LEBANESE AMERICAN UNIVERSITY

Lightweight Intrusion Detection for Mobile Devices

By

Fawzi Breidi

Thesis submitted in partial fulfillment of the requirements for the Degree of

Master of Science in Computer Science

School of Arts and Sciences

May 2012

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences

LAU Beirut campus School of Arts and Sciences
24 MAY 2011
Office of the Dean
RECEIVED

Thesis Proposal Form (Annex I)

Name of Student: Fawzi Breidi I.D.#: 200400542

Division: Computer Science and Mathematics

On (dd/mm/yy) 23/05/11, has presented a Thesis proposal entitled:

Lightweight Intrusion Detection System for Mobile Devices

in the presence of the Committee members and Thesis Advisor:

Dr. Sanaa Sharafeddine [redacted] 23/05/2011
(Name, signature, and date of the Thesis Advisor)

Dr. Nashat Mansour [redacted] 23/5/2011
(Name, signature, and date of Committee Member)

Dr. Faisal Abu Khzam [redacted] 23/5/2011
(Name, signature, and date of Committee Member)

Comments/Remarks/Conditions to Proposal:

Date: 24/05/11 Acknowledged by [redacted]
(Dean of Graduate ~~Studies~~/School of .....)

cc: Division Chair
Thesis Advisor
Student
Dean of Graduate Studies/School Dean

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Lebanese American University

School of Arts and Sciences - Beirut Campus

Thesis Defense Result Form

Name of Student:	Fawzi Breidi	I.D.:	200400542
Program / Department:	MS in Computer Science / Dept. of Computer Science and Mathematics		
Date of thesis defense:	May 23, 2012		
Thesis title:	Lightweight Intrusion Detection for Mobile Devices		

Result of Thesis defense:

☐ Thesis was successfully defended. Passing grade is granted

☒ Thesis is approved pending corrections. Passing grade to be granted upon review and approval by thesis Advisor

☐ Thesis is not approved. Grade NP is recorded

Committee Members:

Advisor: Dr. Sanaa Sharafeddine [redacted] 23/5/2012
(Name and Signature)

Committee Member: Dr. Nashat Mansour [redacted] 23/5/2012
(Name and Signature)

Committee Member: Dr. Faisal Abu Khzam [redacted] 23/5/2012
(Name and Signature)

Advisor's report on completion of corrections (if any):
Rewrite methodology in a more systematic way.
Summarize results and include in the experimental discussion.

Changes Approved by Thesis Advisor: [redacted] Signature: _____

Date: _____ Acknowledged by _____
(Dean, School of Arts and Sciences)

cc: Registrar, Dean, Chair, Thesis Advisor, Student

Lebanese American University

School of Arts and Sciences - Beirut Campus

Thesis Approval Form

Student Name: Fawzi Breidi I.D. #: 200400542

Thesis Title: Lightweight Intrusion Detection for Mobile Devices

Program: MS in Computer Science

Department: Department of Computer Science and Mathematics

School: School of Arts and Sciences

Approved by:

Thesis Advisor: Dr. Sanaa Sharafeddine [redacted]

Committee Member: Dr. Nashat Mansour

Committee Member: Dr. Faisal Abu Khzam

Date: 23.05.2012

THESIS COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY NON-EXCLUSIVE DISTRIBUTION LICENSE

By signing and submitting this license, you (the author(s) or copyright owner) grants to Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. You agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. You also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Name: *Fawzi Y. Breidi*

Signature: 

Date: *23-05-2012*

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that I have read and understood LAUs Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me. This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: *Fawzi Y. Breidi*

Signature: 

Date: *23-05-2012*

Acknowledgment

This research would not have been possible without the help and assistance of many persons. First I would like to express my gratitude to my supervisor Dr. Sanaa Sharafeddine for her guidance and support throughout my Thesis work. A thanks is also to Dr. Faisal Abu Khuzam and Dr. Nashaat Mansour for being on my thesis committee.

Dedication Page

I dedicate this work first and foremost to my parents for their love and support throughout my life and quest for higher education, for without their dedication and persistence I would've have been able to join one of the finest and most reputable universities in the country and this work would have been delayed for several years if ever completed.

Thesis / Lightweight Intrusion Detection for Mobile Devices

Fawzi Y. Breidi

Abstract

Current mobile devices are capable of providing connectivity anytime anywhere while supporting multitude of services and applications. This luxury of all-time connectivity makes mobile devices targets for a wide range of security attacks. Malware can be deployed to steal private data stored on the mobile device such as text messages, call history, photos, emails and others. More serious attacks may occur where malware initiates communication rather than just access stored information. In this thesis, we design and implement a lightweight malware detection technique that appropriately suits resource-constrained mobile devices in terms of low storage and computational requirements. The proposed technique utilizes several user parameters (such as SMS and call activity) and system parameters (such as CPU and memory utilization) over a period of time to model the activity profile of the mobile user rather than storing and checking a large number of abnormal behaviors (such as signatures of possible attacks). These parameters are continuously adapted based on the user behavior. Any violation to the constructed user profile will issue an alert for a potential security threat. We implemented and tested the proposed technique on Android mobile devices with several possible attacks. Results demonstrated its capabilities in terms of high malware detection success rate in addition to low storage and computational requirements.

Keywords: Computer Science, Intrusion detection, IDS, Lightweight, Mobile, Smartphone, Android, Standalone, Security

Table of content

List of Figures

List of Equations

List of Tables

Chapter 1

Introduction

Mobile devices substantially evolved in the last decade to become mini personal computers that fit in your pocket. With technological advances and the support of internet access in all modern mobile devices, they almost replaced the use of computers. Similarly, their small size and extended battery life also substituted gaming consoles such as Game Boy and PSP offering same or better graphic card. With such computer related features as well as mobile connectivity, security issues become an imperative subject to tackle. Based on the above argument, this research paper tackles a new standalone intrusion detection system is introduced to detect malwares.

The biggest challenge in mobile devices is the enormous number of operating systems and different distributions for each and different specs makes security hard. A generic method should be used to overcome such system variety and complexity. Here comes the idea of intrusion detection system (IDS) with standalone functionality that is implemented on each client and works by monitoring user behavior and generates a size efficient profile about his general usage based on various criteria such as CPU, SMS, and CALL.

A light weight algorithm will be used average the accumulated data from the user every 30 minutes and then compute rate change along with other factors

offer better personalization every time slot. Hence, the algorithm runs every 30 minutes to check for anomalies. Also, a normalization function is run to ensure a smooth user behavior and better computation.

Chapter 2

Smartphones: Evolution & Threats

2.1 History of Mobile Devices

A common fact states that there is no single invention that led to cellular phones, as better stated “Cellular radio is not so much a new technology as a new idea for organizing existing technology on a larger scale”- Calhoun, 1988 [1].

In 1843, Michael Faraday, a skilled chemist, began a meticulous research on the ability to conduct electricity through space. His research was the main pillar in the telecommunication field. Four Decades Later, Dr.Cooper made the first cellular phone call, and 6 years afterwards came the naissance of the cellular phones industry. In fact, the whole concept of cellular phones really emerged between the 1940s and 1980s. In 1983, Washington D.C. was chosen as a test subject to demonstrate the usage of cellular phones [2]. Subsequently, cellular phones started gaining an enormous popularity, and reached 50 million users in 1997 and more than 500 million subscribers worldwide in 2000 with an average growth of 24% subscribers yearly up until 2008 [1] [3].

In 2010, Symbian and Java ME were main targets for hackers' attacks, leaving Android with 0.5% of total malwares. In 2011, Android malware rate increased to 46.7% with rapid decrease for Java ME and Symbian (41% and 11.5% respectively as shown in Figure 1) [4]. The change in the percentage of total malware for each Java ME, Symbian, and Android shows the change in the hackers' attitude towards the changes in the market demand for mobile devices. Hence, the decrease in malware for a certain device doesn't specifically mean a safer one; it may imply that the interest of the public decreased for certain mobile brands, thus hackers focus on what is getting popular.

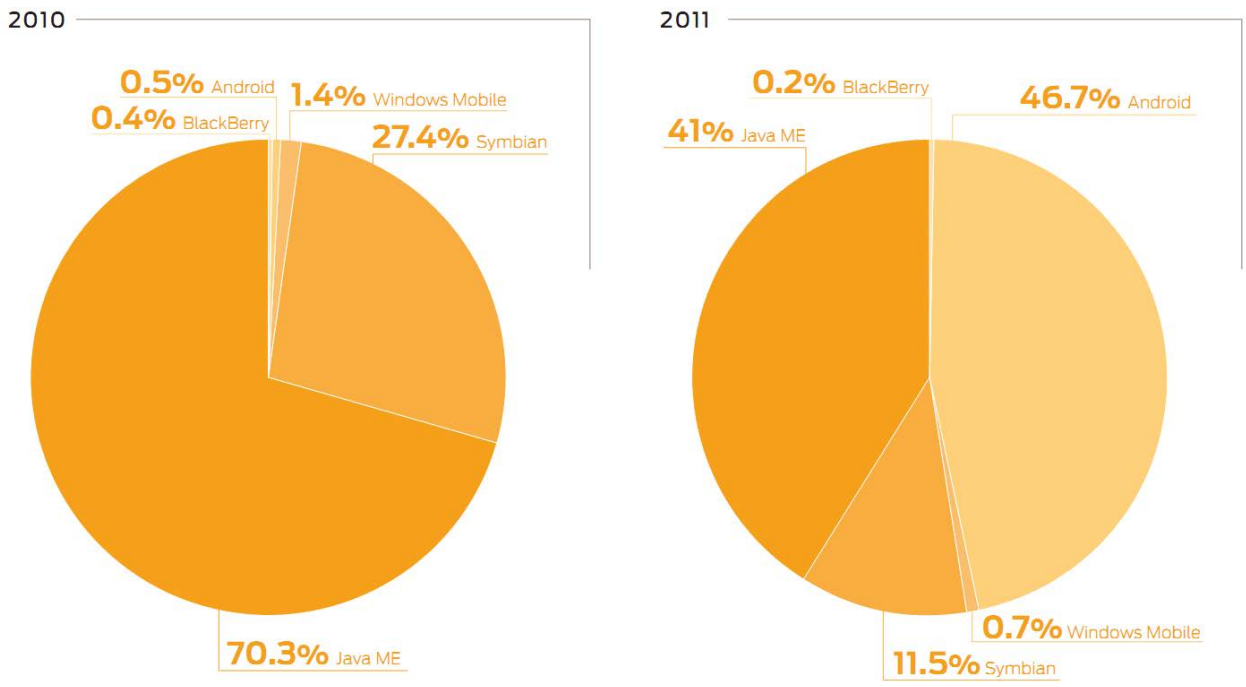

Figure 1 Unique mobile malware samples detected by operating system

Mobile phones underwent an incredible make-over that took them from a simple wireless phone to an indispensable device that closely resembles a micro-

computer. In fact, mobile phones are more personalized and hold more personal information than computers, thus, we can't help but wonder about the impact of mobile hacking and the amount of human endangerment it poses [5].

With the increasing number of mobile phone users placed a high demand on the network operators to offer more different type of services to fit the users' needs. Such needs include the availability of the user's personal, social, and business data, most of which require uninterrupted connectivity. All of which took mobile phones technology from cellular networks more commonly known as First Generation Cellular Network (1G), to all IP data networks, the 4G [6].

It is through this leap between generations, that mobile phones gained the features we now consider as postulates. The 1G allowed users to make voice calls. It is not until the 2G that the text messaging feature appeared and mobile phones came in smaller sizes than those of the 1G and thus became the standardized look of mobile phones. After this phase came the 3G cellular technology which offered faster data rates that made voice calls more vibrant and entertainment features possible, such as internet, gaming, video calls, mobile TV among other features. Subsequently came the Fourth Generation cellular technology (4G) that aims at providing high definition video for its mobile users with premium quality [7].

"With great power comes great responsibility", sates the old saying, and indeed, this fast evolution of cell phone technology has opened a backdoor for mobile phone security threats. For instance, first generation mobiles equipped with simple phone capabilities face only mere threat of eavesdropping, while advanced mobiles, such as the current generation, that are equipped with different network connectivity, (such as cellular, Wi-Fi, Bluetooth, Cameras, high storage and processing powers) are more susceptible to more serious attacks. Therefore the concept of mobile phone security became a grave challenge that needs to be highlighted and confronted [8].

