intention (BDI) model was an improvement of Rao et al.’s architec-
ture [104], which recognizes many of the vital elements of Bratman’s theory of intention.
Their approach considers the agent’s beliefs, goals and plans, which represent slightly
changed theories of the original notions of intention. In their newer model, the intention
has equivalent status to the notions of belief and desire. In other word, in their BDI archi-
tecture, intentions are an important element of an agent, which determine the behaviour
of rational agents as those agents pursues their goals.
The BDI model is desirable for its ability to develop an automated response mechanism
that selects appropriate response plans according to the current context and status of an
ongoing attack. In case of a failed response, it is able to implement a substitute plan, to
accommodate the changing behaviour of attacks [117]. We chose the BDI agent model for
our framework design, because a BDI agent [104] is able to always reason about beliefs,
goals and intentions, and to take action consequently. The fundamental theoretical model
of BDI is defined briefly here, but an extensive and detailed description can be found in
Rao et al. [104].
There are four major concepts important to BDI architecture [150]:
1. Beliefs of an agent are information about the environment. Beliefs can contain
suggestion rules to lead to new beliefs and they are subject to uncertainty and error.

2. Desires are goals to be accomplished, which are allocated to the agent. The rationality of agents requires that their desires be achievable and mutually consistent. Goals represent objectives or situations that the agent would like to achieve or bring about.

3. Intentions are an agent’s commitments to achieve particular goals. Such commitments imply that agents can choose what to do in order to reach the end state defined by their goals.

4. Plans are sequences of actions that the agent can carry out to attain one or more of its intentions.

According to Rao et al. [103], there are three basic requirements for a rational agent.

- First, a rational agent must have the ability to engage in practical reasoning about its beliefs, goals, and intentions at different time points and have the ability to act accordingly.

- Secondly, in case an agent should fail to execute the event, a rational agent must be able to differentiate between a failure in the execution of the event (or action) and an event’s non-occurrence. This distinction will permit the agent to take suitable future actions.

- Finally, the plans (intentions) that an agent formulates must be both partial and hierarchical [24]. In other words, agents formulate plans at an abstract level first, and then later on, as more information become available, redefine their plans to achieve their goal. Moreover, a plan to accomplish an end-goal must contain plans to complete its sub-goals, and achieving the sub-goals is essential to the success of the end-goal.
4.2 Java Agent Development Framework (JADE)

JADE is a software development platform which complies with FIPA specifications. The JADE platform is used to develop multi-agent applications by providing basic middleware-layer functionalities. JADE is an Open Source project, and the complete system can be downloaded from the JADE Home Page [131]. The JADE agent architecture is shown in Figure 4.1. The goal of JADE is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents. JADE is written in a well-known, object-oriented language, Java, providing libraries that fully support the Android platform [27]. The advantage of the JADE platform is its provision of ready-to-use and easy-to-customize core functionalities. The JADE Platform can be adapted to the characteristics of a deployment environment [15].

To achieve such a goal, JADE offers the following list of features to the agent programmer:

- FIPA-compliant Agent Platform, which includes the AMS (Agent Management System), the default DF (Directory Facilitator), and the ACC (Agent Communication Channel). All these three agents are automatically activated at the agent platform.
- Distributed agent platform. The agent platform can be split between several hosts. Only one Java application, and therefore only one Java Virtual Machine, is executed on each host. Agents are implemented as one Java thread and Java events are used for effective and lightweight communication between agents on the same host. Parallel tasks can be still executed by one agent, and JADE schedules these tasks in a cooperative way.

- A number of FIPA-compliant additional DFs can be started at run time in order to build multi-domain environments, where a domain is a logical set of agents, whose services are advertised through a common facilitator.

- Java API to send/receive messages to/from other agents; ACL messages are represented as ordinary Java objects.

- Provision of a FIPA-ACL message transportation mechanism to exchange information between agents. When agents are in remote machines, messages are transformed from JADE’s own syntax into proper FIPA-compliant transport representation and protocols. This mechanism includes implementation of white pages and yellow pages.

- A set of graphic tools to support developers when debugging and testing.

- Support agent mobility, allowing agents to migrate between machines.

- A fully supported library of interaction protocols compliant with FIPA specifications.

- Ontology checking and content encoding. This is supported by the JADE platform to provide languages and ontologies.

- An extended JADE platform to Android platform and updated Android run-time library.
• Graphical user interface to manage several agents and agent platforms from the same agent. The activity of each platform can be monitored and logged. All life cycle operations related to agents (creating a new agent, suspending or terminating an existing agent, etc.) can be performed through this administrative GUI [17].

The logical way to define a platform-independent agent model is to chose appropriate concepts that can be used to build the different features and classes of agents. The proposed procedure allows one to employ the analysis and design of the framework according to several well-known methodologies [87]. The agent meta-model [5] presents five main class-level constructs:

1. An agent refers to an intelligent software agent’s behaviour, including the knowledge base kept in its Belief set and Goal set. To communicate with the outside, an agent uses two message queues, namely, Input and Output that temporarily store incoming and outgoing messages, respectively. An agent can become aware of an event arrival, which is stored in EventQueue. Each agent has a “State”, that relates its life-cycle and its visibility to other agents. The behaviours of an agent include the capabilities of the agent and the types of messages and events to which it responds, as well as the plans it uses to achieve its goals.

2. A set of behaviours is defined in the agent to differentiate between a variety of environments and attention motivations. Behaviours are the instructions that the agent follows to achieve its goals and to handle its designated events. Fundamentally, behaviours are used to cut and delimit the information the agent uses to solve a problem. In order to increase agent efficiency in regard to the problem-solving process, the events and messages that are not associated with the existing agent phase must not be taken into consideration. Each behavior has a “Name” to identify itself and a set of goals that are also related to the “Behaviour”, which could include either activation or conservation situations. Each behaviour also has a “State” rep-
resenting its existing activation condition. One or more behaviours may be active simultaneously.

3. The tasks, plans, events, and the beliefs that an agent is able to fulfill are modeled as “Capabilities”. The “Behaviours” include the “Capabilities”, which in turn describe the agent’s response to certain abilities. The capabilities make it possible for the practical elements that make up an agent to be gathered and reused. An agent includes a number of capabilities, each which has a specific purpose. A capability is labelled by means of a name that recognizes it, its initial event, an initial condition, and the task that has to be performed when the event arrives and the designated condition is satisfied. The state of the capability is also indicated. Only capabilities appropriate to current active behaviours are accomplished. New capabilities will be activated accordingly. An active state where all of the capabilities of a current active behaviour will be noted.

4. Tasks are the component which contains the code related to the agent’s capabilities. In the execution state, there is one task for each capability, which will stay in this state up to its finishing point or until the capability is disturbed as the behaviour it relates to is deactivated. In case of interrupted tasks, there is no rescue or resumption method that has been defined yet.

5. An event refers to an occurrence which the agent must accomplish. The agent uses the event for any notification received that certain rules meet to accomplish its goal in the environment or inside the agent.

6. The intent specifies the goals that trigger the behaviors and events which permit the tasks of capabilities to be executed. To manipulate and store agents’ beliefs, the content provider is used as a database.

The most popular specification for designing JADE agents is the Unified Modeling Language (UML) [13, 87]. The class level categories for JADE constructs (agents, behaviours,
capabilities, task, databases, and events) are defined as follows [5]:

- **<< agent >>** is a class level type that defines a JADE agent;
- **<< behaviour >>** is a class level type that defines a JADE behaviour;
- **<< capability >>** is a package/subsystem level type that defines a JADE capability;
- **<< task >>** is a class level type that defines a JADE task;
- **<< event >>** is a class level type that defines a JADE event; and
- **<< database >>** is a class level type that defines a JADE database.

When defining the behaviours of an agent, the relation between the agent entities and the agent’s communication with other agents is vital. The following set of labels has been used to define relationships between agents, behaviours, capabilities, databases and events [94, 5]:

- **<< chooses >>**, an agent choosing a capability;
- **<< uses >>**, an agent using a task;
- **<< posts >>**, an agent posting an internal event;
- **<< handles >>**, an event handled by a plan;
- **<< privatedatabase >>**, a private database owned by an agent;
- **<< usesagent >>**, a task using an agent implementing an interface;
- **<< sends >>**, an agent sending an event to another agent; and
- **<< modifies >>**, a task modifying a database.
Figure 4.2: The architecture of the Android profiling framework using the JADE platform.
4.3 Multi-Agent System Development

Our framework aims to secure the Android platform by developing a software layer that employs JADE middleware to create an Android user profile. In this section, we present a multi-agent system that has been developed using the JADE platform, as shown in Figure 4.2. The main idea is that agents have been developed to monitor Android platform devices and spot any unusual activities and abuse to the SMS service that may be caused by malicious applications. This abnormal behaviour will be reported to the server where more behaviour analysis will be conducted.

Developing agent-based applications with JADE requires a runtime environment, a library of classes, and a suite of graphical tools. The JADE platform is a set of containers with a main container. Each agent has its own unique name that allows other agents to communicate with each other regardless of their location [28]. Two important agents and the Agent Communication Channel must operate in the main container:

1. The Agent Management System (AMS), which has two special tasks: providing naming services and representing the authority in the platform.

2. The Directory Facilitator (DF), which is responsible for supplying a Yellow Page service that defines how the agent can interact with another agent providing the service required to achieve the goal.

3. All the exchange messages between agents in the platform are controlled by an Agent Communication Channel (ACC). The AMS and DF functions are engaged when the JADE platform is launched, and a messaging service is activated on the ACC to provide a channel for communication.

The AMS and DF live in the main container, to which other containers can connect, and provides a runtime environment for the execution of the agents’ work. Stand-alone execution mode and split execution mode can be used to execute a JADE runtime environment. In the stand-alone execution mode, the container is operated on the targeted
device, whereas, the split execution mode is distributed to a front-end client that runs on a mobile device and a back-end that runs on a server. The front-end and back-end are linked together via a robust connection to avoid any connection failure [143]. The SMS botnet detection framework uses the split execution mode because it is more suited to run on mobile devices, and JADE runtime service (MicroRuntimeService) include into the SMS botnet detection application. The main advantages of using split execution mode are: 1) it is extremely lightweight; 2) it minimizes the communication over the wireless link; 3) it is faster in the start-up procedure; and 4) it can handle temporary disconnections [143].
4.4 Android Profiling Approach

The Android profiling approach is an agent-based implementation of the SMS botnet detection framework. Each module of the SMS botnet detection framework is implemented in order to interact with agents using the JAVA/JADE [16] development environment. The architecture of the proposed multi-agent system is shown in Figure 4.3. In the service provider, there are three types of agents: the central, SMS profiling and Android profiling agents. For each Android smartphone, there are four different types of agent: the Android, signature detection, app-profile, and user-profile agents.

The Android agent establishes a connection with the central agent. It plays a critical role in the system, since it creates a channel that allows communication from the actual phone to the service provider. It also manages and supervises the interactions between local agents, and sends requests to the app-profile and user-profile agents when malicious or suspicious SMS is detected. The signature detection agent monitors incoming and outgoing SMS messages. This agent obtains updates of the signatures and rules from the SMS profiling agents. It forwards all incoming and outgoing SMS messages to the SMS Signature module, which performs the detection and sends the results back to the agent. If the SMS message is labelled as normal, the agent will deliver it to the SMS default app, and if the SMS message is malicious or suspicious, the agent will send a command to the Android agent, which can then request the current profile update to be sent by the app-profile and user-profile agents. All local agents will forward the current profiles to Android profiling agent, while simultaneously sending reported SMS messages to the SMS profiling agent. The app-profile agent is responsible for creating app profiles by observing installed applications that include the granted permissions, and apps that access the browser or try to communicate and broadcast with other installed applications. This agent responds to the requests from the Android agent, and forwards the current profiles to the Android profiling agent. The user-profile agent builds a profile by monitoring user connectivity time, maintains the phone number blacklist, and reports daily usage of the
mobile phone. This agent responds to the requests from the Android agent and forwards the current profiles to Android profiling agent.

An Android mobile device must subscribe to the central agent in order to obtain all the defined services from the SMS profiling and android profiling agents. The central agent registers the Android agents and informs the service provider agents about each new subscription request, so that they may start offering services to all agents in the Android smartphone. The central agent also informs the Android agent that these agents have been added, so they can begin to protect Android smartphones. The central agent sends commands to the Android agents, detailing the decision that has been obtained and established in the defence module. It manages all the local agents situated in the service provider. The central agent also performs activities that are relevant to Android mobile device agents, such as managing, updating, blocking, deleting and controlling. The SMS profiling agent handles reported SMS messages that are considered malicious or suspicious, and maintains SMS logs. These received SMS data and logs are forwarded to the anomaly-based detection module to verify whether they are deemed to be botnet activities. This agent also obtains the generated signatures and rules that are reported by the defence module, which need to be forwarded to the signature detection agents. Once the profile updates are received by the android profiling agent in the service provider, this particular agent will then maintain and update the profile for all subscribed smartphones. Additionally, this agent updates the received changes from the anomaly-based detection module. It responds to anomaly-based detection module requests, which are findings and actions that need to be acted upon.

4.4.1 Service Provider Agents Design

Three agents operate on the server side, as shown in Figure 3.2. These agents have heavy roles to play, to maintain the list of subscriber agents, observe suspicious SMS messages and Android profiles to spot any abnormal behaviour, find any correlation between the
reported Android profiles, and perform actions. These three major agents in the service provider server, which perform the majority of the activities concerning the Android user profiling, are the Central agent, the Android profiling agent and the SMS profiling agent, as outlined below:

4.4.1.1 Central Agent

The central agent offers a suitable user interface for creating and initializing other agents. It permits an administrator to interact with the anomaly-based detection module and defence module, and to respond when an action is needed. The administrator’s interaction with the framework is limited, and only occurs when signatures need to be verified, or in order to request a suspicious application from the user, for analysis. Thus, our methodology lets a security administrator’s intervene when it is required. Each agent provides a graphical or text display of events and decisions for human visual analysis and interpretation.