2.2 Mobile User Behavior:

It is important to study the analytical statistics to monitor the major applications on mobile devices consumers often use. Table below shows similar trends between United States and Europe and bigger gap with Asian countries. It is important to note that 26% of applications are installed and used not more than once and application is left on the device [9].

	United States	Europe	Japan
Used connected media (browser, app or download)	46.70%	41.10%	76.80%
Used browser	36.40%	28.80%	55.40%
Used application	34.40%	28.00%	53.30%
Used messaging			
Sent text message	68.00%	82.70%	41.60%

Instant messaging	17.20%	14.20%	3.60%
Email	30.50%	22.20%	57.10%
Accessed entertainment/social media			
Took photos	52.40%	57.50%	62.90%
Social networking or blog	24.70%	18.00%	19.30%
Played games	23.20%	25.30%	16.30%
Recorded video	20.20%	26.10%	15.80%
Listened to music	15.70%	25.00%	12.90%
Watched TV and/or video	5.60%	5.70%	22.80%
Accessed financial services			
Bank accounts	11.40%	8.00%	7.00%
Financial news or stock quotes	10.20%	8.00%	16.50%
Accessed news, sports, weather, search, retail, travel, reference			
News and information	39.50%	32.20%	57.60%
Weather reports	25.20%	16.40%	34.70%
Search	21.40%	14.90%	31.50%
Maps	17.80%	13.00%	17.10%
Sports news	15.80%	12.00%	18.20%
Restaurant info	10.00%	6.50%	9.70%
Traffic reports	8.40%	7.40%	14.00%
Classifieds	7.30%	4.80%	3.60%
Retail site	6.50%	5.20%	8.50%
Travel service	4.40%	4.60%	2.90%

Table 1 Mobile behavior in United States, EU5 (UK, Germany, France, Spain and Italy) and Japan – October, November, December 2010 Percent of total mobile audience (Age 13+) [10]

2.3 Threats:

Throughout the years mobile phones have witnessed various security threats, first of which was Cabir that originated from the Philippines and developed in 2004. Cabir is the first worm that spread among mobile phones using Bluetooth; it was designed to work on smart phones that operate under a Symbian OS and was first detected by Kaspersky lab in Russia. The worm was not considered dangerous since no harmful code was attached to it, but it aims to drain the mobile battery [11] [12].

In 2005, “Paris Hilton Mobile Hack” spread among all the headlines when her personal mobile phone “Sidekick” a T-Mobile series was hacked by a bunch of teenage kids who ended up posting all of her personal contacts and pictures online [13].

In 2010, up to 4.6 million android users downloaded a malicious wallpaper application known as “Jackeey Wallpaper” that gathered personal information from infected mobile devices and emailed them to a website hosted in China. The Jackeey Wallpaper threat is a simple example of how dangerous mobile applications can be [14].

There are many other stories that started to raise the same worries; *Reuters* reported that in 2010 the number of threats aimed at mobile phones rose 46% from 2009 [15]. *SMobile Systems* confirmed that more than 400 mobile malware threats have already been detected and this number was predicted to exceed 1000 by the end of year 2007 [8], and mobile spywares were on the rise. Fredrik Lindstorm, a senior security consultant at Computing System Innovations said "When you consider that there are more cell phones than computers in today's environment, you stop to think about the impact a cell phone virus could have." [11]

In fact with the market buzz of smart phones and the increase in sophistication, security becomes a major topic for mobile phones. It takes only one look at the increase of sales in smart phones that jumped from 49% in the first quarter of 2010 to a mind blowing 96% in the 3rd quarter of 2010 for one to realize the seriousness of this matter [16].

A recent study conducted by Juniper Networks in 2011 showed a new trend in hacker's attitudes regarding the profile of their malware targets. Also, the study highlighted that hackers who are more sophisticated and equip in this field targeted businesses, governments among other users while aiming for financial gain by getting their hands on sensitive information that may prove profitable. It concluded that such acts put sensitive information under threat of ongoing attacks performed by such hackers. [4]

Chapter 3

Literature Review

3.1 Intrusion Detection Techniques for Mobile Wireless Networks

Y Zhang et al. (2003) highlight the importance of establishing secure mechanism for mobile phone users and the importance of IDS in securing mobile ad-hoc network. The paper [17] proceeds by identifying the vulnerabilities that exist in current IDS techniques and suggests a new IDS architecture for securing mobile ad-hoc networks.

The authors, argue that the existing IDS are tailored for wired networks and don't suit ad-hoc networks. The main differences are: the fact that the uniform infrastructure that ad-hoc network adopts, second, unlike wired networks, there is no single point, such as switches or gateway from which all sufficient data can be collected and tested against anomalies. Third, the decision of intrusion detection shouldn't be based on a single node in the network acting as a server, because that node could be compromised to send false information.

Consequently, this paper suggests a new intrusion detection technique that respects the nature of ad-hoc networks. The intrusion detection and response system are both distributed and cooperative. Every node in the network has an

IDS agent, thus every node is responsible of detecting intrusion locally, while neighboring nodes cooperate in detecting intrusion on the network level. As a result, the intrusion detection system is composed of a local detection engine that examines the local data collected through the data collection module and detects local anomalies. This data constitutes system and user activities as well as communication activities sent by this node and received by other nodes within its radio range. If the local detection engine detects an anomaly with a high confidence rate, it can initiate a response message that propagates to the entire network indicating that the network is under attack. On the other hand, if the local detection engine detects anomaly with a low confidence, it can check with its neighbors. The cooperative engine employs an algorithm that calculates the confidence factor using local data as well as data collected from its neighbors, where the opinion of close neighbors is given a higher weight than farther neighbors. For example, the overall scheme works as follows: A node sends an “intrusion state request” to its neighbors, then, each of these nodes propagates a response about the state of intrusion to its immediate neighbors. Each node can then determine if there is an intrusion based on the majority of responses it receives, and can initiate a response that indicates that the network in under attack. This scheme ensures that the decision of an intrusion is not solely based on one node alone. A compromised node wouldn’t admit that an intrusion occurred fear of getting expelled out of the network. Therefore the rest of the “legitimate” nodes will able to collectively detect this anomaly. Finally, upon detection of an anomaly, the system can either identify the compromised node and expel it from the network, or reinitiate the authentication phase, thus

the compromised node won't be able to authenticate itself hence, and the network is reformed without it.

In addition to cooperative intrusion detection system, an extra feature is embedded in order to lower the false positive ratio, Multi-layer intrusion detection. A node can check at each layer if there is a sign of intrusion because some layers can detect intrusion faster than others depending on data availability. For example, the application layer can detect a large amount of incoming connections which is a sign of a denial of service attack, much faster than the lower layers that relies on the amount of network traffic which takes some time to recognize. Thus intrusion detection system applies to the network as a whole: on all nodes and at all layers of a node.

The anomaly detection is based on classification mining technique, in particular RIPPER the decision tree equivalent classifier and SVM-Light the support vector machine classifier. The data collected by the data collection module labeled "trace data" is thus passed to the classifier which predicts whether there is anomaly based on the initial training set. For example, a routing table is modified only when a node moves, thus the movement of the node and the change in the table are the only indicators a node can trust. These factors can thus be used to test whether the routing information of a node is legitimate or it is a case of misrouting attack. We can capture a node's movement using a GPS

and the changes in the routing table are measured by the percentage of changed routes (PCR), the percentage of changes in the sum of hops of all the routes (PCH), and the percentage of newly added routes. Once the network is trained using many scenarios the classifier can then be ready to be tested at any node.

3.2 Hybrid Multi Agent-Neural Network Intrusion Detection with Mobile Visualization

In [18] Hero et Al. propose a new architecture for Intrusion detection systems entitled “The Mobile Visualization Connectionist Agent-Based IDS” MOVCAB-IDS, this approach extends the agent based IDS to a hybrid multi-agent IDS where each agent is assigned a different role contributing to the overall mechanism of intrusion detection.

The novel architecture is composed of 5 different agents: Sniffer, preprocessor, analyzer, configuration Manager, coordinator and visualizer. The Sniffer is a reactive agent responsible of collecting the data that traverse the network. This flow of data is decomposed into several segments and passed on to the preprocessor. The latter preprocess the data and convert it to the adequate format for subsequent analysis. Next, the preprocessed data is forwarded for analysis. The analyzer is a CBR-MDI deliberative agent; The CBR-MDI model embeds the artificial neural network technique and is mostly employed when heuristic knowledge is not available. [Cite Reference 14 from paper] The analyzer technique returns a solution assumption by comparing the current case

with previous similar case, and returns the most similar one found. This comparison is based on specific predefined attributes such as segment length, network size, and number of protocols. The configuration manager is used to determine these attributes. The analyzer known as CBR (case based reasoning) is thus divided into four stages: First, The retrieval phase that corresponds to the retrieval of the most similar case to the one being currently analyzed; The Second phase is the reuse phase, where the distance (measured using Euclidean distance) between the two cases is measured and various training sets are suggested based on the value of the distance, meaning, more training set for a larger distance value and contrarily. Thirdly, the revision stage where the user gets to visualize the response generated for each of the training sets from which the user gets to choose the best one. Once the user made his choice, we enter the retention phase where the dataset and its corresponding response are stored to be employed in the reuse stage during future analysis. Finally, the coordinator is in charge of circulating the results of the analysis among the different analyzer agents to improve the efficiency of the overall mechanism.

3.3 Trust Modeling and Evaluation in Ad Hoc Networks

Authors present a light weight approach to compute trust ratio for nodes without using an external server [19]. When any node wants to communicate with another it first calculates the paths it should route the data through. We will end up with many distinct routes, choosing the best path depends on the uptime of

that link. As such each node calculates the trust for all its neighbors depending on how rapid it routes the data or not.

The trust metric is measured between 0 and 1; where 0 is the lowest trust ratio possible and 1 is the highest possible trust ratio. All nodes start with 0.5 and when a node fails to process the data, the receiving node will lower the trust ratio of the other node.

Recommendation based trust is when a node receive trust ratio information from different neighboring nodes about a certain node. In such case the receiving node will also prioritize the reliability of such information by also considering the trust ratio for the recommender node.

The idea of calculating trust ratios is very lightweight and can be applied into client side for mobile security, though in typical systems, deployment of such security system is not practical since it relies on other nodes cooperation. If a node was under DoS attack its trust ratio will decrease and eventually be banned from the whole network. For such node to join the network again will be impossible unless ban table is flushed, such details are not mentioned. The idea of malware nodes is limited to the idea of node being cooperative, the authors didn't take into account malware that are being fully cooperative yet giving false recommendations about their neighbors; several of these nodes may totally disable the network to be routed to certain paths where eavesdropping is possible. To solve such crucial limitation, penalties are a must, each node should evaluate the aftermath for each decision taken. For example if node *A* recommended node *B* to *C*, but late node *B* turned out to be malicious then trust

of node *A* for *C* will decrease to penalize it for its bad recommendation. The only extra consideration in this idea is the huge log file that should keep track of all the decisions being made. This same log file can be used to analyze if certain node always favors and declines same nodes. Such debate can be solved when data mining techniques are done on such logs to analyze the behavior to conclude how to best handle big log files.

3.4 A Trust Model Based Routing Protocol for Secure Ad Hoc Networks

The authors present a trust model for MANETS in a network by selecting the best route for data path to avoid any downtime or delay in the system [20]. The protocol was designed based on Ad hoc On-demand Distance Vector (AODV) [21]. Its core features are:

(1) The trust relationship defines the trust routing mechanism for each node
(2) Any malicious behavior done by a node will be detected and forbidden access to the whole network
(3) Verification process using certificates improves the overall performance of the system at every routing step. The idea of the trust model can also be applied into other routing protocols of MANETs.

Like most trust models, the trust ratio or opinion defines the relation between two nodes donated by $w\frac{A}{B}$ where it resembles opinion that node *A* have for node

B. The opinion has three elements: belief (b), disbelief (d), and uncertainty (u); where it satisfies the relation below:

$$b\frac{A}{B} + d\frac{A}{B} + u\frac{A}{B} = 1$$

We calculate trustworthiness using variables *p* and *n* for positive and negative evidence respectively. Therefore, we can express $w\frac{A}{B}$ by *p* and n according to:

$$b\frac{A}{B} = \frac{p}{p+n+2}$$

$$d\frac{A}{B} = \frac{n}{p+n+2} \quad \text{, where } u\frac{A}{B} \neq 0$$

$$u\frac{A}{B} = \frac{2}{p+n+2}$$

The criteria for judging trustworthiness is described in the table below:

Belief	**disbelief**	**Uncertainty**	**Actions**
		> 0.5	Request and verify digital signature
	> 0.5		Distrust a node for an expire time
> 0.5			Trust a node and continue routing
≤ 0.5	≤ 0.5	≤ 0.5	Request and verify digital signature

Table 2: Trustworthiness ratio

The beauty of this approach lies in its light weight instant response to any new node and definition of node interaction is almost immediate. In our approach we learn from such light weight design for better deployment, which is a major issue in almost every intrusion detection approach. For example a universal way to connect devices is Bluetooth, now devices rely on network based connection like

RIM provides (black berry service) and iPhone. In the past, Nokia dominated the market with its Symbian based operating system that supports java. This provided a great deal of software deployment provided by third party application developers. In contrast, we have several major companies and others are joining the market fast, each with its own programming language and API different from the other, which gives compatibility issues at the network layer. To solve this problem, standalone intrusion detection is advised.