In order to access all these services, an Android mobile user must register on the server. The Android mobile user can deregister from the central service provider at any time. This agent has the ability to maintain, update, block, delete, and control all subscribing Android users. This agent handles and accepts Android agent subscription requests and adds them to the “subscribe list”. The central agent keeps track of when each Android agent enters and exits the framework. In addition, it manages the local agents situated in the server. The central agent is in charge of sending commands to the Android agents to take action, based on the decision that has been obtained and established in the defence module. It also has to keep track of the number of reported suspicious SMS messages, and then send requests to the SMS profiling agent, where all suspicious SMS messages and logs are managed, to verify that they have been receiving the suspicious SMS.

The overall design and behaviours of the central agent are illustrated in Figure 4.4. Unlabelled arrows indicate that particular agent has the corresponding behaviours:
1. **Subscribe** is the behaviour to subscribe an Android mobile device so that it may receive all the services.

2. **Unsubscribe** is the behaviour to unsubscribe the Android mobile device upon a request from an Android user or a security administrator.

The behaviour chooses one of the following capabilities as defined for the central agent:

1. **RegisterNewAgent**, to register new agents upon a request from an Android user.

2. **DeregisterExistingAgent**, to deregister an existing agent upon a request from an Android user.

3. **NotifyAboutNewAgent**, to notify other agents about a new agent registered.

4. **NotifyAboutAgentRemoval**, to notify other agents about an existing agent deregistered and removed from the framework.

The central agent has the following tasks:

1. **RegistrationRequest**, which announces a new agent subscription request by sending a NewRegisterRequired event.
2. **DeregistrationRequest**, which announces a new request of agent unsubscription by sending a **DeregisterRequired** event.

3. **NotifyAboutNewAgent**, which notifies the user about a new agent subscription by sending a **NotifyAboutNewAgent** event.

4. **NotifyRemovedAgent**, which task notifies the user about the unsubscription of an existing agent by sending a **NotifyRemovedAgent** event.

Finally, the central agents controls the following events:

1. **SubscriptionRequired**, posted by the central agent when a new agent needs subscription.

2. **NewRegisterRequired**, sent to the new agent that needs registration. This event includes the parameters that allow the registered agent to obtain all the services.

3. **UnsubscriptionRequire**, posted by the central agent when an existing agent needs to be unsubscribed.

4. **DeregisterRequired**, sent to the existing agent that needs deregistration. This event includes the parameters that allow the deregistered agent to unsubscribe from all the services.

5. **NotifyAboutNewAgent**, sent to the new agent that needs to inform other agents in the server about its registration.

6. **NotifyRemovedAgent**, sent to the existing agent that needs to inform other agents in the server about its deregistration.

### 4.4.1.2 SMS Profiling Agent

The SMS profiling agent obtains copies of suspicious and malicious SMS messages and SMS logs. This agent maintains a database for all reported suspicious SMS, and handles
the SMS logs. It forwards all the suspicious SMS messages and log data to the anomaly-based detection module, for it to verify whether they are deemed botnets. In addition, this agent receives the generated SMS signatures from the defence module, maintains a local copy, and then sends a signature update to the signature detection agent on each Android mobile device.

The SMS profiling agent stores the signature database generated in the defence module. These signatures are sent to the subscribed smartphones, which require SMS botnet signatures to be updated frequently. The SMS profiling agent maintains the reported SMS messages and interacts with signature detection agents to accept the suspicious and mali-
cious SMS messages. The anomaly-based detection module reads reported SMS messages that have been collected by the SMS profiling agent to investigate them. The SMS profiling agent receives the characteristics of SMS messages that need to be classified, and applies the appropriate correlation rules. As a result, as long as the classifier is active, the reported SMS messages with new threats will not be detected as malicious but profile analysis will help to spot malicious messages using rules-based correlation techniques.

To describe different databases of the SMS botnet detection framework such as the generated signatures database, some stereotypes at the feature and method levels are required [37, 95], in addition to the class/package and association level stereotypes that we described in Subsection 3.2.1. Attribute level (<<keyfield>>) and <<valuefield>>) and method level (<<indexedquery>>) stereotypes are defined in Figure 4.5 to achieve this distinction.

Figure 4.6: Overall design of the SMS profiling agent.
The overall design of the SMS profiling agent is described in Figure 4.6, including the following three behaviours.

1. *NewSMSReading*, which reads SMS messages from the data set already reported from the Android devices.

2. *NewSignaturesUpdating*, which reads the signature updates from the dataset and sends them to Android devices.

3. *DAMConfigurationChange*, which enables the agent to send a command to take an action in the user device.

Each behaviour chooses one of the four capabilities defined for SMS profiling agent, which are as follows:

1. *NewSMSSuspicious*, capability used by the SMS profiling agent to send the reported suspicious SMS messages to the anomaly-based detection module.

2. *NewSMSMalicious*, the capability used by the SMS profiling agent to send the reported malicious SMS messages to the anomaly-based detection module.

3. *NewSignaturesUpdating*, the capability that enables the SMS profiling agent to update the signatures and sends the update to the signature detection agent in smartphones.

4. *DAMConfigurationChanging*, the capability that enables the SMS profiling agent to change the configuration parameters by using new values provided by the administrator via the central agent.

The SMS profiling agent has also the following four tasks:

1. *SendNewSuspiciousSMS*, used upon receiving suspicious SMS to send it to the anomaly-based detection module by sending a *NewSuspiciousSMS* event.
2. *SendNewMaliciousSMS*, used upon receiving malicious SMS to send it to the anomaly-based detection module by sending an *NewMaliciousSMS* event.

3. *NewSignaturesUpdating*, which applies updates requested to the signatures in the signature detection agent, by sending a *SignaturesRequired* event.

4. *SendDAMCommand*, which applies requested changes to the configuration of an Android agent, by sending a *DAMRequired* event.

Finally, the SMS profiling agent controls eight events:

1. *NewSuspiciousSMS*, which provides the anomaly-based detection module with the latest value of a variable of a suspicious SMS message that is relevant in detecting an attack.

2. *SuspiciousSMSAvailable*, which retains all the reported suspicious SMS messages that require further analysis.

3. *NewMaliciousSMS*, which provides the anomaly-based detection module with the latest value of a variable of a malicious SMS message that is relevant in detecting an attack.

4. *MaliciousSMSAvailable*, which retains all the reported malicious SMS messages that require further analysis.

5. *SignaturesRequired*, which obtains the signatures updates from the defence module that are important in detecting an attack.

6. *NewSignatureAvailable*, which keeps the signature updates and receives the updates from the defence module.

7. *DAMRequired*, which sends commands to Android agents in smartphones that are provided by the defence module.

8. *DAMRequest*, which obtains a command to carry out an action in the smartphones.
### Figure 4.7: Android profiles database.

#### 4.4.1.3 Android Profiling Agent

The main task of the Android profiling agent is to keep track of reported profiles from the app-profile and user-profile agents. The Android profiling agent maintains and updates the profiles for all registered Android mobile devices. These profiles are forwarded to the anomaly-based detection module for further analysis, along with any correlations noted between profiles and reported suspicious SMS. Finally, this agent can request more information from the app-profile and user-profile agents if required.
Android profiles are maintained in “AndroidProfiles” databases. These profiles have different attributes that represent their behaviour. The anomaly-based detection module analyzes the profiles collected by Android profiling agents, as described in Subsection 3.3. The Android profiling agent also accepts the characteristics of reported profiles that should be analyzed and correlates output profiles with reported SMS by applying the appropriate rules to label them. Attribute level (<< keyfield >> and << valuefield >>) and method level (<< indexedquery >>) stereotypes are defined in Figure 4.5 to achieve this distinction.

The current implementation of the Android profiling agent is described in Figure 4.8. This agent has three behaviours:

1. NewAndroidProfileReading reads new reported profile that have the NewAndroidProfileReading capability.

2. ProfileUpdateRequest requests profile updates from the app-profile and user-profile agents in smartphones.
3. *RespondToDMRequest* forwards the profiles to the anomaly-based detection module upon requested.

The Android profiling agent has the following three capabilities:

1. *NewAndroidProfileReading* which reads Android profiles when selected by the *NewAndroidProfileReading* behaviour.

2. *ProfileUpdateRequest*, which updates Android profiles and forwards them to the anomaly-based detection module.

3. *RespondToDMRequest*, which responds to the anomaly-based detection module, and requests additional information from the app-profile and user-profile agents in smartphones.

The tasks of the Android profiling agent are as follows:

1. The *SendNewAndroidProfile* task forwards new Android profiles to the anomaly-based detection module by sending the *NewAndroidProfile* event.

2. The *SendProfileUpdateRequesting* task requests Android profile updates from other agents in smartphones by sending the *SendProfileUpdateRequesting* event.

3. The *ReplyToDMRequest* task enables the Android profiling agent to reply to the anomaly-based detection module request by sending the *DMRequested* event.

Finally, the Android profiling agent handles and post six events:

1. The *SendNewAndroidProfile* event forwards a reported Android profile to the anomaly-based detection module.

2. The *AndroidProfileAvailable* event notifies the agent when an Android profile is received.
3. The UpdateProfileRequest event requests a profile update from the app-profile and user-profile agents in smartphones, upon receiving a request from the anomaly-based detection module.

4. The ProfileUpdateAvailable event handles by the Android profiling agent when profile updates are available.

5. The DMRequested event responds to the anomaly-based detection module.

6. The DMRequestedAvailable event is posted by the Android profiling agent upon receiving anomaly-based detection module requests.

4.4.2 Android Smartphone Agents Design

Each agent has a set of roles designed to allow it to achieve its goals. In order to access all the services, an Android mobile user must registers with a service provider on the server. The agents are then used to detect SMS botnets, observe smartphone behaviour and resources, and from that information to create an Android user profile. As shown in Figure 4.9, there are four agents that reside in each mobile device, including an Android agent, a signature detection agent, an app-profile agent, and a user-profile agent. The design of each agent is outlined below.

4.4.2.1 Android Agent

This agent subscribes to the server and obtains the agent identification from the service provider to establish the communication between the agents on both sides. This agent also obtains all the available services and stays active for the entire life cycle of the agent platform on the mobile device. The Android agent plays a critical role in the system as it creates a channel that allows communication from the actual phone to the central server. This agent has to be aware of the mobile resources, and must read the status of the smartphone, including SMS delivery and battery power. As well as managing
the communications between the local agents on Android mobile devices, this agent is in charge of replying to any request received from the server. It receives an update to protect the Android device whenever a new threat is found. It communicates with the signature detection agent, the app-profile agent, and the user-profile agent to execute any commands.

The implementation of the Android agent is illustrated in Figure 4.10, in which the four unlabelled arrows indicate the following four agent behaviours:

1. The SubscribeToServer behaviour enables the Android agent to subscribe to the central server to obtain all the services, using the SubscribeToServer capability.

2. The PhoneStatusReading behaviour enables the Android agent to read and keep track of the phone status, using the PhoneStatusReading capability.

3. The NewActionReceived behaviour receives a command to carry out a certain action, and it has the NewActionReceived capability. It also has the ability to communicate with local agents to perform actions and change agent configurations.
4. The *InteractWUser* behaviour enables the user to interact with the agent and respond if it is necessary.

The Android agent has the following four types of capabilities:

1. *SubscribeToServer* is the capability to subscribe to the server and obtain the agent identification.

2. *PhoneStatusReading* is the capability to read phone status.

3. *NewActionReceived* is the capability to receive commands from the main server to perform an action to secure the smartphone.

4. *InteractWUser* is the capability that allows the Android user to interact with Android agent, in order to delete malicious applications from the smartphones.

Each capability receives intents. The Android agent has four types of tasks that are launched when an intent arrives and the corresponding condition is fulfilled:

1. The *SendSubsCommand* task sends the “subscribe” command to the server, in order
to obtain its services, by sending a *SubscribeRequired* event and handling a *SubscribeRequest* event.

2. The *NewPhoneStatus* task reads smartphone status by sending a *NewPhoneStatus* event and handling a *NewStatusAvailable* event.

3. The *SendActionCommand* task, used upon receiving a command to perform an action, sends an *ActionPerformed* event and handles a *NewActionAvailable* event.

4. The *SendUserCommand* task, used upon receiving a command from the server, sends an *InteractWUserRequest* event and handles an *InteractWUserRequired* event.

Finally, the Android agent handles eight events:

1. The *SubscribeRequired* event is sent to subscribe an agent to the server.

2. The *SubscribeRequest* event is posted by the Android agent when new agent needs to be subscribed.

3. The *NewPhoneStatus* event is sent to monitor the changes on the smartphone status.

4. The *NewStatusAvailable* event is posted by the Android agent when a new status is received.

5. The *ActionPerformed* event is sent to the agent to carry out a certain action based on a command received from the central agent.

6. The *NewActionAvailable* event is posted by the agent upon reading a new command.

7. The *InteractWUserRequired* event is sent to the Android user, requesting acknowledgement of the feedback from the central agent.

8. The *InteractWUserRequest* event is posted by the Android agent when an interaction with the user is requested by central agent.
4.4.2.2 Signature Detection Agent

The signature detection agent registers with the SMS profiling agent and obtains SMS signatures. This agent monitors incoming and outgoing text messages to collect reported suspicious SMS. Another function is to send suspicious SMS to SMS Profiling Services, for robust analysis. This agent is in charge of observing SMS logs and sending any changes to the SMS profiling agent. The signature detection agent obtains the outcomes from the SMS signature-based detection module and executes one of the following actions: display the normal SMS, block the malicious SMS, or report the suspicious SMS to the SMS profiling agent.
The signature update is the basis of any relation behaviour of known SMS botnets. These signatures can be screened by the signature detection agent to detect malicious SMS messages, using the signature-based detection module. The signature detection agent receives the characteristics of signature updates that can be used to detect SMS botnets. It can also request an update of the signature from the SMS profiling agent. In the current implementation, the security administrator can use the defence module to produce the new signatures and rules that are used by the signature-based detection module to evaluate the degrees of normality or abnormality of each incoming and outgoing SMS message. Attribute level (<<keyfield>> and <<valuefield>>) and method level (<<indexedquery>>) stereotypes are defined in Figure 4.11.