Chapter 4

Intrusion Detection Systems

The efficiency of basic security mechanisms has been proved efficient in protecting the physical connection, but they fail to provide the same security standard in wireless network, this is due to the fact that it is rather more difficult to attack wired network, since it requires both gaining a physical access to the network wires itself and the ability to bypass several network firewalls and gateways. However, wireless networks can be attacked from many direction and at any node and is subtle to both active and passive attacks, thus there are no clear lines of defense. In addition, the nodes are autonomous and dynamic which means that it is rather difficult to detect which node is compromised, and it is also possible that the compromised node could be carrying the secret key. Furthermore, ad-hoc algorithm relies on the cooperation of all the nodes to make the decision, which can be problematic because first it lacks a centralized authority which puts it a risk of untrusted peers attack.

Hence, the need to employ a second wall of protection for wireless network: intrusion detection system. Intrusion detection systems not only defend against attacks, but it also prevents them from happening.

IDS are usually classified based on the data that is being captured and audited for possible intrusion; these categories are better known as: “Network-Based IDS” or “Host-Based IDS” in general these two categories are known to be implemented simultaneously for a more efficient use of IDS. Another criterion by which IDS are divided is the type of detection they offer; the three most widely known are the following:

1. Anomaly Based Detection: In the anomaly based detection, the normal behavior of the user is compared with the captured data of the occurring events. If the captured data was found to be deviated from the normal behavior; the system admin is alerted. This approach can be prone to numerous false positive alerts [22].

2. Misuse Based Detection: In the misuse based detection, a signature of the known attacks is kept and used to compare with captured data. The downside of this approach is that it can only detect attacks whose signatures are known [23].

3. Specification Based Detection: In the specification based detection, a set of indicators is used to define the correct behavior and the violation of these indicators gives away potential security breaches [24].

Many intrusion detection systems already emerged each of which adopts either the misuse detection or anomaly detection technique. Mobile ad hoc network (MANETS) does not rely on the use of switches, routers, or gateways so it is open for malicious attacks and network injections that appear authentic. Moreover, one cannot clearly identify the normal from abnormal activities. In such networking design, the routing of information can be easily compromised by injecting one node into the network that performs outdated information and leads to denial-of-service. To overcome this limitation, intrusion detection techniques are been proposed by either flat or multi-layering depending on the application and its flexibility. IDS is classified into the several categories [25] described below:

4.1 Stand-alone Intrusion Detection Systems

Each node runs totally independent of the other and each node has a copy of the whole IDS software installed on the device that does its own computations and analysis independent of external factors that might be imposed by neighboring nodes. Each node is totally not aware of its neighboring nodes, nor does it exchange any kind of information or process routing of data from node to another. As such this architecture is not effective in many cases where node cooperation is deemed or when the data collected locally at the client side are not enough to reach an action decision. Such architecture is most effective on flat networks where node cooperation is not possible. Therefore stand-alone IDS have not been chosen in almost all MANETs network design structure.

4.2 Distributed and Cooperative Intrusion Detection Systems

Main idea of MANET is not to have a centralized node that might act as a server. Each node makes its own decision but based on the collaboration of trust from neighboring nodes. Each node has its own agent that is mimicked, thus the detection and data collection is node based as well as initiating the response. This model [26] has the basic criteria for node cooperation by exchanging trust information to help in the decision making. Same as stand-alone architecture, distributed and cooperative intrusion detection system is more suitable for flat network rather multilayered infrastructure.

4.3 Hierarchical Intrusion Detection Systems

Hierarchical IDS a cluster based network that is multilayered; it is based on distributive and cooperative IDS architectures. Each cluster have a cluster head that have more functionality than the rest of the nodes inside the same cluster, it acts as a medium of contact between clusters and more trusted. Cluster heads can act as a router or access point and a way to communicate between two clusters. Such logic can highly increase security by avoiding all kinds of contacts between nodes from different clusters, although in hierarchical IDS there is no central server; cluster heads can act as one as they are assigned either automatically via a dynamic algorithm to decide which is the most suitable

routing node based on predefined criteria. Each node holds an agent that is identical in all nodes, every node perform the same tasks and its decisions are done locally, the same applies to cluster heads with the addition of monitoring network packets and setting up the proper response when an anomaly is detected in the network.

4.4 Mobile Agent for Intrusion Detection Systems

On large scale applications and highly intensive computation; deploying the same agent on each node is resource intensive and not practical in real life. To provide efficiency, mobile agents are introduced by assigning each node a mobile agent specific to perform only one task, the information is then routed between the nodes when any node requires a certain output that is necessary for its mobile agent computation. For example node A requires sending data to node B sorted; sending procedure requires encryption of data as well, node A is responsible for encrypting the data, but asks node C to sort them first. This collaboration will make a truly light weight IDS, but its draw back lies in the importance of each node and taking into account that every node is trusted, thus a highly secure system should be implemented to avoid infected nodes to be part of this process. A better version of this architecture is by assigning the same task to multiple nodes to allow error tolerance in case a node crashed [27].

Chapter 5

System Model

5.1 Mobile platforms

One challenge in developing applications is in choosing the right platform to perform the tests. Many operating systems for mobiles are been depreciated and replaced with a more powerful solution enabling many services that haven't been available in the past.

Almost every operating system is targeted for a certain audience and the applications been developed on those OS's uses an API provided by this platform. If an API is not available in the software development kit, the programmer should develop his own. Some operating systems intentionally limit the usage to keep its source code and functionality confidential. We summarized the below operating systems as below:

5.1.1 Android

Project was purchased in 2005 by Google and its initial release was in October 2008. Android Linux based and an open source project under license GPL v2. GPL license is the same license the popular apache server use which states that anyone can take, modify, and reuse the code for his own usage as long as copyright information is provided and not altered. Due to this license, Android

became the most popular operating system for most users and number one target for device companies. It has a very strong community online providing official and online support via forums. Many tests have been done and most of recent research studies use android as a testing and development environment. Within few minutes SDK can be downloaded for free and ready to emulate the behavior of android mobiles on the computer. Many disadvantages arise with all open sources projects and Android is no exception. With such popularity many official versions from Google are published periodically that offers forward version support of applications, but on the other hand with every new version many several editions are released from each handset that uses slightly different APIs and sometimes same APIs but different function names. For any application to work, it should be optimized on each and every handset possible to ensure its working which might be a major problem. Never the less Android proved its self-stable and feasible way to get accurate experimental results in research work.

5.1.2 iPhone

One of the most popular handsets, often over rated and overpriced like most Apple products. Manly focuses for end users and hides all complexities from the user to provide smooth user experience. iPhone is well documented and have huge online support. Gaining a license to gain access SDK tools is not free and will often go under legal issues with unclear terms of service. iPhone strictly prohibit any modifications done in software layers that directly access the

hardware, thus development for scientific purposes is not recommended on such handset and preferably to avoid it as well. In October 2011 Apple Inc. released its enhanced iPhone 4S with superior hardware according to latest benchmarks that surpass most devices for at least the coming year [28].

5.1.3 Maemo

It is based on Debian Linux Distribution. It was first released under name OS2005 and then the upgraded version OS2008. With every new mobile release a new updated version accompanied the device. Maemo targets special kind of audience known as “power users”. It was often said that mobile is a portable minicomputer with the full functionalities of a normal computer with mobile support. Most of the code is open source and the rest can be easily revered engineered. With a very similar environment if not the same as Debian it is a perfect target for all developers. The exact same applications in Linux can be executed on Maemo without edits. Maemo have the ability to emulate windows xp, android and any Linux based operating system. The SDK can be easily understood with similar parameters as Linux. It supports all computer languages that Linux supports with the addition of new QT language that is cross platform language. The crucial disadvantage is the lack of official support and the announcement of a new predecessor just few weeks after launching Nokia n900 which killed its market share and any further updates. It’s been over 3 years since the announced predecessor named Meego and in June 2011 was the first release for such mobile called Nokia N9. Unconfirmed rumors after a month after that Meego project will be dropped. Such uncertainty leaves this operating

system in the dark and better to avoid for development purposes for the near future until a solid plan for it is published.

5.1.4 Meego

The predecessor of Maemo. It is still under development by Nokia with the collaboration of Intel. It uses the same Linux Distribution but with a new more friendly interface. The development focused on a platform that can be installed in most devices as well as netbooks. Not much research is done on Meego even though it was praised after its first release Nokia n9 in the third quarter of 2011. Despite its high sales in Europe it was discontinued by Nokia cooperation and headed its focus on Windows Mobile 7.

5.1.5 Symbian

Symbian project started under the name Psion in 1980. It was later renamed into Symbian after the collaboration of Nokia, NTT DoCoMo, Sony Ericsson and Symbian Ltd. Various improvements are done on the platform throughout the years. It reached a point where Nokia dominated the mobile market with Symbian platforms. Several versions and editions of Nokia platforms were introduced as in S40 and S60. S40 was a classical mobile and mostly used for those who seek the old fashioned mobile. S60 improved into v2 v3 and v5 allowing 3G and Wi-Fi technology into its platform. Despite this improvement, as of 2007; the release of iPhone a huge decline in sales started and the market started to favor the fancy looking iPhone regardless of its inflated price. Few years later with constant decline in sales, Nokia decided in 2011 to close its

Symbian project completely and use the new WP7 on all its new releases. The rage after the news about recent mobile releases caused Nokia CEO to assure that Symbian support will remain for a long time but since this then no update for any version for the Symbian operating system. Google in the other hand opened the door for all Symbian developers to join Google team.

5.1.6 Windows Phone 7

Considered to be the predecessor of Windows CE, released in November 2010 and extensive development ever since. Microsoft offers a new interface with support for third party services. WP7 was named after the successful windows 7. The initial release of WP7 was very buggy and caused fatal errors which caused developers to wait for stable release. Till this date WP7 online support is very limited and the only supports C# as a programming language. Although this operating system offers a great deal of flexibility, yet it doesn't have enough testing for production use. The source code is not available as always known from Microsoft products; consequently modifying the kernel for development purposes is not possible. Online support through forums is also not comprehensive. WP7 have a promising future and it is said to acquire double the sales of that of iPhone. According to IDC [29]; a leading and trusted source for mobile benchmarking, forecasted that in year 2015 WP7 will acquire 20.9% of market share whereas iPhone sales will slightly decrease to 15.3%

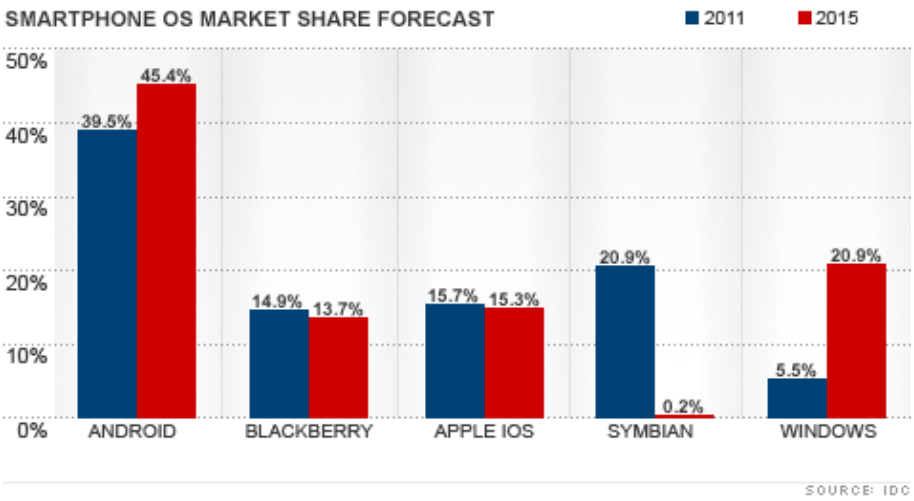

Figure 2: Windows Phone to eclipse iPhone sales by 2015 - forecast

5.2 System Implementation

In our research we divided our work into three phases: Data collection, Simulation and testing. As with every IDS approach we should first learn the normal behavior of the user to detect a pattern of what we call valid signatures. After that a generic virus will run with various attributes mimicking the behavior of user with the possibility of having very similar pattern that of a normal user and attacker. Our aim is to identify this anomaly.