Figure 4.12 shows the implementation of signature detection agent, including the four following behaviours:
1. The AgentRegistration behaviour enables the agent to register, using the corresponding capability, to get all the services this agent offers.

2. The NewTextMessage behaviour reads SMS messages and performs signature detection on them.

3. The SignaturesUpdate behaviour updates the signatures upon receiving an update from the central agent.

4. The GetDetectionResults behaviour reads the results from the signature detection. It also has three capabilities, namely, DeliverNormalSMSToSMSApp, DeleteMaliciousSMS, and SendSuspiciousSMS.

The signature detection agent has the six following capabilities:

1. RegisterWithSMSProfile is the capability to register the signature detection agent with the SMS profiling agent in order to obtain all services.

2. NewSMSReading is the capability to send new SMS to the SMS signature-based detection module.

3. GetSignaturesUpdating is the capability to receive the signature update from the central agent to keep an up-to-date record of attacks.

4. DeliverNormalSMSToSMSApp is the capability to deliver normal SMS to the SMS application.

5. DeleteMaliciousSMS is the capability to delete malicious SMS, send a copy of the malicious SMS to the central agent, and notify the user.

6. SendSuspiciousSMS is the capability to send suspicious messages to the central agent for further analysis, which also notifies the user.

six tasks are used by the signature detection agent:
1. The *RegisterWithSMSProfile* task registers the signature detection agent with the SMS profiling agent, by sending a *RegistrationRequired* event.

2. The *SendNewSMSMessages* task sends the SMS to the signature-based detection module by sending a *NewSMSMessage* event and handling a *NewSMSAvailable* event.

3. The *SendGetSignatureUpdate* task obtains the signature update by sending a *GetSignatureRequest* event.

4. The *DeliverNormalSMS* task delivers the normal SMS to the SMS application by sending a *DeliverNormalSMS* event.

5. The *DeleteMaliciousSMS* task blocks the malicious SMS and sends a copy of the malicious SMS to the SMS profiling agent, by sending a *DeleteMaliciousSMS* event.

6. The *SendSuspiciousSMS* task sends the suspicious SMS messages to the SMS profiling agent, with user’s permission, by sending a *ReportSuspiciousSMS* event.

Finally, the signature detection agent handles eight events:

1. The *RegistrationRequired* event is sent to the SMS profiling agent that needs registration.

2. The *NewSMSMessage* event is sent to the SMS signature-based detection module.

3. The *NewSMSAvailable* event is posted when a new SMS is received.

4. The *GetSignatureRequired* event is sent to the central agent that needs to get the signature updates.

5. The *GetSignatureRequest* event is posted upon receiving an update.

6. The *DeliverNormalSMS* event is sent upon getting the results from the SMS signature-based detection module.
7. The *DeleteMaliciousSMS* event is used upon getting the results from the SMS signature-based detection module.

8. The *ReportSuspiciousSMS* event is enabled upon getting the results from the SMS signature-based detection module.

### 4.4.2.3 App-Profile Agent

The main task of the app-profile agent is to monitor the Android mobile device by reporting running applications, running services, granted permissions for installed applications, browser accessibility, connectivity time, SMS logs, battery usage, communication between applications and any changes to settings. In addition, it observes the Internet connection, since connectivity is a vital aspect that can probably contribute to our awareness of SMS botnet attacks. The app-profile agent works by registering with the Android profiling agent in the server. It responds to commands from the Android agent. This agent maintains a local copy of the applications profile information.
The app-profile agent has the following four behaviours as shown in Figure 4.13:

1. The `RegisterWithAndroidProfile` behaviour registers with the Android profiling agent to obtain all the offered services.

2. The `NewAppOrService` behaviour observes the applications and services on the smartphone.

3. The `MonitoringDeviceResources` behaviour monitors the smartphone’s resources.

4. The `AppProfileUpdate` behaviour sends application profile updates to the Android profile agent when upon receiving a request from the Android agent.

The app-profile agent has the following capabilities:

1. The `RegisterWithAndroidProfile` capability registers the app-profile agent with the Android profiling agent.

2. The `NewAppOrService` capability enables the app-Profile agent to monitor the applications and services.

3. The `MonitorDeviceResources` capability allows the app-profile agent to monitor the smartphone’s resources including the applications that access the SMS application, the browser use, the Internet connection, and the battery level; and

4. The `SendAppProfileUpdate` capability sends the updates to the Android profiling agent when a request arrives from Android agent.

Tasks are used by an app-profile agent as following:

1. The `RegisterWAndroidProfile` task request to registers this agent with the Android profiles agent by sending a `RegisterWAndroidProfile` event.

2. The `SendNewAppOrService` task sends the new application and services to the Android profiling agent to update the profiles.
3. The ReportAppAccessSMSApp task reports on the SMS application’s access to the applications and browser, and monitors the battery usages by sending a SendNewAppProfile event and handling a NewAppProfile event.

4. The SendAppProfileUpdating task sends an update to the Android profiling agent when a new request is received, by sending an AppProfileUpdate event and handling an AppProfUpdateAvailable event.

Finally, the following events are handled and posted by an app-profile agent:

1. The RegisterWAndroidProfile event subscribes to the Android profiling agent and establishes communication with the agents in the server.

2. The NewAppOrService event monitors applications and services.

3. The NewAppOrServiceRunning event is posted when a new application is installed or run.

4. The UpdateNewProfile event updates the local profiles and keeps track of profile changes.

5. The NewProfile event is posted when new profile data is received.

6. The AppProfileUpdate event reports the new profile update to the Android profiling agent, along with other local agents.

7. The AppProfUpdateAvailable event is posted upon completion of an update, to report the new profile to the Android profiling agent.

4.4.2.4 User-Profile Agent

Professional attackers have developed a smart way of performing malicious activities without users’ knowledge by setting their malware to run only when a device is in sleep or idle mode. To counteract this, the user-profile agent is in charge of observing user connectivity.
time by keeping track of the way a user uses his phone, and building a profile. In addition 
to keeping a local copy of user profiling information, this agent maintains a blacklist of 
phone numbers. After registering itself with the Android profiling agent in the server, this 
agent reports daily usage of the mobile device and responds to the Android profiling agent 
when that agent receives a command from the Android agent.

Figure 4.14 shows the overall design of the user-profile agent, including the following 
behaviours:

1. The RegisterToProfileService behaviour enables a user-profile agent to subscribe to 
the Android profiling agent.

2. The MonitorConnectivityTime behaviour monitors user connectivity time.

3. The UserProfileUpdate behaviour sends the user connectivity time to the Android 
profiling agent, by keeping track of the changes and the updates. The user connectiv-
ty time update is submitted simultaneously with the signature detection agent’s 
submission of the suspicious SMS messages to the SMS profiling agent.

4. The MaintainBlacklist behaviour maintains the blacklist of phone numbers and in-
forms the user of any malicious numbers in the user contact list.

The user-profile agent has the following capabilities:

1. *RegisterToProfileService* is the capability that subscribes the user-profile agent to the Android profiling agent to obtain its services and communicate with it.

2. *UserConnectivityTime* is the capability that monitors changes and keeps track of user connectivity time.

3. *SendUserProfileUpdate* is the capability that sends the update of the user connectivity and its profiles to the Android profiles agent

4. *MaintainBlackList* is the capability that maintains a blacklist phone numbers.

The user-profile agent has the following tasks:

1. The *RegisterWAndroidProfile* task subscribes the user-profile agent with the Android profiling agent.

2. The *SendUserProfileUpdating* task updates the local profiles and receives update requests from the Android agent. It responds to these requests by sending a *UserProfileUpdate* event and posting a *UserProfileUpdateAvailable* event.

3. The *UpdateUserConnectivity* task updates the local profile and keep track of changes by sending a *ConnectivityTimeUpdate* event.

4. The *BlackListUpdating* task receives updates of the blacklist by sending a *BlackListRequired* event.

Finally, the user-profile agent control the following events:

1. The *RegisterToAndroidProfile* event is sent to register with the Android profile agent.

2. The *UserProfileUpdate* event is sent to update user profile.
3. The `UserProfileUpdateAvailable` event is posted upon receiving an update event.

4. The `ConnectivityTimeUpdate` event is sent to submit the new update to the Android profile agent.

5. The `BlackListRequired` event is sent to maintain the blacklist of phone numbers.

### 4.5 JADE Agents Implementation

We have developed a multi-agent system, using JADE platform composed of JADE agents which are able to execute specified tasks and to communicate with each other. We use the JADE agent platform framework as a launch for the agent platform, which allows other agent containers to connect remotely to the agent platform as shown in Figure 4.2. One of the agent containers, the service provider, is used only with the anomaly-based detection module and the defence module. On the other hand, agent containers operate within Android application that runs on Android smartphone devices in order to create user profiles and to monitor Android smartphone resources for subscribing clients. In this section, we discuss the details of implementing agents on a service provider server and as well as the interaction between agents and activities on the Android platform.

#### 4.5.1 JADE Implementation for Service Provider Agents

Developing JADE Agents in a computer environment has received a lot of attention from researchers, since JADE is a well-known framework. All JADE libraries and documentation are available on the Tilab website\footnote{http://jade.tilab.com/}. In order to launch the JADE agent platform runtime as middleware, the JADE development environment needs to be built and set with variables by installing the Java Runtime Environment (JRE) and downloading the “jade.jar” and “commons-codec-1.3.jar” libraries. To launch a detection application with
agents in the server, MicroBoot and MicroRuntime must be declared in the extending MicroBoot class at the beginning as shown in Listing 4.1.

Listing 4.1: Startup class snipped code.

```java
import jade.MicroBoot;
import jade.core.MicroRuntime;
import jade.core.Agent;
import jade.util.leap.Properties;
...
public class Startup extends MicroBoot {
    ...
    public static void main(String args[]) {
        MicroBoot.main(args);
        ...
    }
    ...
    public void actionPerformed(ActionEvent e) {
        MicroRuntime.startAgent("CentralAgent", "agent.CentralAgentClient", null);
        MicroRuntime.startAgent("ProfileAgent", "agent.profileAgentClient", null);
        ...
        dispose();
    }
    ...
}
```

Within the JADE platform, defining a class that extends the jade.core.Agent class to create a JADE agent and agent implementation must be included in the setup() method, and the takeDown() method invoked before agent termination, in order to carry out agent clean-up operations, as shown in Listing 4.2. We defined three service provider agents in the server, all of which have unique names for identification. The getArgument() method of agent class is used to pass information to agents, and can be retrieved as an array of objects. The main goal of agents is to perform actions within “behaviours” [17]. Agents have the ability to execute several behaviours concurrently, and every agent uses two types of behaviour, namely “One-shot” and “Cyclic”. Cyclic behaviours are used to handle reported messages and to maintain the list of subscribed agents. One-shot behaviours send signature updates
and actions, based on the decisions that have been made in the server, to all signature detection agents, as shown in Figure 4.2.

In order to allow agents to execute tasks, initial behaviours can be added at any time using the addBehaviour() method that is defined in the setup() method. In the extending behaviour class, the action() method must be implemented to define the tasks to be carried out when the behaviour is being executed [17]. To allow agents to communicate with each other, messages exchanged have to follow the format of ACL language that is defined by FIPA international standards for multi-agent systems [15]. Using an ACLMessage object to send a message to another agent by invoking send() method and receive() method allows an agent to pick up messages, as shown in Listing 4.2. JADE implementation and functionalities are discussed extensively in the JADE programmer’s guide [16].
Listing 4.2: Central agent class snipped code.

```java
private ACLMessage reportedMessages;
...
protected void setup() {
    addBehaviour(new agentListener(this));
    // Initialize the message used to convey messages
    reportedMessages = new ACLMessage(ACLMessage.INFORM);
    reportedMessages.setConversationId(CentralAgent_ID);
    CentralGui = new CentralAgentGui(this);
}
...
protected void takeDown() {
    if (CentralGui != null) {
        CentralGui.dispose();
    }
}
...

class Manager extends CyclicBehaviour {
    private MessageTemplate temp;
    ...
    public void onStart() {
        ACLMessage agentlist = new ACLMessage(ACLMessage.SUBSCRIBE);
        agentlist.setConversationId(convId);
        ...
        CentralAgent.send(agentlist);
        temp = MessageTemplate.MatchConversationId(convId);
    }
    ...
    public void action() {
        ACLMessage receivedmessage = CentralAgent.receive(temp);
        ...
        if (receivedmessage.getPerformative() == ACLMessage.INFORM) {
            AbsPredicate pp = (AbsPredicate)
                CentralAgent.getContentManager().
                extractAbsContent(receivedmessage);
            ...
        }
    }
    ...

private class ReportMessage extends OneShotBehaviour {
```
4.5.2 JADE Implementation for Android Smartphone Agents

The first step of the design process is to define the roles of the agent as identified in Subsection 3.1.3. These roles can include sending and receiving information from each other, monitoring smartphone resources, and observing user usages in order to protect smartphones and prevent damage to them.

One of the JADE capabilities, which is fully supported by Android SDK, is the ability to send and receive an ACLMessage to or from agents running on different containers or on a remote JADE platform. To develop JADE agents on Android mobile devices, JADE add-on is an important element that allows the agent container to attach to a JADE main container. An extensive explanation of JADE for Android devices is provided by Bergenti et al. [19] and also by a meaningful example called Chat application that is illustrated by Ughetti et al. [135]. Developing JADE agents on Android is entirely different from developing in the normal environment but fortunately Jade-leap add-on was created to solve this issue of implementing the JADE runtime environment on handheld devices, and the issue of using wireless characteristics with the respect of the fixed network to deploy JADE agents in Android smartphones. During implementation, we must consider the communications among the Jade agents, interactions between agent and Android activity and the inter-operations among Android activities. We imported the “JadeAndroid.jar” library, and a Jade-leap add-on that provides JADE classes and some additional services. The library includes two main classes:
1. `jade.android.RuntimeService`, which has the capability to configure the JADE environment and to start or stop JADE runtime in order to connect or disconnect from the main container; and

2. `jade.android.MicroRuntimeService`, which allows wrapping of both full containers and split containers [28].

As shown in Listing 4.3, to begin the process, the SMS botnet detection application requests permissions on behalf of its agents in the manifest file. The defined permissions used by each agent are illustrated in Table 4.1. The MicroRuntimeService service is also declared in the Androidmanifest.xml as a service application component and Android activity bound with the service, as shown below. We developed a system that can run an application that uses JADE runtime which is wrapped by the Android service on Android devices supporting Android version 2.1 or higher.

Listing 4.3: Androidmanifest XML file snapshot.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<uses-permission android:name="android.permissionINTERNET" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.ACCESS.BATTERY_STATS" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<application>
<service android:name="jade.android.MicroRuntimeService" />
</application>
</manifest>
```

The next step is to bind MicroRuntimeService with Android activity, which activates
Table 4.1: Agent - permission mapping tables.

<table>
<thead>
<tr>
<th>Agents</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>android.permission.READ_PHONE_STATE</td>
</tr>
<tr>
<td></td>
<td>android.permission.INJECT</td>
</tr>
<tr>
<td></td>
<td>android.permission.ACCESS.BATTERY_STATS</td>
</tr>
<tr>
<td>Signature Detection</td>
<td>android.permission.RECEIVE_SMS</td>
</tr>
<tr>
<td></td>
<td>android.permission.READ_SMS</td>
</tr>
<tr>
<td></td>
<td>android.permission.WRITE_EXTERNAL_STORAGE</td>
</tr>
<tr>
<td>App-Profile</td>
<td>android.permission.GET_TASKS</td>
</tr>
<tr>
<td></td>
<td>android.permission.ACCESS_NETWORK_STATE</td>
</tr>
<tr>
<td></td>
<td>android.permission.WRITE_EXTERNAL_STORAGE</td>
</tr>
<tr>
<td>User-Profile</td>
<td>android.permission.READ_CONTACTS</td>
</tr>
<tr>
<td></td>
<td>android.permission.WAKE_LOCK</td>
</tr>
<tr>
<td></td>
<td>android.permission.WRITE_EXTERNAL_STORAGE</td>
</tr>
</tbody>
</table>

JADE runtime for smartphone to connect to JADE main container. The MicroRuntime-Service service allows jade.android.MicroRuntimeServiceBinder objects to be retrieved and to perform JADE management operations [28] (Listing 4.4). The advantage of using the service component is that it runs silently in the background due to the service always having higher priority than passive activities which make the application continue to run and respond to the user’s action in the background. The Android activity imports MicroRuntimeServiceBinder when the application launches, by using the binding bindService method.

Listing 4.4: MainActivity class snapshot.

```java
import MicroRuntimeService;
import MicroRuntimeServiceBinder;
...
final Properties p = new Properties();
p.setProperty(Profile.MAIN_HOST, host);
p.setProperty(Profile.MAIN_PORT, port);
p.setProperty(Profile.MAIN, Boolean.FALSE.toString());
p.setProperty(Profile.JVM, Profile.ANDROID);
...
...
if (microRuntimeServiceBinder == null) {
```
Before creating and starting a split container, there should already be a main container that can be accessed from the subscribed Android device running on a Jade platform. After retrieving the MicroRuntimeServiceBinder object, it can start to create a split container.

A snipped segment of code of the startContainer() method is shown in Listing 4.5. In order to connect to the main container, the host and port where the main container is hosted should be stated in the “properties” object.

Listing 4.5: startContainer() method snipped code.

```java
final Properties profile = new Properties();
profile.setProperty(Profile.MAIN_HOST, host);
profile.setProperty(Profile.MAIN_PORT, port);
profile.setProperty(Profile.MAIN, Boolean.FALSE.toString());
profile.setProperty(Profile.JVM, Profile.ANDROID);
if (AndroidHelper.isEmulator()) {
    // Emulator: this is needed to work with emulated devices
```
When JADE runtime is launched and running, an agent is started in the split container mode, as shown in the following snipped code (Listing 4.6).

Listing 4.6: StartAgent() method snipped code.

```java
    public void onSuccess(Void thisIsNull) {
      // Successfully start of the container.
      ...
    }
    ...
    public void onFailure(Throwable throwable) {
      // Failed to start the container
      ...
    }
  });
} else { // AgentStartupCallback
  ...
}
```

To implement the information exchange mechanism (interactions) between a JADE agent and Android activities, the O2A (Object-to-Agent) interface mechanism uses an Android component to allow the agent to expose one or more interfaces. The use of an extranet component to retrieve interfaces is a method that can be invoked to trigger an agent task [27]. The interaction between Jade agents and Android activity, and also between Jade agents in the Android device and agents in the service provider server can be categorized as following:

1. **Forward information from Jade agent in Android device to agents in service provider:** the agents have the ability to forward selected messages to specified agents in the service provider by using the `handleInteraction()` method. The behaviors of each agent are implemented in the `handleInteraction()` method, which adds behaviours to the agent and executes them. The `getServiceProviderAgentNames()` method is used to obtain the list of the service provider agents that can interact with and exchange the information. A code snippet of the `SMSDetectionInterface` class is shown in Listing 4.7.

   ```java
   public interface SMSDetectionInterface {
     public void handleInteraction(String s);
     public String[] getServiceProviderAgentNames();
   }
   ``

   **Listing 4.7:** `SMSDetectionInterface` class snipped code.

2. **Transfer information from Android activity to Jade agent:** using the agent `setup()` method allows agents to expose Object-to-Agent interfaces by the signature detection agent. This method can retrieve the application context and other objects as parameters to the agent. The snipped code of the `setup()` method is shown in Listing 4.8.

   ```java
   setup() method snipped code.
   ```
protected void setup() {
    // Add initial behaviours
    // Activate the GUI
    registerO2AInterface(SMSDetectionInterface.class, this);
    ...
    Intent broadcast = new Intent();
    broadcast.setAction("agent.SHOW_SMS");
    context.sendBroadcast(broadcast);
    ...
}

3. **Transfer information from Jade agent to Android activity**: The agents must display the outputs in GUI to allow the user to see the recommendations and the actions that should be performed by the Android user. Android activity uses the `onCreate()` method, which shows how the GUI component can retrieve the Object-to-Agent interface provided by the agent (Listing 4.9). When results are reported, the user will be able to see the results, which will then be sent to the service provider agent. In order to notify the GUI that a signature update has been received, a reported agent action shows the result to the user by creating and broadcasting an Intent. The `SMSDetectionActivity` registers a receiver to intercept Intents carrying received messages, and handles them (Listing 4.9).

Listing 4.9: `onCreate()` method snipped code.

...
for (int i = 0; i < listOfmalicious.getCount(); i++) {
    Detectedmessages += listOfmalicious.getItemAtPosition(i) + "\n";
}

for (int i = 0; i < listofsuspicious.getCount(); i++) {
    Detectedmessages += listofsuspicious.getItemAtPosition(i) + "\n";
}

for (int i = 0; i < listofnormal.getCount(); i++) {
    Detectedmessages += listofnormal.getItemAtPosition(i) + "\n";
}

if (Detectedmessages != null && !Detectedmessages.equals("")) {
    try {
        SMSDetectionInterface.handleSpoken(Detectedmessages);
        listOfmalicious.setFilterText("");
        listofsuspicious.setFilterText("");
        listofnormal.setFilterText("");
    }
}

private class MyReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (action.equalsIgnoreCase("agent.REFRESH_SMS")) {
            final TextView resultField = (TextView) findViewById(R.id.SMSTextView);
            SMSField.append(intent.getExtras().getString("Detectedmessages"));
        }
    }
}

4.5.3 Deploying JADE Agents

The agent deployment diagram shows where Android mobile device agents and service provider agents are deployed. The agent deployment diagram for the SMS botnet detection
The integration of JADE makes it more solid and sophisticated; however, to cope with the limitations and constraints of smartphones and wireless networks, and to make it compatible with Dalvik JVM, the JADE-leap add-on is required in order to make the capability of executing an agent more efficient and reliable on Android mobile devices.

For Android mobile devices, four agents are deployed on each Android emulator running Android version 4.1.2, which is used to collect data to build the Android profile and the user profile, and to observe SMS to detect SMS botnets. On the other hand, the server side has three agents that offer services to Android device agents. In the server, the JADE platform starts up and runs the main container to allow each Android agent and the server agents to register or deregister with the DF (JADE Directory Facilitator). Also in the server, three agents start at the same time to respond to Android agents, as explained above. All the agents that are operating on the server stay active all the time to provide services to subscribed Android agents. For each Android device, the Android user has to choose a name for the Android agent and send a subscription request to the server. Split containers launch and start four agents that are connected to targeted service provider agents. Each agent obtains information on Android activity and transfers it to its service provider agent.

Initially, Android agents interact with local agents to make sure they are running. A
signature detection agent acquires SMS signatures and stays active to receive any signature update. In addition, this agent scans current text messages and labels all the SMS. If there is a malicious SMS, it is deleted; when a suspicious SMS is found, it is sent to the server. Also, this agent monitors incoming and outgoing text messages. If any malicious SMS or suspicious SMS are detected, this agent requests current profiles from app-profile and user-profile agents to be sent with the detected text messages. The app-profile agent creates an Android profile that includes all the features that need to be analyzed in order to investigate the suspicious SMS and to spot any abnormal activities. The names and descriptions of these features are defined in Table 3. Also, this agent responds to any request coming from the Detection Agent and then sends the requested information to the Android profiling agent on the server. The user-profile agent keeps track of the user connectivity time and builds a user profile, as well as responding to any commands from the detection agent. Both profiling agents maintain a local copy of the constructed profiles and interact with the Android profile agent on the server.

4.6 Concluding Remarks

In this chapter, we described the design and implementations of JADE agents that are used in the development of multi-agent system. The JADE agents operate based on their beliefs about the current status of the smartphones and use their predefined tasks and capabilities to cope with real world intrusion detection and automated response problems. The JADE agents also monitor and collect defined features on mobile devices and then send them to a server to perform behaviour analysis in order to detect abnormal activity and detect SMS botnets. We use a multi-agent system to allow agents to interact with each other and stay active to protect Android mobile devices. The next chapter, chapter 5, presents the proposed framework evaluation and discusses the results and findings of the experiments.
Chapter 5

Experiments and Results

The proposed solution framework has been discussed in chapter 3. In this chapter, experiments were conducted to evaluate the framework performance, and the experimental results are discussed. So, here we begin by describing the reference datasets used for the experiments, and providing the evaluation result of the proposed framework system. To determine the capability of the multi-agent system, signature-based detection, anomaly-based detection module, and defence module in accurately detecting SMS botnets, we conducted different experiments in three phases.

In the first phase, we focused on evaluating the efficiency of the signature-based detection module in Android devices and provided details of the experiment. A comprehensive performance analysis of the anomaly-based detection module is conducted in the second phase. This includes a detailed study on the performance of this module. In the last phase, we analyzed the overall performance of our proposed framework and provided a thorough analysis of JADE agents monitoring mechanism after demonstrating the capability of each module individually.
5.1 Experimental Data Sets

In order to evaluate the accuracy of our proposed approach, it was imperative to select relevant datasets. Fortunately, there are five well-known public datasets that we were able to consider in our experimental study. The DIT SMS Spam Dataset [40] contains 1,353 spam SMS text messages that were collected from two UK public consumer complaints websites. Each message was stamped with the date it was reported on and with the source that it was extracted from. The collection period of this corpus is from late 2003 to the middle of 2010. This dataset is provided in XML format and we wrote a script to extract the text messages, date, number, and source of spam text messages.

The SMS spam collection dataset [8] contains labelled spam and normal SMS messages. The dataset has 5,574 English SMS messages labelled as spam and normal. 14% of the messages are spam while the rest are ham. They were collected from various resources and from a smaller previous collection (SMS Spam Corpus v.0.1). The details of SMS spam collection dataset are provided in Almeida et al. [8].

We also report the experimentation results on the IIIT-D SMS Spam Dataset [148] provided by the smsAssassin creators, which has 2,000 English SMS messages labelled as spam and ham. The dataset was primarily collected using two different modes (i.e. intensified crowdsourcing and personal contacts). We received the dataset in two folders and each SMS messages is in a text file. To create the dataset, we wrote a script to merge and randomize the messages.

We additionally evaluated the proposed framework on the British English SMS Corpora [90] that has 875 SMS messages labelled as spam and legitimate. 49% of SMS messages are spam that was collected from GrumbleText [54] and the rest of SMS messages are legitimate messages that were collected from Caroline Tags PhD Thesis [130]. We use the script to merge the messages and randomize them.

We report the experimentation results on NUS SMS Corpus [32] which is an unlabelled dataset and has about 55,000 English text messages. These SMS messages were collected
using different methods such as Web-based transcription, SMS export, SMS upload, Amazon Mechanical Turk, and free SMS website. Since SMS often contains both personal and confidential information, such as telephone numbers and email addresses, NUS SMS Corpus was given anonymity when the SMS were submitted. More details and comprehensive study of the collection can be found at [32]. Table 5.1 summarizes the details of the five datasets.

Table 5.1: Details of the datasets used for experiments.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Dataset Creator</th>
<th>Labelled</th>
<th>Unlabelled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT SMS Spam</td>
<td>Delany et al. [40]</td>
<td>0</td>
<td>1,353</td>
</tr>
<tr>
<td>smsSpamCollection</td>
<td>Almeida et al. [8]</td>
<td>4827</td>
<td>747</td>
</tr>
<tr>
<td>IIIT-D SMS Spam</td>
<td>Kuldeep et al. [148]</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>British English SMS</td>
<td>Nuruzzaman et al. [90]</td>
<td>450</td>
<td>425</td>
</tr>
<tr>
<td>NUS SMS Corpus</td>
<td>Chen et al. [32]</td>
<td></td>
<td>55,835</td>
</tr>
</tbody>
</table>

To provide a comprehensive evaluation of the proposed framework, we collected a large set of SMS botnet samples that has seven botnet families. These families describe the behaviour of SMS botnets. The SMS botnet dataset collected botnet samples from the Android Genome Malware project [154], malware security blog [36], VirusTotal [140], and samples provided by a well-known anti-malware vendor.