5.2.1 Mobile setup

Operating system: Android is our chosen platform which is a perfect environment for low level testing in almost all research work. Its open source environment allows us to edit every aspect of it offering a great deal of flexibility. Each platform has many editions and versions; in Linux they call them distributions (*distro*). Before making any research we based our development on

the latest version of Android, but Android offers a forward compatibility which means that every current application that works on version *x* can work perfectly on version *x+1*. As a result we diverted our focus to the most popular version; according to Google Android official website the most used version is 2.2 [30].

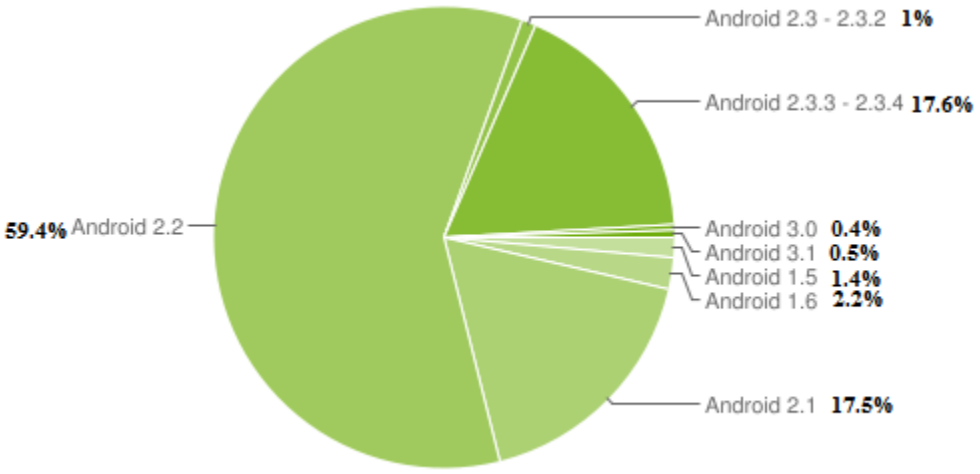

Figure 3 Pie chart based on the number of Android devices that have accessed Google Play within a 14-day period ending on the data collection date on July 5, 2011 [30]

We want to expand our application user visibility for end users, so to date Android 2.2 is our choice, it is for sure that with time the above numbers are prune to change with every new version that stops the development of its predecessor, but keeping in mind the forward compatibility of applications, version 2.2 is the most recommended for most developers.

5.2.2 Mobile choice

Application deployment should be tested on various mobiles to ensure its compatibility. It is important to note that due to the open source nature of Android operating system, each company is developing its own API. Many times it also renames many common java classes which causes serious problems and lose of time. To solve this issue we limited our research on major mobiles only; Nexus and HTC. A subclass is made for each mobile that accounts for such special cases, this solved the problem and it is the only way any application will be able to support multiple mobiles.

Mobile hardware should have the below minimum criteria:

- decent ram: above 256MB of ram
- 500 Mhz Scorpion processor
- Support of external SD card
- Build in QWERTY keyboard
- Touch screen can be capacitive or resistive

Though, the above are only guidelines that assist us to choose the mobile to test upon it and testing can be done on different criteria. For example, the idea of choosing mobiles with keyboard is to be able to identify inputs done by keyboard and not only using the touch screen. The support of external SD card which an included option in almost every mobile, will allow us to export the results to external memory card for further offline analysis.

Based on the above the testing was done on the below mobiles:

- HTC Desire Z
- Google Nexus

5.2.3 Software

Official Android SDK manager Build 11 is used, the following packages are installed:

Android SDK Tools, revision 11

Android SDK Platform-tools, revision 4

Documentation for Android SDK, AP112, revision 1

SDK Platform Android 2.3.3, API 10, revision 1

SDK Platform Android 2.3.1, API 9, revision 2 (Obsolete)

SDK Platform Android 2.2, API 8, revision 2

SDK Platform Android 2.1 – update1, API 7, revision 2

Android Compatibility package, revision 2

Google Admob Ads Sdk package, revision 2

Google Market Billing package, revision 1

Google Market Licensing package, revision 1

IDE used is Eclipse SDK version 3.6.2 and Google ADV plugin specific for Eclipse is used to connect IDE with android SDK. The ADV plugin provided by android ease up the process of integration and provides us with memory tools to check for any possible software memory leakage. Other tools include File manager, Emulation Control, Threads, Heap, Allocation tracker and many more.

Although Android virtual emulation works, it is very slow and behavior changes dramatically from that of a real device, so we connected our Android enabled device using an usb plug to our computer and upon each test we upload the new software to the mobile. Special synchronization software should be installed for each mobile so that Eclipse SDK can detect it, such software can be downloaded directly from the mobile official website software update page.

The process might sound time consuming but during our long trial and error testing, it saved us countless hours and over 5 minutes wait that we normally used to wait on the machine to start running the virtual simulator. In addition, most developers test their applications pretty much the same approach.

5.3 Implementation Phases

To speed up the process of development and testing we split up the whole process into three phases: Data collection, testing simulation known as phase 1, and application deployment known as phase 2. Figure 4 shows different stages of the project that starts with collecting multiple log files from different users and processed using the simulator. When the algorithm is approved we go from complete logs to per entry log that will be processed. We can divide different stages as follows:

Phase 1: Algorithm testing and modification

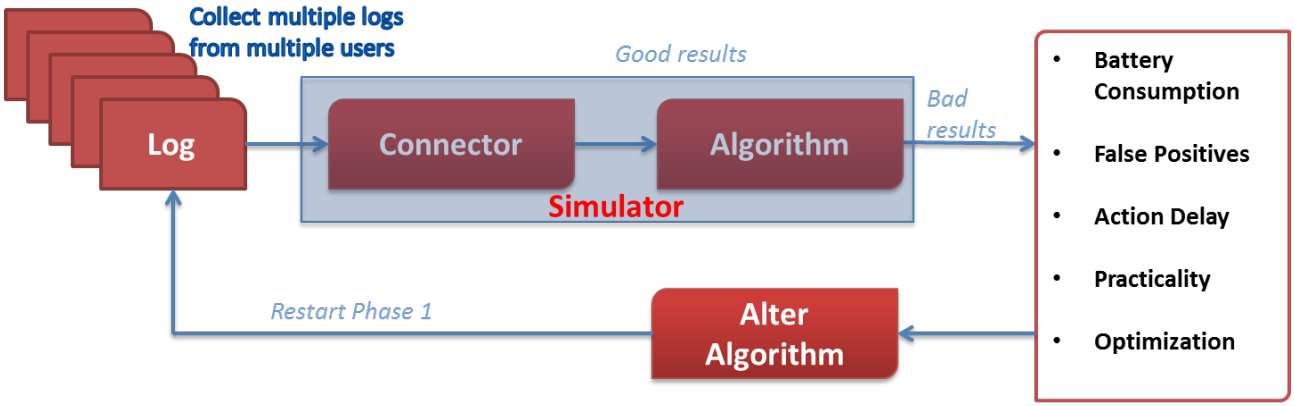

Figure 4 Development process for phase 1

1 Log collection from different users that will serve as multiple test cases.

2 Log files are fed to the simulator[1] one at a time.

3 The algorithm will produce output results. Those outputs will be manually analyzed and check if the anomaly detected where correct and in case of intentional virus injection is should also detect them.

3.1 In case of bad results the algorithm is altered and phase 1 is repeated. Bad results may be for various reasons, most commonly false positives and false negatives.

[1] Simulator is basically made from the algorithm and a connector that will interpret the large log files and acts as if those inputs were real time. The connector is simply a switch that will make the system run logs or real time mode.

Phase 2: Algorithm in real time

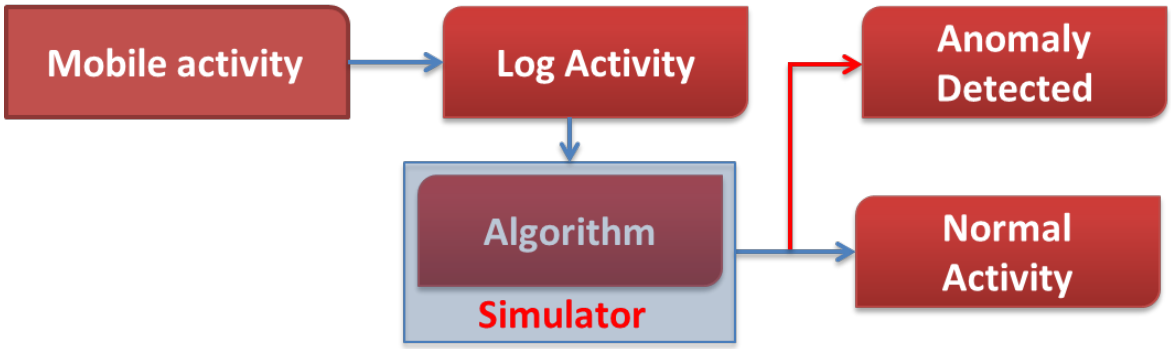

Figure 5 Development process for phase 2

1 Logs are now done per entry and saved into a temporary database for 30 minutes.

2 At the end of each 30 minutes the algorithm is executed. The algorithm is the same one in phase 1, but this time without the connector that was responsible for creating the real time behavior for the algorithm to understand.

2.1 In case of no anomaly the device will continue working normally without any interruption.

2.2 In case of anomaly detected the device will log the anomaly into a separate file and issue a separate task. For development purposes the task should be “null”, for users the task can be either an external sms notification to a predefined number or device lock with a password.

5.4 List of Indicators:

A mobile application is created to create a log file that monitors several attributes from the user. This data collection phase is often known as the learning phase or the monitoring phase; it collects various indicators from the mobile that are based on user inputs and system monitoring. Those data are saved in one table called log. Note that this table is for the data collection phase only.

Indicators are the following:

Indicators	Value	Extra1	Extra2
Bluetooth	boolean	N/A	N/A
WIFI	boolean	N/A	N/A
SDCARD	%used	Usage	Total
DATA_STORAGE	%used	Usage	Total
RAM	%used	Usage	Total
CPU	Unix load	Avg in 5 mins	Threads in queue
IDLE	Duration	N/A	N/A
GPS	Boolean	X	Y
SMS_IN*	Length	number	Boolean
SMS_OUT*	Length	Number	Boolean
CALL_IN*	Duration	Number	Boolean
CALL_OUT*	Duration	Number	Boolean

Table 3 List of indicators and fields inside the database. *extra1*, and *extra2* are values that might be used when *value* is not enough.

5.4.1 Bluetooth:

Daemon waits and not evoked explicitly until Bluetooth is turned on or off. It logs the following:

- Status: On or off
- Date: Unix timestamp in milliseconds

5.4.2 WIFI:

Daemon waits and not evoked explicitly until WIFI is turned on or off. It logs the following:

- Status: On or off
- Date: Unix timestamp in milliseconds

5.4.3 SDCARD:

Also known as external storage. It is highly important to have an external card for our research, it will simplify the process rather than just having an additional criteria. All imports and exports for logs are done using SDCARD.

A daemon background process is executed periodically after a predefined time (default is 5 minutes). In our research, it is important that the user doesn't change his SDCARD else a huge change in data collection will occur and it might be considered a breach. In Android, there is no direct way to calculate physical memory inside SDCARD, the below equations is used to calculate total, free, and used size in bytes:

total = getBlockCount() * getBlockSize()

free = getAvailableBlocks() * getBlockSize()

used = total - free;

- getBlockCount: The total number of blocks on the file system.
- getBlockSize: The size, in bytes, of a block on the file system.
- getAvailableBlocks: The number of blocks that are free on the file system and available to applications.

The following variables are then logged:

- % Used: (*used*/*total*)*100
- Total usage in bytes: calculated via *used* equation
- Total size: calculated using *total* equation
- Date: Unix timestamp in milliseconds

The most important criterion is the %used. We don't need to compute the percentage during the simulation process. The battery consumption at this point is almost zero, so it is advised that all calculations produced by the equations above that might be used later on to be done and saved at this point.