A variety of SMS malware used to send premium-rate SMS without user knowledge was analyzed. These SMS were extracted, along with the Tophone#, and added to our dataset. In addition, we created a dataset that has malicious URLs, commands, and phishing words. We used Android mobile botnet dataset that were collected using different sources to extract the URLs by analyzing APK files. We also gathered Android botnet URLs from the UNB ISCX Android Botnet Dataset [2]. The botnet commands were extracted from the dataset samples by analyzing apk and from other sources [86], [14], and [65], [121]. Also we created a phishing words list from datasets in Table 5.1. We collected over 150 words, which are considered phishing words. The summary of extracted C&C and URLs are illustrated in Table 5.2.
Table 5.2: Overview of the extracted C&C and URLs.

<table>
<thead>
<tr>
<th>Botnet Family</th>
<th>Total samples</th>
<th>C&amp;C</th>
<th>URLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>DroidDram</td>
<td>336</td>
<td>14</td>
<td>500</td>
</tr>
<tr>
<td>Geimi</td>
<td>266</td>
<td>4</td>
<td>157</td>
</tr>
<tr>
<td>MisoSMS</td>
<td>101</td>
<td></td>
<td>195</td>
</tr>
<tr>
<td>NickiSpy</td>
<td>203</td>
<td>16</td>
<td>139</td>
</tr>
<tr>
<td>NotCompatible</td>
<td>76</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>PjApps</td>
<td>213</td>
<td>7</td>
<td>189</td>
</tr>
<tr>
<td>Pletor</td>
<td>85</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Rootsmart</td>
<td>33</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>TigerBot</td>
<td>97</td>
<td>10</td>
<td>26</td>
</tr>
<tr>
<td>Wroba</td>
<td>94</td>
<td>10</td>
<td>36</td>
</tr>
<tr>
<td>Zitmo</td>
<td>81</td>
<td>23</td>
<td>29</td>
</tr>
<tr>
<td>Other Malware Family</td>
<td>1054</td>
<td>275</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>2639</td>
<td>379</td>
<td>1297</td>
</tr>
</tbody>
</table>

These datasets took into consideration the following SMS threats:

- New SMS spam: SMS spam is any junk message delivered to a mobile phone as an SMS message. One of the activities that a botnet engages in is to send an SMS spam message that makes an end-user helpless to block the number of any SMS spam they receive.

- SMS with malicious URLs: This kind of SMS is employed to propagate malware by tricking users and infecting their devices by using URLs to link to an application (APK file), which is then downloaded by the user. A well-known botnet is called Zeus-in-the-Mobile (ZitMo) malware [78]. Once ZitMo is installed within a device, it infects other devices by sending forged SMS messages from the original device, allowing other users to install an infected update.

- SMS with Command and Control (C&C) instructions: Bot masters have the ability to control users’ devices and steal user information by using SMS to communicate with infected devices. Bot masters can issue specific commands to control the bot, such as “find node and send sysinfo” [150]. For instance, command communications with the CHULI malware [14] are utilized to send stolen data to C&C servers.
• Sending unauthorized SMS: Malware can be used to send premium-rate SMS messages without user knowledge at specific times. There are a growing number of apps in the Android platform; some of these apps misuse the outgoing SMS permission and send malicious SMS. For example, Zsone malware is implemented to send SMS to specific premium numbers, which is costly to affected users [126].

• Phishing words: Attackers attempt to trick Android phone users into opening URLs, which then become injected with malware, or into performing calls to a premium phone number. Attackers send SMS with malicious URLs associated with phishing words such as “amazing”, “free”, and “download”. In one case, Koler malware sent SMS with malicious URLs, for example, “someone made a profile named -Luca Pellicari- and he uploaded some of your photos! is that you? http://bit.ly/xxxxxx“ [155]. We consider uploaded as a phishing word that will be flagged as suspicious SMS even though the URL is not on our blacklist.

In order to determine the efficiency of the proposed framework, we use samples from the Android Malware Genome Project dataset [154] and Contagio malware dump [36] and modify them for testing purpose that has two Special Characteristics can send premium rate SMS messages and Specific Operational Times. We repackaged a real-world malware sample known as AndroidOS/Fakeplayer.A [77]. This malware pretends to be a movie player and shows a messages in Russian. It sends SMS messages and contains the string “798657” to Russian SMS short code numbers (3353 and 3354) that may charge the user without their knowledge. We changed the short code number from 3353 and 3354 to “1-555-521-5562”. We also developed an application called “DroidDreamTest” that can send out SMS messages at certain times. This application has similar silent patterns as DroidDream that only operate from 11 pm to 8 am [127]. We were able to monitor a Test application and its behaviour relating to SMS being sent out at specific times when the device at sleep mode and send out SMS without the user’s permission. If the same SMS message is reported by more than one Android agents and it has the SMS botnet
characteristics, it is flagged as malicious.

5.2 Evaluation Measures

To differentiate between anomalous and normal data, anomaly detection approaches are usually employed in order to evaluate the proposed framework. The confusion matrix is the best way of representing the classification result (Table 5.3). Due to the two-class nature of the detection, there are four measures as follows:

Table 5.3: The confusion matrix.

<table>
<thead>
<tr>
<th></th>
<th>Legitimate</th>
<th>Malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Legitimate</strong></td>
<td>TN</td>
<td>FP</td>
</tr>
<tr>
<td><strong>Malicious</strong></td>
<td>FN</td>
<td>TP</td>
</tr>
</tbody>
</table>

- True positive (TP): represents a malicious message detected correctly as malicious.
- False negative (FN): refers to a malicious message detected incorrectly as legitimate.
- True negative (TN) represents a legitimate message detected correctly as legitimate.
- False positive (FP) refers to a legitimate message detected incorrectly as malicious.

In addition, we measure the performance of different detection modules using the standard metric true positive rate and false positive rate. The standard metrics extract part of the information from the confusion matrix to produce some numeric value. The higher the true positive rate and the lower the false positive rate, the better the detection is.

1. True Positive Rate (TPR): TPR is the proportion between the malicious messages detected correctly as malicious. (Equation 5.1)

\[
TPR(\text{recall}) = \frac{TP}{TP + FN} \quad (5.1)
\]
2. False Positive Rate (FPR): FPR is the ratio between the number of misclassified normal messages and the total number of normal messages. (Equation 5.2)

\[
FPR = \frac{FP}{FP + TN}
\]  

Precision and recall have been commonly used in the literature to measure how well the detected objects correspond to the reference objects [6]. We also use accuracy, precision, recall f-measure and MCC. The higher the values of these measures are, the better the detection. In the following we explain each of the measures used.

- **Accuracy**: This metric represents the total number of text messages that are correctly classified including normal and malicious SMS messages. (Equation 5.3)

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN}
\]  

- **Precision**: P is a metric that shows the proportion of the predicted malicious SMS messages that were correct. The aim of the framework is to obtain a high precision that means the number of false alarms is minimized. (Equation 5.4)

\[
Precision = \frac{TP}{TP + FP}, precision \in [0,1]
\]  

- **Recall**: it is the proportion of malicious SMS messages that were correctly detected. It is calculated using the same equation as the true positive rate. Thus, it is anticipated a classifier will have a high recall value [81]. (Equation 5.5)

\[
Recall = \frac{TP}{TP + FN}, Recall \in [0,1]
\]  

- **F-measure**: Precision, and Recall metrics do not absolutely explain the accuracy of an IDS, although a combination of them would be more suitable to use. F-measure
is the harmonic average of precision and recall. (Equation 5.6)

\[ F\text{-measure} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \]  

(5.6)

Due to using the accuracy metric as an evaluation measurement, the F-measure is chosen. With high precision and recall, the F-measure would have maximum values that mean that the classifier has low false alarms and a high detection rate. Consequently, the F-Measure of a classifier is preferred to be as high as possible [53].

- The Matthews correlation coefficient (MCC): MCC is used in machine learning as a measure of the quality of binary (two-class) classifications and uses comparative analysis. (Equation 5.7)

\[ MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \]  

(5.7)

In addition, we assessed the significance of the proposed framework. Consequently, we will conduct incremental and ablation experiments first. In the following sections, we explain how these experiments are conducted and the results of each. In phase three, we also used end-to-end testing approaches to evaluate the performance of the multi-agent system [119]. The performance measurements for the JADE agent in the experiment are CPU usages and memory usages.

### 5.3 Phase 1: Signature-Based Detection Module Evaluation

In order to evaluate the signature-based detection module, we created signatures based on the DIT SMS Spam Dataset [40] and the Android botnet samples from VirusTotal [140], which are labelled data sets. Table 5.4 summarizes the number of extracted features from
We evaluated the performance of the algorithm described in Section 3.2.2 using the SMS Spam Collection Dataset [8] and the NUS SMS Corpus [32] by loading all the SMS messages to an Android emulator and determining the accuracy of our approach in classifying the SMS text messages.

As is shown in Figure 3.6, we first ran the feature extractor and then applied our SMS signature detection algorithm to classify the SMS messages into three categories:

1. Normal: the SMS messages are legitimate text messages and do not match any of the defining features.

2. Suspicious: the SMS signature detection algorithm spots the messages that match the defining set of rules as shown in Table 3.5.

3. Malicious: the SMS messages that have corresponding signatures are malicious messages.

If a message was determined to be suspicious, we added a time stamp with an explanation of the SMS labelled “suspicious” to enable the Android user to make a final decision based on the information we provided. The experiments were performed on an emulator running Android version 4.1.2, in which we installed our application to perform the detection on real SMS loaded into the emulator.

5.3.1 Experimental Results

For the experiments, we used the datasets described in Section 5.1. The signature set contains 1,350 text messages and two test sets consisting of 5,574 real label SMS text
messages, as well as 5,085 unlabelled SMS text messages. We applied a real-time content-based mechanism using the signature-based approach for SMS classification to identify whether the SMS is malicious, suspicious, or normal. We carried out two experiments

5.3.1.1 SMS Spam Collection Dataset Experiment

In the first experiment, we used the SMS Spam Collection Dataset [8]; the result of the signature detection approach for the labelled dataset is shown in Table 5.5. The proposed solution detected 747 malicious SMS, in which we have the SMS body signatures for 740 SMS text messages. Of the seven remaining SMS messages detected, one of the malicious text messages has malicious URLs, while six of the malicious text messages have malicious phone numbers. Also, we found two suspicious SMS messages that were labelled as normal SMS in the SMS Spam Collection Dataset [8]. The first text message contained a URL and the second SMS message was flagged as suspicious commands because it was only one word and began with string “#”. This reflects the JokerBot command, which is an Android-based botnet that begins with the string “#!” used to control the bot [65].

Table 5.5: SMS spam collection dataset experimental results.

<table>
<thead>
<tr>
<th>Types</th>
<th>Features</th>
<th># of SMS</th>
<th>Total</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malicious</td>
<td>SMS body</td>
<td>740</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Phones#</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>URLs</td>
<td>1</td>
<td>747</td>
<td>13.5%</td>
</tr>
<tr>
<td></td>
<td>Commands</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FromPhone#</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ToPhone#</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Phones#</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suspicious</td>
<td>URLs</td>
<td>1</td>
<td>2</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td>Commands</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td></td>
<td>4825</td>
<td></td>
<td>85.5%</td>
</tr>
</tbody>
</table>
5.3.1.2 NUS SMS Corpus Experiment

In the second experiment, we employed a non-labelled dataset and selected 5,085 text messages. Table 5.6 illustrates the result of the signature detection methods for the unlabelled dataset. The signature detection approach did not detect any malicious SMS that matched the signatures, but our approach did detect 349 suspicious SMS messages that match the patterns and the defined set of rules as shown in Table 3.5. Eleven of the text messages were detected as suspicious SMS because they have suspicious phone numbers, while 13 of the text messages were detected as suspicious SMS because of suspicious URLs. Additionally, 325 of the suspicious SMS messages were detected due to suspicious commands. Android users have to take full responsibility for deciding whether to delete the suspicious SMS or not based on the explanation that we have provided with suspicious SMS. Further analysis needs to be performed on the suspicious SMS, for which we use agents to forward all malicious and suspicious SMS messages and their profiles to service provider agents.

Table 5.6: NUS SMS corpus experimental results.

<table>
<thead>
<tr>
<th>Types</th>
<th>Features</th>
<th># of SMS</th>
<th>Total</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SMS body</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Phones#</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malicious</td>
<td>URLs</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Commands</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FromPhone#</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ToPhone#</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suspicious</td>
<td>Phones#</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>URLs</td>
<td>13</td>
<td>353</td>
<td>6.9%</td>
</tr>
<tr>
<td></td>
<td>Commands</td>
<td>329</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td></td>
<td>4736</td>
<td>93.1%</td>
<td></td>
</tr>
</tbody>
</table>

5.3.1.3 Validation of Results

To validate the experimental results, the experimental findings were reviewed manually to ensure that our detection approach detected only truly malicious SMS. We found 747
malicious SMS text messages and 351 suspicious SMS text messages. It can be determined from the results presented in Table 5.5 that our signature-based detection module achieves 100% detection rate with zero false negatives (FN) during the detection of SMS botnets in the SMS Spam Collection Dataset [8]. Additionally, the results presented in Table 5.6 have been verified manually to ensure our detection method achieves optimum accuracy with the NUS SMS Corpus [32]. Due to the limitations of the SMS Spam Collection Dataset, we were not able to detect “FromPhone#” and “ToPhone#”.

5.4 Phase 2: Anomaly-Based Detection Module Evaluation

In this section, we evaluate the proposed detection approach using the standard metric. The SMS botnet anomaly-based detection module receives the reported SMS messages and Android profiles, and then performs detection using a clustering technique, a classification approach, a profiles analysis approach, and a rule-based correlation method. The algorithms were implemented in Java and Weka software used for clustering approaches. All experiments were conducted using an Intel Core i7 of CPU @ 3.10GHz machine with 16 GB and running Windows 7 Professional 64-bit.