5.4.4 DATA_STORAGE: Also known as internal physical memory or Phone memory. A process is executed periodically after a predefined time (default is 5 minutes) and reads internal physical memory. The total internal physical memory is always fixed. Also, in our research, the announced memory for the device is always different from the actual memory. Such difference is due to the limitation android devices have on the user level; only a specific part of the system can be accessed by the user and thus read. This area is being logged

by our Handset monitor software. Such limitation have absolutely no impact on the effectiveness of our software and its calculations, since our aim is to log and monitor the behavior of the device, so deviations from the actual data won't have any impact on the output results..

For the internal physical memory, android gave the user set of functions to retrieve such information, but a trace logger should be used.

The following variables are then logged:

- % Used: (*total usage*/*total size*)*100
- Total usage: total used user space in bytes
- Total size: total user space size in bytes
- Date: Unix timestamp in milliseconds

In most cases, internal storage is prune to sudden changes due to the option that allows users to move their applications to external memory. Thus the idea of combined calculation of internal and external physical memory might be used. Another approach is to give priority for each criterion.

5.4.5 RAM: Random Access Memory is buffer storage to speed up read and write operations. It is one of the most influential aspects that define the usage pattern of a device and even any system.

A process is executed periodically after a predefined time (default is 5 minutes) and reads memory usage variables.

There are two ways to fetch memory details (RAM) from android device.

1. Read and parse file /proc/meminfo
2. Use internal android functions

```
MemoryInfo mi = new MemoryInfo();
ActivityManager activityManager = (ActivityManager)
getSystemService(ACTIVITY_SERVICE);
activityManager.getMemoryInfo(mi);
```

- The above snippet code will initialize a constructor and then use getMemoryInfo method to retrieve RAM information.

We noticed slight change in information from the two methods, but any chosen method will work and no impact on the efficiency of the data, since we are studying user pattern and how it changes with time and “not” the actual collected data. Yet once a method is used we shouldn’t use a different one, therefore our aim was to find the fastest way to fetch such information; so parsing text file was the fastest method and gives more realistic real time statistics of the device.

The following variables are then logged:

- % Used: (*total usage*/*total size*)*100
- Total usage: memory in bytes
- Total size: total memory in bytes
- Date: Unix timestamp in milliseconds

It is important to note that in all android devices, applications do not exit, they reside in memory until the system is out of memory it starts killing old processes. In such cases the RAM can be a very important criterion in case of most regular mobile users who limit their usage on common mobile functions. As for power users[2] the memory will be almost full in most cases and thus, it might not help to detect any kind of anomalies. As such, monitoring CPU usage is crucial.

5.4.6 CPU: Most important criterion among all, it defines how much load the device is having, it can also define which game or application is being run depending on CPU load data. CPU load information can also give us averages in the past 5 or 10 minutes as well as number of sleeping threads and threads in queue. Such detailed information can characterize the device and draw a clear user behavior.

A process is executed periodically after a predefined time (default is 5 minutes) and reads CPU usage variables from a UNIX file called /proc/loadavg

Below is a sample of the output information:

6.59 5.59 5.46 4/341 2172

The following variables are then logged, others are discarded as they are not required:

- Instant cpu load: taken from 1st value (example 6.59)

[2] user with extensive knowledge of computers and can take advantage of advanced capabilities of his hardware and software

- Average load: takes the average of every 5 minutes, value taken from 2nd value (example 5.59)
- Threads in queue: taken from 4th variable second part (example 341)
- Date: Unix timestamp in milliseconds

Since we are fetching CPU information every 5 minutes (default value), average load gives better information about what happened during those 5 minutes.

Processes are divided to system and user processes. In most cases CPU load should be constant when mobile is idle and no interaction with user unless background applications are working in background and have not been killed. In case of system process; it has been optimized not to consume CPU during idle state. In case of user process, a cleanup job is run by the system to kill all abusive threads.

5.4.7 IDLE: Defines the duration where the device is not being used or no human interaction was made with the device. Such criterion helps to determine if mobile action was done while the user was away or not. Moreover, it defines the timetable in which user is active. For example during night user interaction varies that in day time.

Due to technical limitation in android devices it doesn't give any API for the developer to check for any user screen input. This is done to provide security against key loggers (both touch screens and keyboard). To overcome this limitation we defined IDLE time to be the duration in which screen turned off

and then on. By means, it starts a counter to measure the duration of on screen. This provides a very good estimation for most cases. In some cases where user doesn't define a screen time off the Handset monitor will not be able to detect. Though, such special custom settings are never used for practical mobile users.

Variables logged are as follows:

- Date: Unix timestamp in milliseconds taken when the screen is off
- Duration: Unix timestamp in milliseconds taken when the screen is on after the occurrence of date timestamp.

Based on above the off time is *date-duration*

5.4.8 GPS: A system for determining position on the Earth's surface by comparing radio signals from several satellites. When GPS positioning is completed the system will consist of 24 satellites equipped with radio transmitters and atomic clocks.

Depending on your geographic location, the GPS receiver samples data from up to six satellites, it then calculates the time taken for each satellite signal to reach the GPS receiver, and from the difference in time of reception, determines your location.

The idea of monitoring GPS was first for on and off status but then upgraded to also include longitude and latitude only if GPS fix was found. GPS fix means the

location was found, to minimize all kinds of computations *x* and *y* are saved only in this case and ignored if GPS is on and no GPS fix or GPS is off.

Variables logged are as follows:

- Status: Defines whether GPS is *on* or *off*
- X: Longitude (between -180 and 180)
- Y: Latitude (between 0 and 90)
- Date: Unix timestamp in milliseconds

In some cases GPS might receive totally inaccurate data, in case the following rules were not satisfied GPS location will be ignored:

- Maximum value of Latitude is 90 degrees (at poles)
- Minimum value of Latitude is 0 degree (at Equator)
- Maximum value of Longitude is 180 degrees (eastward from Green Witch)
- Minimum value of Longitude is -180 degree (westward from Green Witch)

GPS option can be used at any time to develop a location awareness system that defines points of interests for the user, either path and/or clustering.

5.4.9 SMS_IN & SMS_OUT: Daemon waits for signals generated by message application in android operating system. SMS activity provides clear image about user behavior, like most people that send messages, favorite

contacts, time of SMS activity, and if those numbers inside your contact list or not.

When a signal is detected the following are logged:

- length: text message length counted per character
- number: phone number or N/A if not detected
- Phonebook: Boolean that determines if *number* is in phone book or not.
- Date: Unix timestamp in milliseconds

We separated SMS_IN and SMS_OUT into two tables for simplicity and since each one of them is totally different from the other. In other words the number of SMS_IN never defines nor has a relation with SMS_OUT. So separation for both indicators is advised.

5.4.10 CALL_IN & CALL_OUT:

Daemon waits for signals generated by phone application in android operating system. Phone activity provides clear image about user behavior, like most people that calls, favorite contacts, time of Phone activity, and if those numbers inside your contact list or not.

Call duration is defined when the call is established and not when the call starts. This allows us to differentiate between established calls and missed calls.

When a signal is detected the following are logged:

- duration: call duration in milliseconds
- number: phone number or N/A if not detected

- Phonebook: Boolean that determines if *number* is in phone book or not.
- Date: Unix timestamp in milliseconds

We separated CALL_IN and CALL_OUT into two tables for simplicity and since each one of them is totally different from the other. In other words, the number of CALL _IN never defines nor has no relation with CALL _OUT. So separation for both criteria is advised.

5.5 Handset Monitor Application:

Also known as HSM, it is an application developed to monitor all the indicators described in the earlier section, it works in the background as a service and ensures that Android operating system don't kill it in case the device went idle or low on memory.

5.5.1 GUI:

Main screen: Contains all the logs captured in a table ordered by date and supports both portrait and landscape mode.

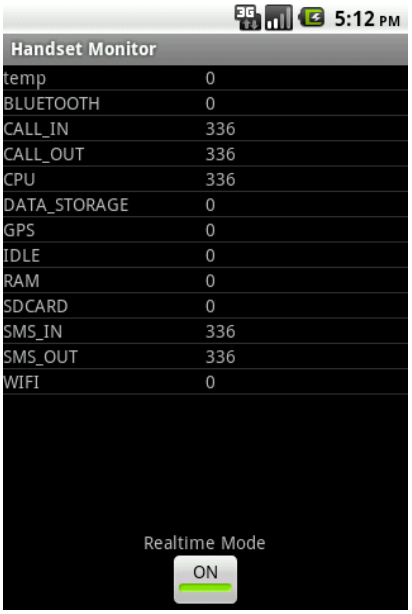

Figure 6 HSM main screen

Menu options:

- Start: Starts a daemon that will initiate handset monitor application along with its service.
- Stop: Kills service and stops the application from logging
- Export: Saves all the logfile into main directory of SDCARD with filename called *logs.db* in *sqlite* format
- Log: Saves all the logfile into main directory of SDCARD with filename called *logs.txt* in comma separated text format.

  The comma separated format can be read using any text editor, whereas the sqlite format should be opened using a third party application specific for it. For quick debugging purposes we use the comma separated.

  Live mode: Switches the whole application from logging mode into real simulation.

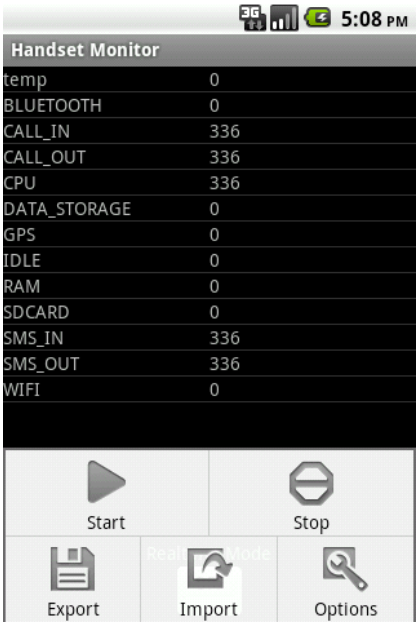

Figure 7 HSM menu settings

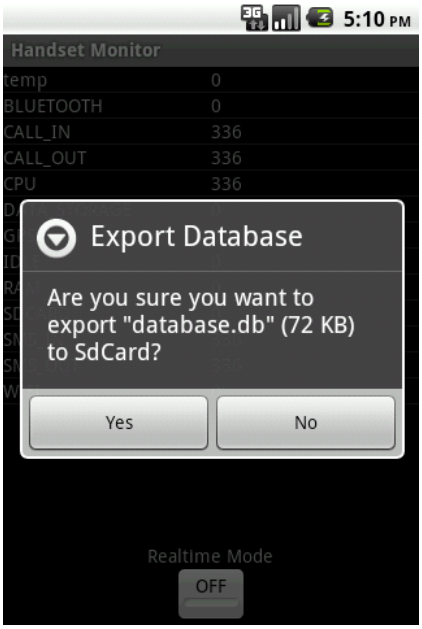

Figure 8 Exporting Database to SDcard

- Options:
    - Refresh Interval: Changes periodic check indicators (5, 15,30 minutes, 1,3,6,12 hours) Default is 5 minutes.
    - Log filter settings: Select which indicators to monitor and log. Available indicators: WIFI, Bluetooth, GPS, SMS, Calls, Idle time, CPU, RAM, SDCARD, and data storage.

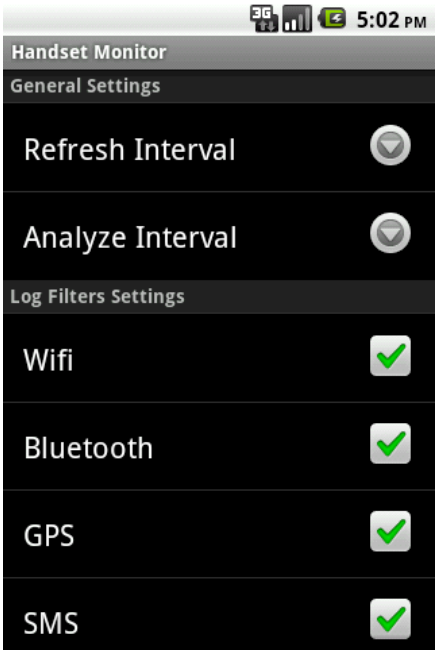

Figure 9 HSM Settings

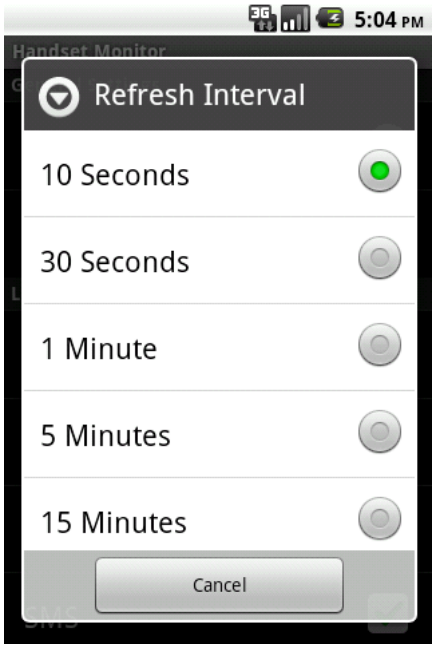

Figure 10 HSM Refresh Interval settings

Note that handset monitor is roughly limited to capture user actions and monitor mobile changes. In later stages Monitoring will be merged with simulation part to generate actions in real time.