We performed the experiments using various datasets. The input to the anomaly-based detection module consisted of three types of data: well-known datasets, reported SMS messages, and reported Android profiles. First, it was vital to select appropriate datasets to evaluate the precision of the proposed approach. For the experiments, we chose two public datasets and our C&C command dataset:

1. SMS Spam Collection Dataset [8], a labelled dataset that has 747 spam SMS and 4,827 normal SMS;
2. IIIT-D SMS Spam Dataset \cite{[148]}, a labelled dataset which has 1,000 spam SMS messages and 1,000 normal SMS; and

3. The third dataset, our C&C command set, we formed by analyzing several malware families and extracting commands from the list of botnets. We also gathered a number of commands from other sources \cite{[86]}. After selecting datasets, we reported the experiment results on the real-time signature detection approach in Section 5.3 using NUS SMS Corpus. We chose 5,085 text messages and built a test application that would send out SMS messages while the smartphone is in sleep mode.

We loaded the SMS messages to an Android emulator and ran our real-time signature-based detection in order to label SMS as malicious, suspicious or normal. The signature detection produced 353 unknown SMS messages which we called suspicious, and we reported these to the anomaly-based detection module, along with their Android profiles, for further analysis. The third input data is the Android profiles, which were collected from Android devices as the collected feature described in table 3.6 which had to be processed in order to analyze the profiles containing reported SMS messages.

5.4.1 Evaluation Methodology

In order to get the anomaly-based detection module to perform well and to detect SMS botnets intelligently, we used four steps to detect SMS botnets as follows:

- First, take the labelled datasets that contain malicious and normal SMS \cite{[8]} and cluster them based on content similarities using the X-means algorithm. The result of the clustering produces a number of clusters that are analyzed and categorized into four class labels.

- Second, use the 353 reported SMS messages that need to be classified into one of the four class labels using the SMS classification approach.
• Third, apply profile analysis to the Android profiles using aggregation and prioritization techniques to produce an abnormal profile table (APT).

• Finally, apply rule-based correlations to SMS messages in the four label classes and the profiles outputs in order to label each message in each class label as a malicious or normal message.

### 5.4.2 Experimental Results and Findings

For performance evaluation, we evaluated the detection performance using over 13,000 SMS text messages. The following seven metrics were employed to evaluate the performance of the detection [145]: accuracy, precision, true positive rate (recall), False negative rate, True negative rate, False positive rate, and F-measure. To find the optimal value for each parameter, we perform different rounds of experiments. The accuracy is averaged over 10 runs with random initialization of prototype extraction.

We conducted three experiments, the first experiments were performed with the SMS spam collection dataset [8]. We divided the SMS spam collection dataset into two parts: a first part for training where 750 text messages were normal and 250 text messages were malicious, and the rest of the dataset for testing. The second experiment used the IIIT-D SMS Spam Dataset [148], which we divided into 80% for training and 20% for testing. In the third experiment, we evaluated this module by using the 353 reported text messages with related reported Android profiles.

#### 5.4.2.1 First Experiments Results

The aim of this framework is to identify SMS botnet. To intelligently detect SMS botnet, we decided to perform two experiments. The experiment A excludes pre-process steps and the experiment B applies pre-process steps.

1. Experiment A: we used 5,574 evaluated text messages [8] without applying any pre-
Table 5.7: The experiment A confusion matrix.

<table>
<thead>
<tr>
<th></th>
<th>Legitimate</th>
<th>Malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legitimate</td>
<td>4719</td>
<td>108</td>
</tr>
<tr>
<td>Malicious</td>
<td>68</td>
<td>679</td>
</tr>
</tbody>
</table>

process methods. The confusion matrix illustrates the four metrics values as shown in Table 5.7; a total of 108 legitimate SMS messages were incorrectly detected as malicious with 68 false negatives, which is malicious SMS messages detected as normal. Table 5.9 shows the results of Recall, Precision, and F-measure of anomaly-based detection module. The anomaly-based detection module achieved a 96.84% rate of accuracy, with 97.76% high recall and 98.58% precision. The proposed anomaly-based detection module obtained 9.1% of false positive rate and 2.2% of false negative rate. F-measure scores the balance between precision and recall. The F-measure is a measure of the accuracy of a test and our proposed anomaly-based detection module achieved 98.2%.

Table 5.8: The experiment B confusion matrix.

<table>
<thead>
<tr>
<th></th>
<th>Legitimate</th>
<th>Malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legitimate</td>
<td>4624</td>
<td>106</td>
</tr>
<tr>
<td>Malicious</td>
<td>84</td>
<td>663</td>
</tr>
</tbody>
</table>

2. Experiment B: we pre-processed the dataset and ended up with 5,477 text messages. In the preprocess step, we apply stop words removal methods and stem word methods. Table 5.8 is shown the confusion Matrix of experiment B. We obtained 106 false positives, which are legitimate SMS messages detected as malicious, and 84 false negatives. Table 5.9 shows the detection performance results. The accuracy obtained by the anomaly-based detection module experiment was 96.53%, with 97.76% recall and 98.22% precision. The proposed anomaly-based detection module obtained 11.2% of false positive and achieved 2.2% of false negative rate. The tradeoff between the precision and recall is referred to as the F-measure. In this
experiment, the F-measurement score is 98%.

Table 5.9: First experiments anomaly-based detection module performance.

<table>
<thead>
<tr>
<th>Detection Metric</th>
<th>Experiment A</th>
<th>Experiment B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.968</td>
<td>0.965</td>
</tr>
<tr>
<td>Precision</td>
<td>0.986</td>
<td>0.982</td>
</tr>
<tr>
<td>Recall (TPR)</td>
<td>0.978</td>
<td>0.978</td>
</tr>
<tr>
<td>FNR</td>
<td>0.022</td>
<td>0.022</td>
</tr>
<tr>
<td>TNR</td>
<td>0.909</td>
<td>0.888</td>
</tr>
<tr>
<td>FPR</td>
<td>0.091</td>
<td>0.112</td>
</tr>
<tr>
<td>F-measure</td>
<td>0.982</td>
<td>0.980</td>
</tr>
</tbody>
</table>

These experiments results show the capability of the anomaly-based detection module. In addition, we are comparing the results of the previous two experiments. As illustrated in Figure 5.3, the two experiments obtained a small difference in accuracy and precision but have similar recall results. The idea of the anomaly-based detection module is to detect malicious SMS messages especially (C&C) instructions that are sent through SMS service by attackers. We decided not to consider the pre-process steps due to the fact that botnet commands sometimes use stop words. For example, the following botnet command “How are you?” that was used by Android/SmsHowU.A malware [86]. This malware sends a location using GPS and Google Maps link to current geographic location via SMS. The botnet command has “you”, which is consider as a stop word, as defined in Section 3.3

Figure 5.1: First experiment results comparison.
In intrusion detection, normal data usually outnumber intrusion [53]. For instance, with 95% normal and 5% attack, the accuracy metric is misrepresentative due to the probability that a system will rank a randomly chosen positive instance higher than a randomly chosen negative one. In this case, a system always classifies all data as normal with a high accuracy (97% in our example).

We reported the results of the experiments that had different percentages of malicious SMS messages. We built 10 different data sets that had the following percentage of malicious SMS message 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, and 90%. For example the first data set has 95% of SMS messages are normal and 5% of Messages are malicious. Figure 5.2 shows the capability of the proposed module. With changing the percentage of amount malicious SMS messages, the anomaly-based detection module achieved 95.68% precision and 87.54% recall when 20% of SMS messages are malicious. Although when malicious SMS messages exceed 60% of dataset, the persistent results are increased and the recall result are decreased.

5.4.2.2 Second Experiment Results

In this experiment, we evaluated the proposed module on the IIIT-D SMS Spam Dataset [148] that has 2,000 SMS messages [148]. The anomaly-based detection module obtained 77 malicious SMS messages that were detected as normal (FN) and 57 normal text messages...
were labelled as malicious (FP). The evaluation metrics are computed for the dataset and the obtained result for all attacks and normal data are given in Table 5.10, which is the overall detection performance of the proposed anomaly-based detection module on the IIIT-D SMS Spam Dataset. By analyzing the result, the overall performance of the proposed system is improved significantly and it achieves more than 93% accuracy for all types of attacks. The proposed module obtained 94.3% precision and 92.5% recall. The anomaly-based detection module with this dataset achieved 5.7% of false positive rate and 7.7% of false negative rate. The score of the F-measurement, which this module has achieved is 93.4% in this dataset.

Table 5.10: The detection performance of the proposed anomaly-based detection module.

<table>
<thead>
<tr>
<th>Detection Metric</th>
<th>2nd experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.933</td>
</tr>
<tr>
<td>Precision</td>
<td>0.943</td>
</tr>
<tr>
<td>Recall (TPR)</td>
<td>0.925</td>
</tr>
<tr>
<td>FNR</td>
<td>0.077</td>
</tr>
<tr>
<td>TNR</td>
<td>0.923</td>
</tr>
<tr>
<td>FPR</td>
<td>0.057</td>
</tr>
<tr>
<td>F-measure</td>
<td>0.934</td>
</tr>
</tbody>
</table>

5.4.2.3 Third Experiment Results

In the second experiment, we had a set of reported text messages that had not yet been classified as normal or malicious. We applied our SMS classification to 353 SMS messages and determined that 100% of the suspicious SMS were labelled as malicious SMS messages. As shown in Table 5.11, of these; 325 text messages were considered C&C instructions that are similar to the commands of JokerBot and CHULI malware and included some dangerous permissions; 11 text messages were identified as malicious phones number; 13 text messages had malicious URLs; and four text messages were labelled as malicious because the SMS messages were sent out while the device was in sleep mode. In addition,
the anomaly-based detection module demonstrated the ability to detect blacklist phone numbers and to label them as malicious.

Table 5.11: Third experiment findings.

<table>
<thead>
<tr>
<th>Experiment name</th>
<th>SMS body</th>
<th>URLs</th>
<th>Phone number</th>
<th>C&amp;C instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd experiment</td>
<td>4</td>
<td>13</td>
<td>11</td>
<td>325</td>
</tr>
</tbody>
</table>

5.4.2.4 Comparative Evaluation

In Table 5.12, we compare the results of the anomaly-based detection module approach with those of the author who released an SMS Spam Collection Dataset [8] and with the results of the CLUTO framework [138]. We compared our results with others’ work based on the following performance measures: Spam Caught (SC%), Blocked Hams (BH%) Accuracy (Acc%) and Matthews Correlation Coefficients (MCC) [138]. Other authors have applied well-known machine learning techniques for detecting spam SMS messages. Based on evaluation of the information in Table 5.12, the linear SVM previously achieved the best results with an accuracy of 97.64%. It was able to detect 83.10% of all spam, blocking only 0.18% of legitimate SMS messages. However, our anomaly-based detection module was able to detect 90.9% of all malicious SMS messages, blocking only 2.24% of legitimate SMS messages and achieving an accuracy level of 96.84%.

Table 5.12: Comparison with other results.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>SC</th>
<th>BH</th>
<th>Acc</th>
<th>MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>83.10%</td>
<td>0.18%</td>
<td>97.64%</td>
<td>0.893</td>
</tr>
<tr>
<td>Boosted NB</td>
<td>84.48%</td>
<td>0.53%</td>
<td>97.50%</td>
<td>0.887</td>
</tr>
<tr>
<td>Boosted C4.5</td>
<td>82.91%</td>
<td>0.29%</td>
<td>97.50%</td>
<td>0.887</td>
</tr>
<tr>
<td>PART</td>
<td>82.91%</td>
<td>0.35%</td>
<td>96.26%</td>
<td>0.887</td>
</tr>
<tr>
<td><strong>Our Approach</strong></td>
<td><strong>90.9%</strong></td>
<td><strong>2.24%</strong></td>
<td><strong>96.84%</strong></td>
<td><strong>0.867</strong></td>
</tr>
<tr>
<td>MDL</td>
<td>75.44%</td>
<td>0.35%</td>
<td>96.26%</td>
<td>0.826</td>
</tr>
<tr>
<td>CLUTO</td>
<td>79.51%</td>
<td>4.62%</td>
<td>93.25%</td>
<td>0.781</td>
</tr>
<tr>
<td>C4.5</td>
<td>75.25%</td>
<td>2.03%</td>
<td>95.00%</td>
<td>0.770</td>
</tr>
<tr>
<td>Bern NB</td>
<td>54.03%</td>
<td>0.00%</td>
<td>94.00%</td>
<td>0.507</td>
</tr>
</tbody>
</table>

The summary of the detection performance of SMS Spam Collection and IIIT-D SMS spam
datasets in Figure 5.3. First experiment results of the SMS spam collection dataset show the capability of the anomaly-based detection module in providing comparable detection performance accuracy compared to the second experiments. The detection performance of the anomaly-based detection module has an average accuracy of 95%. It obtained an average precision of 96.5% and an average recall of 95.2%. The detection average of false negative rate is 3.95% on applied datasets.

![Figure 5.3: First and second experiments results comparison.](image)

To validate the results, the experimental findings were reviewed manually to ensure that our detection approach detected only truly malicious SMS. In the experiment A, we found 679 malicious SMS messages and we found 923 malicious messages in the second experiment. In the third experiment, we found 353 malicious text messages and the anomaly-based detection module achieved a 100% detection rate with zero false negatives (FN). Additionally, the results of the fourth experiment have been verified manually to ensure our detection method achieves optimum accuracy with NUS SMS Corpus and our dataset of SMS botnets.
5.5 Phase 3: SMS Botnet Detection Framework Evaluation

The persistence of this phase of evaluation is to demonstrate the effectiveness of applying hybrid detection using a multi-agent system on the overall performance of the proposed framework. A subscribed Android mobile device is where signature detection is performed and the agents scan, monitor, and observe smartphone’s features and then report the data to the server. A service provider server is used to collect data including reported SMS messages and Android user profiles. It is also carries out extensive anomaly detection, and then generates new signatures for new threats. The central agent forwards the decision that has been made to a subscribed Android device that allows the Android user to take the most appropriate action.