5.5.2 Application usage:

After successful installation the program takes roughly 1.2MB of internal storage. Upon 5 months of testing the application memory usage is around 8.5MB and never showed any kind of system delay or delay in application response. Refer to battery consumption section for more details about battery usage before and after.

5.6 Implementation of a Genetic Malware:

A program created to simulate the behavior of a real malware. It is installed on the device and can be enabled and disabled using an easy to use interface. It contains a filter option that allows developers to choose which criteria to enable. Note that application name *Virus* is a convention of a generic malware and we will refer to all kinds of anomaly generated by applications as malware applications.

Malware applications usually infect all users with bug inside certain software or by a direct installation and permission by the user. With the new technology in operating system multilayered their OS and permission should be granted to access from one layer to another. For example to access hard disk permission should be granted by the user first. In android, such security behavior is also used, each application prior to installation shows a list of permission allowance to access certain features of the mobile; giving a clear idea about what will this program do. Table 4 shows a list of common malware and their threats [31].

Malware	Description
Snake	Information theft – while playing a snake game, in the background the application is quietly taking pictures and sending them off to a remote server
Tip calculator	Denial-of-service – launches a background service that in turn spawns hundreds of threads; thus effectively overloading

	the system
Malware injection	Denial-of-service – A virus on a PC secretly installs a virus on the Android which blocks outgoing phone calls
SD-card leakage (HTTP uploader)	Information theft – reads sensitive information stored on the SD- card and other locations and sends it to a remote server
Lunar Lander & SMS scheduler	Information theft – exploiting the Shared-User-ID and reading contacts and sending them via SMS

Table 4 Some popular malware on Android operating system

Even for a malware to get installed, the user should grant permission for the application. Same goes for our generic virus; it was granted full privilege over the mobile device and was set as a *device administrator* under the name *Virus*.

5.6.1 GUI:

Certain features were removed from the application such as: run on startup, it completely crashed the mobile during testing phase. Such crash is due to a possible bug in the virus application that runs an infinite loop. It is highly advised not to run such applications on startup and for development purposes it is better to run them manually.

Upon first launch, all features are disabled and setting is OFF. The user should first enable the virus and then individually enable the indicators he/she wants to activate. With one click on "*ON*" the virus will toggle to offline mode as shown in the figure below.

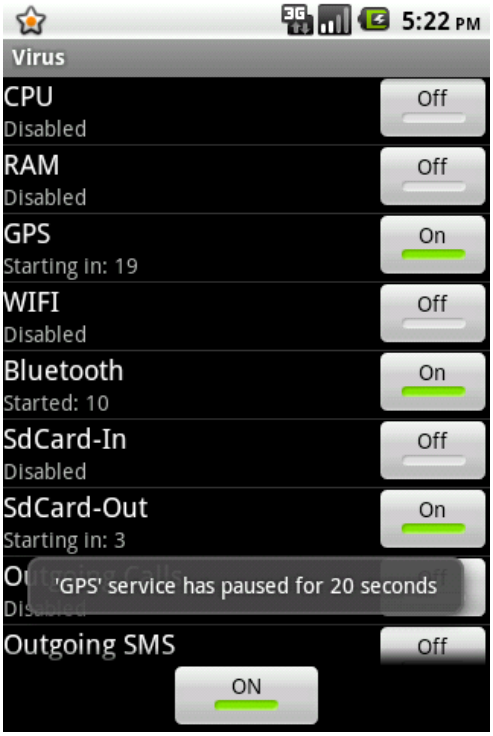

Figure 11 Virus main window

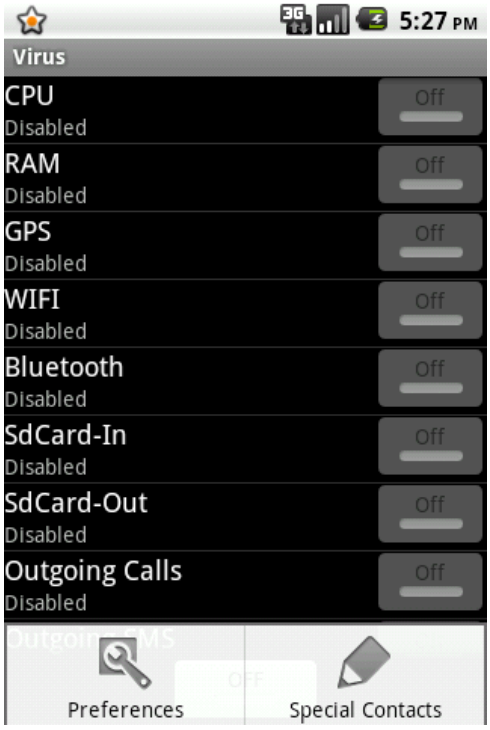

Figure 12 Virus menu settings

Menu contains the following:

- Preferences:
    - Frequency precision %: How precise the each filter should be. This helps to provide better randomized data. For example if default fake SMS is every 20 seconds and frequency precision is 50%. It means that the next SMS can occur from anything from 10 to 30 next seconds.
    - Duration precision %: How precise the duration for each filter should be. This helps to provide better randomized and flexible durations. For example if default fake call duration is 20 seconds

and frequency precision is 50%. It means that call can be anything from 10 to 30 seconds.

- Phonebook: checkbox to enable or disable random selection for a number from internal phonebook
- Special list: checkbox to enable or disable random selection for a number from a predefined contact list called "*special contacts*"

- Special Contacts:
  - Add: adds a new number and label to a special list different from phonebook contact list
  - Remove: removes number previously added and its label
  - Edit: edit number previously added and its label

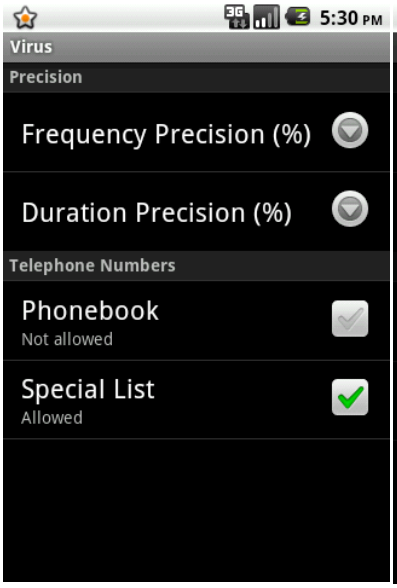

Figure 13 Virus Settings

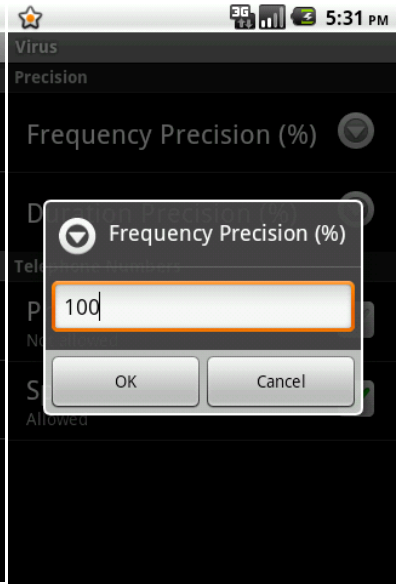

Figure 14 Virus frequency precision

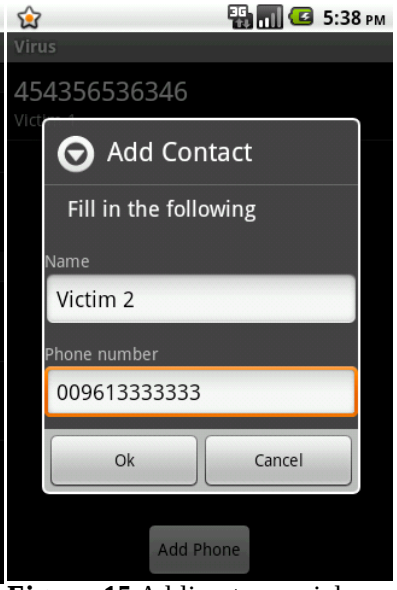

Figure 15 Adding to special contacts

5.6.2 Filters:

The main screen of the *Virus* application contains a list of indicators to be enabled and a button at the end to enable or disable all features.

To get accurate test results we added all indicators that are found in *Handset monitor* application. The indicators are as follows:

- CPU [stress cpu]
    - Frequency: After how many seconds this filter will run
    - Duration: Number of seconds cpu will get stressed
- RAM [fill ram]
    - Frequency: After how many seconds this filter will run
    - Duration: Number of seconds ram will keep filling
- GPS [set on and off]
    - Frequency: After how many seconds this filter will run
    - Duration: Number of seconds GPS will set to on before off
- WIFI [set on and off]
    - Frequency: After how many seconds this filter will run
    - Duration: Number of seconds Wi-Fi will set to on before off
- Bluetooth [set on and off]
    - Frequency: After how many seconds this filter will run
    - Duration: Number of seconds Bluetooth will set to on before off again.
- SDcard-in [fill sdcard inside folder *out*]

    - Frequency: After how many seconds this filter will run
    - Size: Defines how the size in KB for the file to fill when run
- SDcard-out [remove files from folder *out*]
    - Frequency: After how many seconds this filter will run
    - Number of files: number of files to remove to free up space. Such files are previously filled in SDcard-in. No user data is removed.
- Outgoing calls [automatically initiate outgoing calls]
    - Frequency: After how many seconds this filter will run
    - Duration: Call duration from the second call is initiated
- Outgoing SMS [automatically sends random characters as sms]
    - Frequency: After how many seconds this filter will run
    - length: SMS length generated randomly
- Idle [sets screen on and off]
    - Frequency: After how many seconds this filter will run
    - Duration: Seconds screen will go off and on

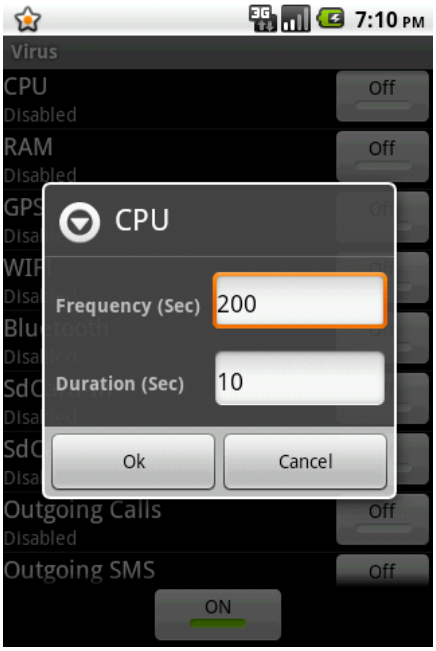

Figure 16 Editing virus settings for CPU

For example: Take the following given; frequency precision of 50% and duration precision of 0% along with outgoing calls of frequency 100 seconds and duration of 5 seconds. It means that every 0 to 200 seconds an outgoing call will be initiated for duration of 2.5 seconds to 7.5 seconds. If frequency precision and duration precision where 100% then every 100 seconds a 5 second duration call will be initiated.

It is always advised to set Frequency at least twice as duration and never less, else *Virus* will initiate two tasks at the same time and the task will keep over lapping over each other and crashes the mobile. Our aim to create a light weight malware that runs silently in the background and cannot be noticed by the user. For example, if we make a phone call for 10 seconds every 2 seconds, this will crash the whole system in few seconds by creating multiple requests even before the first request was made.

To provide diversity in our data set we should consider having predefined number in the *special contacts* and lower the *frequency precision* and *duration precision* features.

5.7 Process

5.7.1 Observation:

The data collected is in sqlite format and includes all the information for each criterion during that period. The handset monitor runs first with live mode off. If live mode is on, the tmp table is pruned and will be used for 30 minutes collector instead of the whole period. After 2 month period of data collection we noticed a large delay in handset response, mainly because the insert takes more time on larger tables. Although the operations being done is only insert, but it takes time and CPU load with limited resources mobile devices provides to do an sqlite query. This is not a big problem in this phase since it's just data collection and mobile response at this phase does not affect our output result or the algorithm runtime in anyway. The final product will be live mode being on permanently. This phase is just to collect as much data as possible for in depth testing phase later.

5.7.2 Simulator:

It is an application that runs on the mobile to simulate real time user behavior within few seconds. It takes as input the output of the export file generated by

Handset Monitor application. The program will then run on the mobile for few seconds and generate a detailed report about all anomalies detected.

Input: sqlite database generated by *HSM* consisting of one table

Output: detailed report about anomalies detected, such details are:

- Time of anomaly which occurred
- Indicator(s) that cause the anomaly
- All variables and indicators' data during the detection

During simulation the program will keep running even after finding such anomaly. Its only task is to process the log file and generate a report. Based on this report, algorithm is tweaked to better detect and differentiate user normal behavior from a certain anomaly.

Logs are processed against the algorithm inside the simulator and treating each row as a real time input. The model of the algorithm might change and altered countless times, but the log file is always the same. Such method saves countless days of experimentations by recollecting new data on every change in the main algorithm.

After several tests, we noticed that it is time consuming and not feasible to run all tests cases. For example, if we have 3 data sets with 4 variables each will give us 12 different test cases. To overcome such limitation, we run *"Cartesian product"* on MATLAB and created a hard coded array inside our java code that will run all test cases sequentially. The output will be appended to a filename *hsm_log.txt* located in the root directory of the SDCARD.

We divided each of the indicators into separate tables to better fetch information from the database. Tables' structures are as follows:

1. ***temp*** table structure:

Contains all inserts done during the last 30 minutes thus called temporary table, every 30 minutes a task is triggered to move all rows into several tables based on the indicator name. Table is pruned[3] after task is completed, inserts during this process will be on hold.

#	Column	Type	Default	Extra
1	id	int(12)		AUTO_INCREMENT
2	timestamp	int(13)	0	
3	attr	varchar(16)		
4	value	Varchar(32)		
5	Extra1	Varchar(32)	NULL	
6	Extra2	Varchar(32)	NULL	

1. **Id**: unique id provided automatically for each inserted row
2. **timestamp**: timestamp UNIX date in milliseconds
3. **attr**: Indicator name (example Bluetooth, CPU, RAM, etc..)
4. **value**: value for indicator
5. **extra1**: extra optional value for indicator
6. **extra2**: extra optional value for indicator

[3] Pruning is a terminology used in database system that indicates wiping out the table rows but keeping the table structure intact.

Each indicator has its own table, columns, and set of attributes. Each table have the same number of rows, each row represents duration of 30 minutes during the day. We limited table to 7 days => 336 rows only. We call each of the rows or 30 minute duration a time slot. We provide more attributes to each slot to better characterize user behavior.

2. **Indicators' table structure:** Contains all information about a certain indicator usage.

#	Column	Type	Default	Extra
1	timeslot	int(10)		PRIMARY KEY
2	tot_cycles	Int(10)	0	
3	rate_cons	Int(10)	0	
4	tot_avg	Varchar(32)	0	
5	Rate_change	Varchar(32)	0	
6	*			

Table 5 General sqlite table structure for an indicator.

1. **timeslot**: unique timeslot identifier for each 30 minutes
2. **tot_cycles**: Number of weeks passed on this slot while *HSM* is running
3. **rate_cons**: Number of cycles this timeslot is consistent
4. **tot_avg**: Value for indicator
5. **rate_change**: Extra optional value for indicator
6. *Extra columns that might be used for certain* indicators.

Notes:

- Tot_cycle is not always the same in all timeslots. Sometimes the mobile is off during certain hours or the Monitor application is not running.
- * Each table have its own columns and attributes that are the same in all with some additions depending on indicator requirements
- Computation of tot_avg varies from one indicator to another

More details about each attribute is defined in algorithm section

5.7.3 Algorithm

Mobile devices have limited resources in terms of memory and space. Dynamic programming and other similar approaches may provide with definite solutions but requires more time depending on the problem. Heuristic algorithms are introduced to find the most approximate solution where finding the solution in a predefined time is critical. Intrusion detection system (IDS) works in two phases; (1) monitoring user behavior (2) Identifying normal from abnormal trend. Phase (1) is already covered in *Handset Monitoring* application and *Virus* application was created as a supplement for real viruses that might permanently damage the mobile. Our Virus application does the same functionality of that of normal virus but with total control over its action.

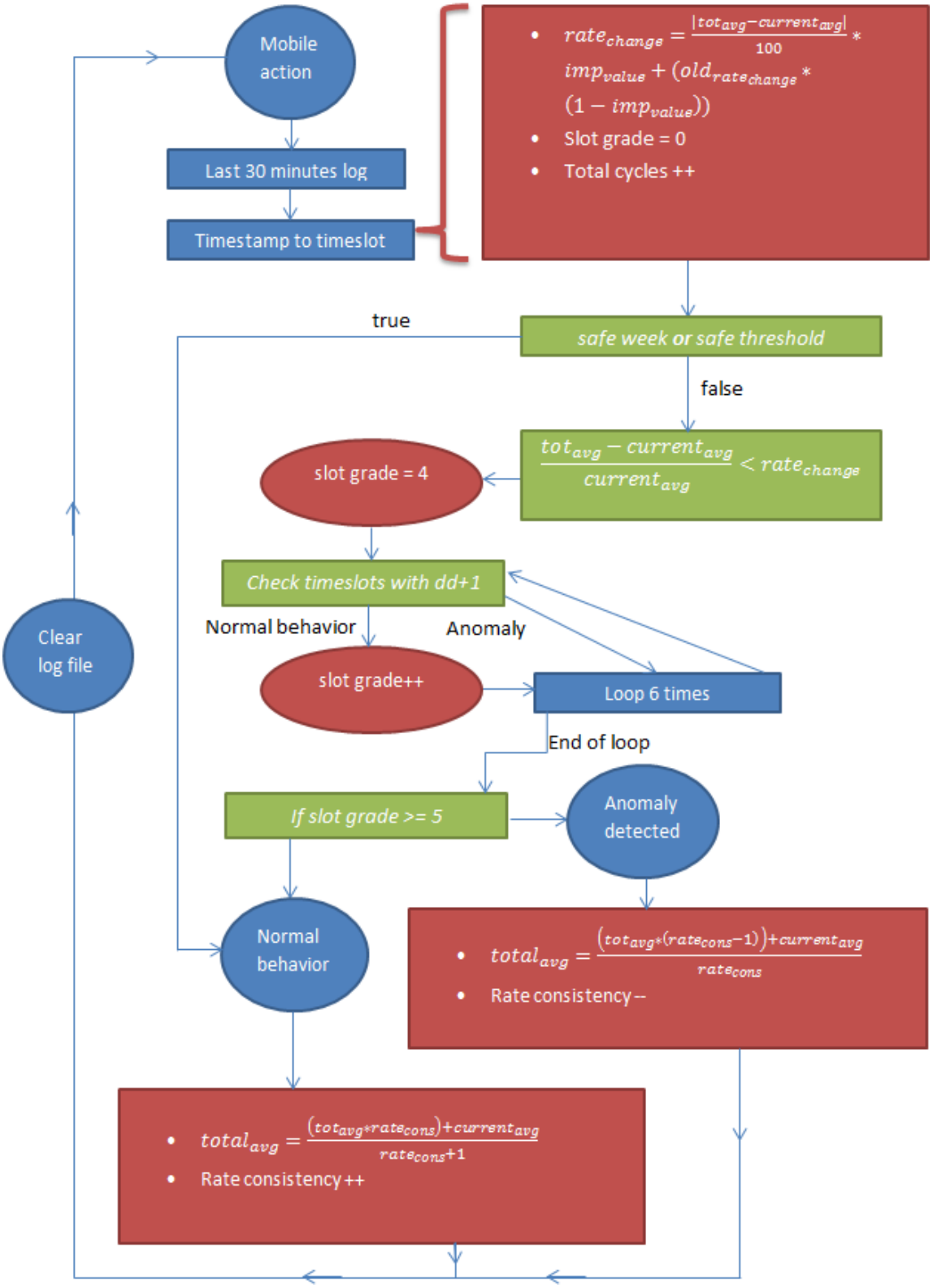

Figure 17 **Algorithm Flow chart**

5.7.3.1 Timestamp to timeslot:

As described in the simulator configuration section, all week is divided into slots each of 30 minutes. Each timestamp is mapped into a timeslot; therefore we use a slot identifier with special name convention *dhhmm*.

- *d*: Day number of the week from 1 to 7 from Sunday to Saturday respectively.
- *hh*: hour during the day in 24 hour format; thus value is an integer from 0 to 23.
- *mm*: resembles minutes. Since timeslots are 30 minutes long, value is either 00 or 30.

Example:

- *10830* maps to *Sunday 8:30 am to 8:59:59 am*
- *52000* maps to *Thursday 8:00 pm to 8:29:59 pm*

Consider: *dhhmm-30* < *dhhmm* < *dhhmm+30* and timestamp *T*

If *dhhmm* <= T < *dhhmm*+30 → *T* belongs to *dhhmm* timeslot

Note: timestamp is converted to timeslot upon every insertion to find its right place during the week.

5.7.3.2 Total cycles:

Tot_cycles is a variable inside each timeslot that increments from the day the application start. *Tot_cycles* defines how old each slot is, thus the running time of the mobile and when it is online as well as when the *HSM* application is

running. If the device is 100% on and the handset monitor is activated 100% of the time *tot_cycles* will be the same.

5.7.3.3 Rate consistency:

rate_cons is a variable that defines the strength of a slot denoted by an integer. Every week, rate_cons for each slot is updated once. Its runs the algorithm and then decides if this slot shows anomaly or consistent with its history. The higher the value the more consistent this slot is. For example a value of 5 means this slot is consistent for 5 weeks. The decrease in value implies a change in user behavior, for example an employee shifted to work at night, this employee behavior changed, and our proposed algorithm should detect this as a change and not as an anomaly. Consequently, rate_cons will decrease by 1 every week until it adapts to the new change.

Rate consistency is computed using the below formula:

If

$$\frac{tot_{avg} - current_{avg}}{current_{avg}} < rate_{change}$$

Then

$$rate\_cons{+}{+}$$

Else $rate\_cons$--

5.7.3.4 Current average:

$current_{avg}$ is the average value of the $temp$ table for a certain indicator just before $temp$ table gets pruned.

5.7.3.5 Rate change:

$rate_{change}$ is the flexibility value of each slot. Rate change defines how flexible this slot can be and how often a certain indicator varies. Value ranges between 0 and 1. The *new temporary rate change* value is computed using the below formula:

$$new_{tmp_{rate_{change}}} = \frac{|tot_{avg} - current_{avg}|}{100}$$

Equation 1 New computed temporary rate change

To compute the new rate change we give more importance for $new\_tmp\_rate\_change$. Importance value denoted by imp_value and is by default 20%. In other words $rate\_change$ will be adjusted by 20% of $new_{tmp_{rate_{change}}}$. All values range between 0 and 1. If we want to progress the new value faster we increase imp_value. Therefore our new rate change is computed by the below equation

$$rate_{change} = new_{tmp_{rate_{change}}} * imp_{value} + (old_{rate_{change}} * (1 - imp_{value}))$$

Equation 2 New rate change

5.7.3.6 Total average:

tot_{avg} is the most essential value of indicators' attributes inside each slot. It is the total average of all values. All values are computed based on $rate\_cons$. We compute tot_avg using the formula below:

If

$$\frac{tot_{avg} - current_{avg}}{current_{avg}} < rate_{change}$$

Equation 3 Anomaly detection based on rate change and average

Then

$$total_{avg} = \frac{(tot_{avg} * rate_{cons}) + current_{avg}}{rate_{cons} + 1}$$

Equation 4 New total average if normal behavior detected

Else

$$total_{avg} = \frac{(tot_{avg} * (rate_{cons} - 1)) + current_{avg}}{rate_{cons}}$$

Equation 5 New total average if anomaly detected

5.7.3.7 Normalize:

To adapt to new changes and to avoid false positives we migrate tot_{avg} and $rate_{change}$ into the neighboring slots, we use the table below to define imp_{value}.