5.5.1 Evaluation Framework

In order to evaluate the proposed framework, we tested our implementation using the Android platform and ran real-time behaviour monitoring for the collected features as described in Table 3.3. We carried out different experiments to evaluate the overall framework that included a signature-based detection module, an anomaly-based detection module, and a multi-agent system that was composed of data collection agents and service provider agents. The performance testing for an agent-based system is end-to-end testing [119]. The performance measurements for JADE agent in the experiment are CPU usages and memory usages. The experiment consists of four parts. The first part evaluates the detection of abnormal SMS messages on smartphones and reports to the server to perform an anomaly detection. The second part assesses the detection approach by monitoring Android features and creates profiles that are sent by an app-profile agent. In the third part, we study the scenario where malicious applications try to send out SMS messages at specific time and send to a premium-rate phone number by mimicking human
behavior. The experiment shows that our proposed framework is still robust in this situation. At last, we discuss the application prototype in terms of its performance related to CPU and memory, battery usage, and loss of connection with an agent main container.

### 5.5.1.1 Abnormal SMS Detection

SMS botnet detection is the main goal, with detected suspicious SMS messages confirmed as either malicious or normal SMS. We developed a signature detection agent to scan existing messages and observe incoming and outgoing text messages. This signature detection agent performs the scan, obtains the SMS log, and then sends commands to app-profile and user-profile agents to gather information at the same time to determine if there are any malicious or suspicious SMS messages. This data was transferred to the service provider for further analysis. In the service provider, the SMS profile agent receives the reported SMS with additional associated information. The signature-based detection module required signatures to be frequently updated in order to detect malicious SMS messages. To generate signatures, we used five types of datasets as shown in tables 5.13 and the details of these datasets are provided in Section 5.1.

#### Table 5.13: The number of signatures records.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Content</th>
<th>URLs</th>
<th>Phone Num</th>
<th>Phishing Words</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIT SMS Spam</td>
<td>1353</td>
<td>95</td>
<td>1165</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IIIT-D SMS Spam</td>
<td>1000</td>
<td>129</td>
<td>304</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SMS Spam Collection</td>
<td>747</td>
<td>104</td>
<td>490</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>British English SMS</td>
<td>425</td>
<td>56</td>
<td>248</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Our dataset</td>
<td>0</td>
<td>824</td>
<td>0</td>
<td>170</td>
<td>397</td>
</tr>
</tbody>
</table>

In this experiment, we used two datasets. The first dataset is a labelled dataset called the British English SMS that has 425 malicious text messages. The second dataset is the NUS dataset that has over 55,000 unlabelled text messages to evaluate the proposed framework. First, the experiment used the British English SMS; we loaded all SMS messages and ran our application prototype, then reported the results. In the second experiment, we randomized the NUS dataset that has total of 55,835 text messages that have the threats
that are defined in Section 5.1. We divided the 55,196 SMS messages into 11 sets, each set having approximately 5,000 SMS messages, as shown in Tables 5.14. We used 11 Android emulators to load each set to an emulator and ran our SMS botnet detection application prototype. If a suspicious SMS is detected, the signature detection agent sends it to the SMS profiling agent and then sends commands to other local agents requesting the current profile be sent to the service provider.

Table 5.14: The NUS experiments datasets.

<table>
<thead>
<tr>
<th># of sets</th>
<th># of SMS messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>set #1</td>
<td>5003</td>
</tr>
<tr>
<td>set #2</td>
<td>5002</td>
</tr>
<tr>
<td>set #3</td>
<td>5008</td>
</tr>
<tr>
<td>set #4</td>
<td>5000</td>
</tr>
<tr>
<td>set #5</td>
<td>5000</td>
</tr>
<tr>
<td>set #6</td>
<td>5024</td>
</tr>
<tr>
<td>set #7</td>
<td>5000</td>
</tr>
<tr>
<td>set #8</td>
<td>4999</td>
</tr>
<tr>
<td>set #9</td>
<td>5002</td>
</tr>
<tr>
<td>set #10</td>
<td>4987</td>
</tr>
<tr>
<td>set #11</td>
<td>5171</td>
</tr>
</tbody>
</table>

In addition, this agent makes an initial decision regarding the suspicious SMS by interacting with other subscribed Android mobile device agents and reported responses from local agents. The decision is based on one of the following factors:

1. The app-profile agent reports any running application that has SMS permissions as described in Table 5.15 or

2. The user-profile agent responds with any non-human behaviour when SMS is detected, for example, SMS is sent out while the user is inactive.

5.5.1.2 Monitoring of Android Features

Creating Android user profiles require monitoring Android features that malicious apps used to initial an attack. To test this approach, the app-profile agents obtained the list of
installed applications and running applications. The app-profile agent also monitors the
granted permissions and tracks any permissions related to SMS permissions, as well as
Internet connection permissions. Table 5.15 shows the permissions that the agent keeps,
observes and monitors. This includes accessing the state of the network and Internet
permissions. In addition, this agent observes services that are running as an important
element to determine which service is an application component that can carry different
types of operations that run in the background.

Table 5.15: Monitored permissions.

<table>
<thead>
<tr>
<th>Permission Name</th>
<th>Permission Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECEIVE_SMS</td>
<td>to monitor incoming SMS</td>
</tr>
<tr>
<td>SEND_SMS</td>
<td>to send out SMS</td>
</tr>
<tr>
<td>READ_SMS</td>
<td>to read current SMS</td>
</tr>
<tr>
<td>WRITE_SMS</td>
<td>to write to SMS Content Provider</td>
</tr>
<tr>
<td>BROADCAST_SMS</td>
<td>to broadcast an SMS notification</td>
</tr>
<tr>
<td>INTERNET</td>
<td>to obtain full access to the Internet</td>
</tr>
<tr>
<td>ACCESS_NETWORK_STATE</td>
<td>to access ConnectivityManager</td>
</tr>
<tr>
<td>CHANGE_NETWORK_STATE</td>
<td>to change network state</td>
</tr>
<tr>
<td>ACCESS_WIFI_STATE</td>
<td>to access WifiManager</td>
</tr>
<tr>
<td>CHANGE_WIFI_STATE</td>
<td>to change Wi-Fi connectivity state</td>
</tr>
<tr>
<td>WAKE_LOCK</td>
<td>to monitor if the process is a wake</td>
</tr>
</tbody>
</table>

Malicious applications have the feature of reading and interacting with other installed
applications. We are keeping track of the interactions between the applications, and
monitoring any request that is sent to the web browser. Furthermore, we are observing
the use of Android mobile resources, including CPU usage, memory usage, battery usage,
and network usage. In case of low battery level, the agents have the ability to adjust their
behaviour and notify the Android users to recharge their smartphones. We have created
an Android user profile based on described features in Sections 3.3.4 that must be sent to
the Android profiling agent. If the app-profile agent receives commands from the android
agent, this agent responds with the result and the android agent has to take an action.
This agent also transfers the current profile to the Android profiling agent in the service
provider to perform profiling analysis.
5.5.1.3 Human Behaviour Observation

The user-profile agent is responsible for observing human behaviour. Attackers monitor user usage of the Android device in order to launch an attack during specific times (for example, when the user is sleeping or inactive). Based on this point, we decided to observe user behaviour, including user connectivity time, to monitor the time the phone is in wake or sleep modes. For instance, any SMS sent out while the smartphone is in sleep mode is flagged as malicious SMS and the information regarding the application that sends the SMS is recorded by user-profile agent. We are also monitoring user usage of phone services, such as SMS, calls and use of the Internet, as well as the users’ contact list, in order to manage the whitelist and blacklist. We evaluate the capability of these agents by using our test application and determined if the agent is monitoring whether the SMS is sent from the mobile device while it is in wake mode and at specific period of time. Also, the user-profile agent sends information on the human behaviour upon receiving request from Android agent to the android profiling agent in the service provider.

5.5.1.4 Android Application Evaluation

The JADE agent platform is launched on server and the service provider agents begin offering the services for the subscribed smartphones. The evaluation environment consisted of a server and Android emulators that run on Intel Core i7 3.10GHz, 16GB RAM, Windows 7 Professional 64-bit. The SMS application is installed on emulators with Android version 4.2.1. The performance testing is only through the emulator sd the JADE platform is a well known platform that has been comprehensively and independently evaluated and its performance evaluation is beyond the scope of this thesis. We use the end-to-end method to evaluate an Android device using an Android developer kit (SDK). SDK provides powerful tools to evaluate the developed application, as follows:

- Android Debug Bridge (adb) [128] is a useful command line tool that allows us to connect with an emulator instance in order to control and debug Android devices.
Dalvik Debug Monitor Server (DDMS) [42], is a debugging tool included with SDK and Android Studio. This tool provides a variety of features that can be used for performance testing, which include port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more. DDMS can be launched from Eclipse or command line.

All four types of agents that are running on Android smartphone have different goals to achieve. After installing our SMS botnet detection application prototype, the Android agent receives the signatures from central agent and SMS signature detection agent scans existing SMS messages and monitor incoming and outgoing SMS messages. This agent reported the detected SMS messages to the SMS profiling agent and other local agents reported their profiles to the Android profiling agent. These are the peak of CPU and memory intense tasks in the proposed framework. We examined our Android application prototype by using the performance metrics that ran tasks and started the profiling method that supported by DDMS to monitor the CPU and memory usages.

We have developed script that uses two Android command line tools [42]:

- “top” [42] that is used to monitor the CPU usage of the android device
- “procrank” [23] that is used to show a summary of process memory utilization including virtual set size (Vss), resident set size (Rss), proportional set size (Pss) and unique set size (Uss), sorts by Vss. We only considered the two important parameters of Android application processor Pss and Uss.

We started the process of the performance test by running agent containers in the server and in the Android emulator. After that the script recorded the CPU and memory usages then started the agents to achieve the following tasks, which we performed each task individually in order to ensure the efficiency of the proposed framework:

- The Android agent obtains signature updates from central agent.
• The signature detection agent performed signature detection on existing SMS messages and got the results back.

• The results forwards SMS profiling agent and the app-profile and user-profile agents get the current profiles and forward them to Android profiling agent.

5.5.2 Experimental Results and Discussion

In this section, we present the results that have been obtained by SMS signature-based detection module and anomaly-based detection module, we also discuss the performance of the JADE agents on smartphones.

5.5.2.1 Signature-Based Detection Module Results

In smartphones, the signature detection agents obtained the results of the signature detection and then requested other agents to report the profiles to the service provider agents with malicious and suspicious SMS. For the first experiment, we show the results of the signature detection on the British English SMS [90], shown in Tables 5.16. The signature detection agent reported 425 malicious SMS to SMS profiling agent in service provider server. The signature-based detection module detected 179 SMS messages that have corresponding content signatures and 417 phone number are malicious which match the phone number signatures. The signature-based detection module spotted 56 malicious URLs and this dataset does not have any botnet commands. The detection accuracy on British English SMS dataset is 100% with zero false alarm since, we have signature. The malicious SMS will be blocked and removed without any further analysis.

Table 5.16: The proposed framework experimental results.

<table>
<thead>
<tr>
<th>Types</th>
<th>SMS content</th>
<th>Phones#</th>
<th>URLs</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious</td>
<td>170</td>
<td>417</td>
<td>56</td>
<td>0</td>
</tr>
</tbody>
</table>

In the second experiment, Figure 5.4 shows the experiment distribution results of signature-
based detection module for each set. Also, the summary of NUS dataset signature detection results are shown in Table 5.17. The signature detection agents send 3,115 suspicious SMS text messages and 165 malicious SMS messages to the SMS profiling agent and sends commands to app-profile and user-profile agents requesting the current profiles be sent to the Android profiling agent. 139 of SMS messages contained C&C botnet commands that have corresponding command signatures and 26 malicious SMS messages have malicious URLs. The signature detection agents reported 869 suspicious phone numbers, 144 suspicious URLs, and 2,182 suspicious commands. These features match defined rules as described in Table 3.5.

Figure 5.4: The experiments distribution result of signature detection.

Table 5.17: The proposed framework experimental results.

<table>
<thead>
<tr>
<th>Types</th>
<th>Features</th>
<th># of SMS</th>
<th>Total</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td></td>
<td></td>
<td>51721</td>
<td>94%</td>
</tr>
<tr>
<td>Malicious</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMS body</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Phones#</td>
<td></td>
<td>3</td>
<td>3</td>
<td>0.5%</td>
</tr>
<tr>
<td>URLs</td>
<td></td>
<td>23</td>
<td>165</td>
<td>0.5%</td>
</tr>
<tr>
<td>Commands</td>
<td></td>
<td>139</td>
<td>139</td>
<td>13.9%</td>
</tr>
<tr>
<td>FromPhone#</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>ToPhone#</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Suspicious</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phones#</td>
<td></td>
<td>869</td>
<td>869</td>
<td>86.9%</td>
</tr>
<tr>
<td>URLs</td>
<td></td>
<td>144</td>
<td>3081</td>
<td>5.5%</td>
</tr>
<tr>
<td>Commands</td>
<td></td>
<td>2182</td>
<td>2182</td>
<td>21.8%</td>
</tr>
</tbody>
</table>
5.5.2.2 Anomaly-Based Detection Module Results

In the service provider server, where all agents reported the SMS messages and its profile. We started by combining all the datasets that are described in Table 5.1 and we removed duplicated SMS messages from the datasets. Although SMS spam messages are characterized by obfuscation, we kept many of the non-identical messages that might still be close matches. We randomized spam SMS messages and normal SMS messages and chose 500 normal SMS messages and 500 spam SMS messages.

Table 5.18: The SMS clustering and classification results.

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>M</th>
<th>MM</th>
<th>N</th>
<th>MN</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset SMSes</td>
<td>338</td>
<td>107</td>
<td>153</td>
<td>402</td>
<td>1000</td>
</tr>
<tr>
<td>Malicious SMSes</td>
<td>165</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>165</td>
</tr>
<tr>
<td>Suspicious SMSes</td>
<td>56</td>
<td>39</td>
<td>2891</td>
<td>95</td>
<td>3081</td>
</tr>
</tbody>
</table>

We used the same evaluation methodology described in Subsection 5.4.1. In the first step, we clustered 1000 SMS messages using X-mean clustering technique and then applied our clustering analysis method, as described in Subsection 3.3.2. The results of the SMS clustering algorithm are described in Table 5.18. In the second step, we applied an SMS classification algorithm by taking 165 reported malicious SMS messages that were received from signature detection agents and classified all malicious SMS to malicious class labels which help to classify suspicious SMS messages. The SMS classification algorithm also classified the 3,081 reported suspicious SMS messages to one of four class labels. The result of the classification are given in Table 5.18. For the NUS dataset, 2891 of SMS messages are classified as normal and 95 SMS messages are classified as majority normal. 56 SMS messages are labelled as malicious and 39 SMS messages are labelled as majority malicious.

In the third steps, we analyzed the reported profiles by applying the Android profiles analysis algorithm that produces the abnormal profile table. The illustration of profile analysis are described in Subsection 3.3.4. In the fourth step, we employed SMS correlation
algorithm that applied correlation rules to label the instances in each one of four class labels. Table 5.19 shows the results of the detection on NUS dataset. 941 of SMS messages are labelled as malicious and 2,152 of SMS messages are labelled as normal.

Table 5.19: The anomaly-based detection module results for NUS dataset.

<table>
<thead>
<tr>
<th>Label</th>
<th>Total</th>
<th>Phones#</th>
<th>URLs</th>
<th>Commands</th>
<th>Phishing Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious</td>
<td>941</td>
<td>818</td>
<td>136</td>
<td>281</td>
<td>144</td>
</tr>
<tr>
<td>Normal</td>
<td>2152</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.5.2.3 JADE Agents Evaluation Results

For performance evaluation, we tested the implementation using a set of collected features. The experiments were performed on emulators where JADE agents were running and a server which runs the main container of the JADE platform. After the user registers with the service provider, the detection agent scans the current SMS and monitors incoming and outgoing SMS messages. App-profile and user-profile agents start to observe activity on Android devices and all agents interact with their service provider.

![Figure 5.5: CPU usages for obtaining signatures update.](image)

It is important to understand how the agent gathers the data and obtains the signature
updates. The agent does not gather the data every second and the agent collects the signature update once it has received an update request. Following the procedure described in Subsection 5.5.1.4, we evaluated the proposed application prototype in the emulator and produced diagrams showing the results in two evaluation measures (CPU and memory usages). We obtained the results for three tasks: Figure 5.5 and Figure 5.6 show the CPU and memory usages to obtain signatures update at first time with over 6,600 signatures as described in Table 5.4.

From these charts related to signature updates, we can observe that the prototype application tends to consume more CPU power and more memory while communicating with the central agent to obtain signature update the first time. It took around 383 seconds to get 6,600 signature records and deliver them to Android agents. Figure 5.5 shows linear trending of the agent communications in order to obtain the signature update from the central agent. Due to the large size of the signature updates, it takes more than 400 seconds to get the update. The average of CPU usages to obtain signature updates is 32.18% and the linear trendline is shown in Figure 5.5. The linear equation 5.8 that describes the CPU usage to get the signature update is the following:

\[ y = 0.0002x + 0.2719 \]  \hspace{1cm} (5.8)

On the other hand, Figure 5.6 shows exponential increased until 280 seconds and decreases after that then trends to stay at the same level. The average memory usages of Pss is 19,066 with 17,109k average of Uss usages. The equation 5.9 shows the exponential increase of the memory usage until 280 seconds and a falling trend between 280 seconds and 292 seconds. This trend shows the signature updates required a lot of memory usage during a period of time.

\[ y = 13284e^{0.0043x} \]  \hspace{1cm} (5.9)
To address this shortcoming, we decided to pack the signature updates with an application prototype and frequently update the signature when a new threat has been discovered.

In the second and third tasks, Figure 5.7 and Figure 5.8 are a summary of CPU and memory usage of the signature detection algorithm for 200 SMS messages and the reported results of three suspicious SMS messages with related profiles to service provider agents. To perform signature detection for 200 SMS messages required less than 60 seconds with 38% average of CPU power and 23,911K of Pss usages and 22,062k of Uss of memory usage.

As showing in Figure 5.7, there are two trends of CPU usages, the peak between 10 to 83 seconds which is the signature detection performance and the second peak is 91 seconds to 119 seconds when the app-profile and user-profile agents update the current profiles in smartphone, and report the profiles and the abnormal SMS messages to service provider. Since the values of CPU usage is continuously changing and has fluctuating values, the Polynomial trendline suits the trend of the CPU usages. The Polynomial equation \(5.10\) is described as following.
It is the same as in Figure 5.8 showing two peaks, one for signature detection and the other one for reporting the results by all agents. Equation \(5.11\) shows the memory usage
is continuously changing.

\[ y = 0.1115x^4 - 10.489x^3 + 305.05x^2 - 2720.2x + 26704 \]  

(5.11)

5.5.2.4 Discussion

We are focusing on detecting malicious applications that misuse the Short Message Service (SMS). Considering the privacy of users' information, the proposed approaches always show the user what the agent will send and get the user’s permission to send it. We have collected all the features that we think will play an important role in identifying these malicious applications. These features are critical to user privacy and the user may be concerned about the data that the agents capture. One solution is to give the user full decision-making power on whether or not to allow the agents to send the information to the service provider. Our approach provides full details about the detected SMS to the user, who may take action by deleting the SMS and removing all the reported applications.

The goal of the proposed framework is to protect SMS from malicious interference. Google has improved the use of SMS provision starting with Android version 4.4 but attackers always have ways to obfuscate Android security, and thousands of mobile devices are still using older Android versions.

To avoid loss of connection, Android agents are in charge of confirming that the data are received and stored safely in the service provider. JADE has the ability to maintain the connection between agents and has a store-and-forward mechanism; if the connection is lost, the agent will re-establish the connection as soon as it is available and re-transmit the data as soon as the connection is up [19].
5.6 Concluding Remarks

In this chapter, we presented the comparative results of the proposed framework on diverse datasets that included English text message datasets and Android malware samples. In the first phase, we focused on the signature-based detection module, in which we achieved a 100% detection rate with zero false negatives (FN) on the SMS spam collection dataset. The experimental result showed the capability of the signature-based detection module as a first defence line to detect SMS botnets in smartphones. A second phase was performed with the aim of evaluating the anomaly-based detection module using well-knows datasets and evaluated our clustering, classification, profiling analysis, and correlation algorithms. We found the Android user profiling and correlation approaches to be additional methods that significantly increase the detection performance. The results show that the our anomaly-based detection module obtained a high detection rate of 95% while keeping the false alarm rate as low as 3.95%. Having studied the performance of each module individually, in the last phase, we analyzed the overall performance of our proposed framework applying intelligent agents to monitor Android platform. The conducted experiments show a significant success in detecting SMS botnet and malicious applications. The next chapter concludes the thesis and gives possible of future works from this thesis.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we have proposed an SMS-based botnet detection framework that uses multi-agent technology based on observations of SMS and Android smartphone features. The proposed detection framework is based on a multi-layer model which consists of three processing modules with the use of JADE agents: 1) SMS signature-based detection; 2) intrusion detection; and 3) a defence module. In addition, Multi-agent technology is a powerful tool that can monitor certain environments and report abnormal behaviour in order to protect user data.

The inability of signature-based detection approaches in recognizing unknown attacks has encouraged many researchers to build automated detection approaches using anomaly-based detection, behavioural profiling analysis, and rule-based correlation techniques in order to detect an SMS botnet. We employed JADE agents in order to monitor, observe, and protect the subscribed smartphones and then built a profile for further investigation to confirm whether the SMS message has C&C instruction. In this research, we improved the content-based detection approach by extracting extensive domain knowledge in the form of general and high-level rules. We also built an Android user profile that has a set
of features related to the application behaviour and the user behaviour in order to detect the different characteristics of a mobile-based botnet. By cross-correlating the results of the anomaly detection and profile analysis, we were able to create rules for summarizing malicious behaviour of SMS messages. These correlation rules have higher discriminative power that are used to label an SMS message with its profile to malicious, normal, or required more analysis.

In the proposed framework, we have focused on the SMS content features, applications, service and Android resources features. The current implementation of the framework covers four main components that work intelligently together to provide full protection and mitigation against SMS botnet activities. Based on the results of the SMS signature-based detection module, the JADE agents reported the Android user profiles that are required to be used with suspicious SMS message to identify if it is malicious or not. These rules can be updated over time by finding new information about new botnet behaviour, and includes specific characteristics of malware that has unique behaviour in defence module. JADE agents depend on human knowledge and the set of rules that are programmed into them to make intelligent decisions autonomously. They continuously observe the smartphones, perceptively analyze and hold the characteristics of the abnormal behaviour, and autonomously respond to it. When an attack is recognized, there is often a variety of possibilities regarding where and how a response can be deployed. The choice of the actual response action or response locations can have both positive and negative impacts on the functionality of the network.

Having prepared our data set, we conducted the experiments in three phases to have a detailed comparison of the proposed system. The evaluation of the proposed framework shows that the SMS botnet detection performed obtained a high detection rate with the use of profile analysis. In phase 1, we demonstrated the performance of a real-time signature-based detection approach for SMS botnets, in which we applied pattern-matching detection for incoming and outgoing SMS and rule-based techniques to label unknown SMS as
suspicious or normal redundant. We obtained high detection rates with zero false alarms. In phase 2, we designed a module to detect SMS botnets and to correlate the android user profiles with SMS text messages in order to stop the harm caused by these attacks. We described the design of our module as having the ability to recognize SMS botnets in Android smartphones by performing behavioural profiling analysis that is reported from smartphones. The anomaly-based detection module is comprised of three different detection approaches namely: anomaly detection, correlation, and behavioural analysis, which operate to detect an unknown SMS botnet and its activities. A clustering method is used to group SMS messages based on the similarities between the SMS text messages, classification is used to classify current reported SMS to one of the class labels; behavioural profiling analysis is employed to carry out a robust and efficient detection that takes into account the requested permissions and behaviour of Android devices, and rule-based correlation of cluster outputs with profiles is used to reconstruct attack scenarios and label SMS messages in each class as normal or malicious. The module also provides specific details about type of attacks.

Having studied the performance of each module individually, in the last phase, we analyzed the overall performance of our proposed framework employing a multi-agent system to monitor Android user behaviour, applying signature-based detection to filter incoming and outgoing SMS messages, and using an anomaly-based detection module to perform anomaly detection and profiling analysis and to detect any correlation between SMS messages and its profile, identifying the superlative defence strategies to prevent and mitigate against mobile-based botnet. The conducted experiments show a significant improvement in protecting smartphones against SMS botnets.
6.2 Future Work

The work performed in this thesis provides a basis for future research of intrusion detection systems based on a multi-agent system in mobile devices. This section presents many different alternatives for future research.

- One area of future work is applying a broader range of features for intrusion detection. These features need to be calculated in real-time to enable the detector to keep up with the large number of reported SMS messages and their profile. Moreover, customized machine learning methods should be devised to minimize the CPU and memory consumption of the anomaly detector. It is also beneficial to employ a powerful random sampling method to reduce the huge number of SMS messages that are fed to the system as the training set.

- One of the areas that benefits from extension of improvement is the SMS classification algorithm. Although cosine similarity has proved to be effective, assigning the probabilities is quite challenging. This can be done by either applying more features or by utilizing a more efficient classification technique.

- In addition to experimenting on a larger dataset, costly components of the framework should be modified to make it scalable. Parallelization can be performed to provide better run-time performance on common multi-core systems.

- The JADE agents in Android platform have limitation resources such as agent mobility. An interesting concept to consider for future work is to explore the idea whether an agent can move from one Android device to another.

- Another interesting area that can be investigated in the future is to extend the framework. If an agent can make decisions on the fly about suspicious SMS messages. These decisions are based on other agents findings if they report the same SMS message with its characteristics.
• Finally, agent trust in mobile devices is another promising area for exploration. Building an agent trust model will aid more benefit in protecting the framework from compromising an agent. If an agent has been compromised, our framework would vulnerable to gain an access and perform man-in-middle attacks. One of the suggestions in case of an agent has been compromised, a backup agent will act behalf of the compromised agent in order to achieve the goals and reported the attack to central agent. Another suggestion is the authenticity and the integrity of the agent shared among different smartphones would need to be acknowledged because all agents could be compromised and unknowingly submit unreliable data. An authentication mechanism would be required among the cooperating networks
Bibliography


[60] Jingyu Hua and Kouichi Sakurai, A sms-based mobile botnet using flooding algorithm, Proceedings of the 5th IFIP WG 11.2 International Conference on Information Se-


Classification, review and open issues, Transactions on Internet and Information Systems 9 (2015), 1471–1492.


190


International Conference on Mobile Ubiquitous Computing, Systems, Services and

[136] F. Valeur, Giovanni Vigna, C. Kruegel, and R.A. Kemmerer, *Comprehensive ap-
proach to intrusion detection alert correlation*, IEEE Transactions on Dependable

sity of California at Santa Barbara, 2006.

and mining user-generated contents, ACM, 2011, pp. 27–34.

[139] Jignesh Vania, Arvind Meniya, and HB Jethva, *A review on botnet and detection
 technique*, International Journal of Computer Trends and Technology 4 (2013), 23–
29.


[141] Ping Wang, Sherri Sparks, and Cliff C. Zou, *An advanced hybrid peer-to-peer botnet*,
Proceedings of the First Conference on First Workshop on Hot Topics in Under-

[142] Yong Wang, Kevin Streff, and Sonell Raman, *Smartphone security challenges*, jour-

[143] Yu Weihong and Yan Chen, *The development of jade agent for android mobile
phones*, Proceedings of the 2012 International Conference on Information Technology


Vita

Candidate’s full name: Abdullah J. Alzahrani

University attended:
University of New Brunswick, Fredericton, NB, Canada
Ph.D. Candidate, Faculty of Computer Science, started September 2011
Thesis Topic: SMS-Based Botnet Detection Framework Using Intelligent Agents
Advisor: Dr. Ali A. Ghorbani

Lawrence Technological University, Southfield, MI, United States
M.Sc. in Information Security, May 2008
Thesis Topic: The Diagnostic and Statistical Manual of Mental Disorders (DSM) System
Advisor: Prof. Ghassan Azar

Umm Al-Qura University, Makkah, Saudi Arabia
B.Eng. in Computer Engineering, September 2005
Thesis Topic: Mechanical Tools Classification System
Advisor: Dr. Khalid Al-handi

Publications:
Journal Articles:


Refereed Conferences:


**Technical Reports:**

**Seminars and Expos:**