Slot	Imp_value 1	Imp_value 2
Slot – 3	4%	10%

Slot – 2	7%	20%
Slot - 1	15%	30%
Current slot	20%	40%
Slot + 1	15%	30%
Slot + 2	7%	20%
Slot + 3	4%	10%

Table 6 Normalization table showing rate change percentages for different time slots.

Imp_value 2 shows larger values than imp_value 1; meaning that the algorithm will adapt faster to the new changes due to the new data HSM provides. In our results we will observe the difference between the two and select the best for each criterion. For future work, it is best to make those variables dynamic and adapts in accordance to user behavior. The number selection is

For current slot:

- $rate_{change}$ previously altered by 20% or 40% of $new_{tmp_{rate_{change}}}$
- tot_{avg} for current slot is only computed using $rate\_cons$

For neighboring slots:

- $rate_{change}$ changed by imp_{value} ratio of $new_{tmp_{rate_{change}}}$
- tot_{avg} changed by imp_{value} ratio of $current_{avg}$

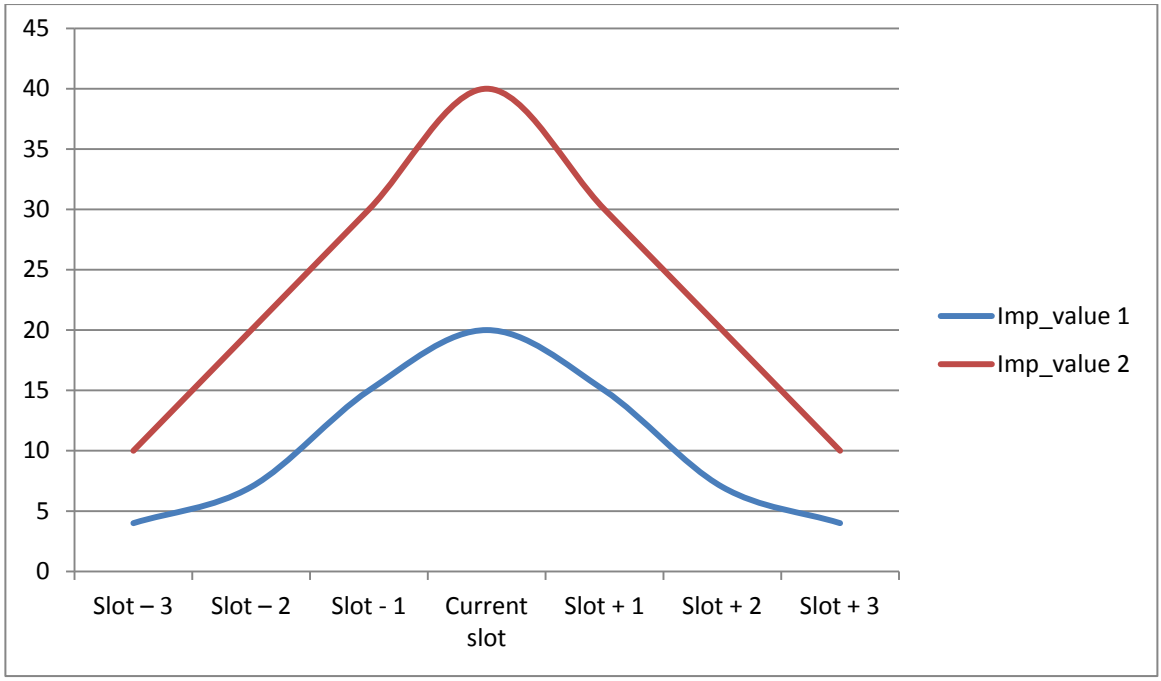

Figure 18 Neighboring imp values percentage change

Based on Gaussian model [32] we derive the above values based on 40% and 15% with smoothed curve. There are 7 slots in total being altered every 30 minutes giving a range of 3.5 hours where we consider user behavior to be similar.

5.7.3.8 Slot Grading:

Each slot occurs once a week; any new event will be compared not only to its corresponding slot but to the same time each day of the week called children slots. We give 4 points for the parent slot and 1 point for the rest of the slots equivalent for the rest of the days of the week making a total of 10 points. For a slot to pass it should have a grade of 5 and above. In other words the slot must either have the approval of the parents slot and any child slot. In case parent slot

didn't approve, 5 of 6 children slots must approve in order for the main slot to pass. Pseudo code as below:

```
slot_grade = 0;
if(slot(current, average) != anomaly)
  slot_grade=4;

for(current-3 to current +3 denoted as n_slot){
 if(slot(n_slot, average))

   slot_grade++;

}
```

For example, consider timeslot 41630, the children slots will be {51630, 61630, 71630, 11630, 21630, and 31630}. Parent slot will receive 4 points and rest of the children nodes 1 point each.

The main purpose of this method is to minimize the number of false positives and alert of possible false negatives. After applying slot grading method to our logs we noticed an immediate improvement with a reduction of false positives to 60 - 80%. Therefore slot grading is always enabled in all our test cases.

5.7.3.9 Safe Weeks:

We define a safe mandatory initial period set by default into 3 weeks labeled *safe_weeks*. During *safe_weeks* period, the algorithm computes averages and updates each slot as well as normalizes the neighboring nodes. If anomaly was detected it will be ignored and considered safe and values will be computed as if it was safe. This gives the application more time to learn about the user behavior. In logic the longer this safe period is the better and more accurate our results are, in the result section; it will show in depth details about how the initial *safe_weeks* will affect the output. The only drawback in safe_weeks is

when a real anomaly occurs during this time, in such case, it will be considered safe and re-computes the values. This is why we normalize the data throughout the day.

5.7.3.10 Safe Threshold:

With certain indicators, user behavior doesn't have a restriction, for example phone calls or SMS that was never occurred during certain time will be considered an anomaly. Thus, we define a safe threshold that allows an x number of actions during any 30 minutes, beyond that the application will launch the algorithm to check. The value is static and is defined prior to the start of the algorithm. Default number is 3 and it illustrates the maximum allowed number of actions done during 30 minutes duration (0 to 29 minutes and 30 to 59 minutes). In case of anomaly but it's still under safe threshold it will be considered safe and algorithm will recomputed the variables accordingly; such value will provide more flexibility for the application in case of sudden change in user actions.

5.7.3.11 SDCARD and STORAGE computations:

For some indicators as in SDCARD and STORAGE the change behavior changes differently than others. When the user downloads or removes a file the change affects the whole timeslots that follows. In the normal algorithm behavior, if an anomaly was detected in slot a, slot $a+1$, $a+2$, ..., $a+x$ will issue an anomaly as well till the normalization function fixes the issue. To overcome such limitation we should set the same value for all the proceeding slots. The idea and the concept of slot are then eliminated with such change and we can rely on one slot

only for computation. For example if a slot have a value of 100, then all timeslots will be updated to value 100 as well before proceeding in running the algorithm for the next slot.

5.7.3.12 SMS and CALL computations:

Unlike other indicators, SMS and CALL entries have extra attributes logged along with them; *phone_rate* and duration (check section 5.4.10). The same algorithm to compute *avg_rate* is applied on *phone_rate* and *duration*. In such case we have 3 criteria, and 2 out of 3 should pass the anomaly test for a certain timeslot to pass.

Chapter 6

Results

6.1 Battery Consumption:

Battery type: Rechargeable Lithium-ion battery

Capacity: 1400 mAh

Advertised talk time:

- WCDMA: Up to 390 mins
- GSM: Up to 400 mins

Advertised standby time:

- WCDMA: Up to 360 hours
- GSM: Up to 340 hours

Battery times (talk time, standby time, and more) are subject to network and phone usage.

A Standby time specification is an industry standard that is only intended to allow comparison of different mobile phones under the same circumstances. Power consumption in a standby state is strongly dependent on factors including but not limited to network, settings, location, movement, signal strength and cell traffic . Comparisons of different mobile phones using such a specification can

therefore only be done in a controlled laboratory environment. When using any mobile phone in real life circumstances for which the mobile phone is intended, the standby time could be considerably lower and will be strongly dependent on the factors as mentioned above.

6.2 Benchmarking:

Benchmarking the mobile prior to testing helps us evaluate the impact our proposed security algorithm might change the performance of the device. Most important factor being monitored is the battery consumption in different modes such as idle mode, stress mode; in both with and without the security program being installed.

Action	Time
Talk Time (2G)	6 hours and 40 minutes
Talk Time (3G)	4 hours and 10 minutes
Video Playback	5 hours and 20 minutes
Audio Playback	10 hours and 10 minutes
Web Surfing	10 hours and 10 minutes
Idle Time without network	400 hours
Idle time with network[4]	270 hours

Table 7 Time required by running only 1 action till battery drains completely.

[4] Represents the value of idle state without having sim card or any kind of network connection such as Wi-Fi enabled.

When handset monitor *(HSM)* was activated we noticed no change in any of the above values in the table above. This is because the application monitors user actions and in the above cases it was mostly one action being made. For example if a phone call was made once, the *HSM* will activate and log this action once. Therefore further analysis was required to compute the consumption of each action. Such analysis includes battery consumption per minute, CPU usage, and processing time for algorithm.

6.3 Stress Test:

Stress tests are used to evaluate a device under heavy conditions. Stress test is performed by running a small script that will utilize the CPU until the mobile battery drains and the mobile closes. In normal conditions Android will kill the process after some time to avoid abusive behavior from an application that fully utilizes device resources but we made sure to keep it alive in case the process was killed. *HSM* was run under such condition and the algorithm run time average was monitored, below are the results:

Battery drain time: 2 hours and 6 minutes

Average response[5] from HSM (without stress test): 46ms

Average response from HSM (with stress test): 191ms

All other applications lagged to over 60 times more in response. By using Eclipse, we monitored the usage of each of our application and it shows very low

[5] Average response represents the time required for HSM to capture an action and input it to sqlite database as an entry.

memory consumption 12Mb of memory and almost 0.1 of CPU usage using UNIX measurements, and medium IO utilization of 3Mb which consumed less than a fraction of a second. Basically IO is due to the database being read of size around 2Mb and the rest of the packages are loaded into memory.

Under stressful conditions the *HSM* shows no signs of lagging considering algorithm is run only once every 30 minutes duration and logging is done per action only; making *HSM* a very light weight application.

6.4 Test report:

HSM logs all actions done by the user, but we perform tests on the following variables: SMS_IN, SMS_OUT, CALL_IN, CALL_OUT, RAM, CPU, SDCARD, and STORAGE. We will run the test on 3 datasets logged by 3 different users. We will then inject anomaly records randomly into the set as a bunch; defining bunch as being 4 or more actions done during 30 minute duration for the same indicator, for example 4 more SMS or calls. We will then run our virus application on a 3rd dataset and check it against *HSM*. The outcome should give as little false positives as possible and ability to detect most of anomalies.

1. Safe weeks: 0, 1, 2, 3
2. Safe Threshold: 0, 1, 2, 3
3. Normalize: 1, 2
    - Set 1 { 0.04, 0.07, 0.15, 0.2, 0.15, 0.07, 0.04}}
    - Set 2 { 0.1, 0.2, 0.3, 0.4, 0.3, 0.2, 0.1}

For simplicity the following name conventions are used:

- *SW-n* for safe weeks; where n is a positive number.
- *ST-n* for safe threshold; where *n* is a positive number.
- *N-n* for normalize; where *n* is a positive number.
    - Represents which normalizing set being used.

Row with best settings is marked with grey in each of the tables in the appendix section.

Some terminologies:

- *Anomaly detected*: total number of anomalies detected by HSM, detected anomalies might be right or wrong.
- *False positives*: number of wrong anomalies being detected.
- *False negatives*: number of right anomalies that were not detected.

Total number of assumed anomalies to be detected:

Anomaly detected – false positives + false negatives

Ratio of anomalies detected is:

(anomaly detected-false positive) / (anomaly detected + false negative)

6.4.1 Dataset 1:

HSM was installed on a new mobile device for duration of 8 weeks and used by one person 'Fawzi Breidi'. During the first 4 weeks he performed continuous installation of applications. Regular incoming and outgoing SMS throughout 8 weeks except for the last 2 weeks the device received more incoming SMS and no

outgoing SMS. In the last 3 weeks of usage, mobile usage decreased and mainly focused on phone calls only. Sleeping behavior was irregular throughout the whole usage, thus mobile usage time was irregular. Virus was activated twice to initiate many external phone calls at once then hang up the line. Due to software error only the last 2 weeks were logged.

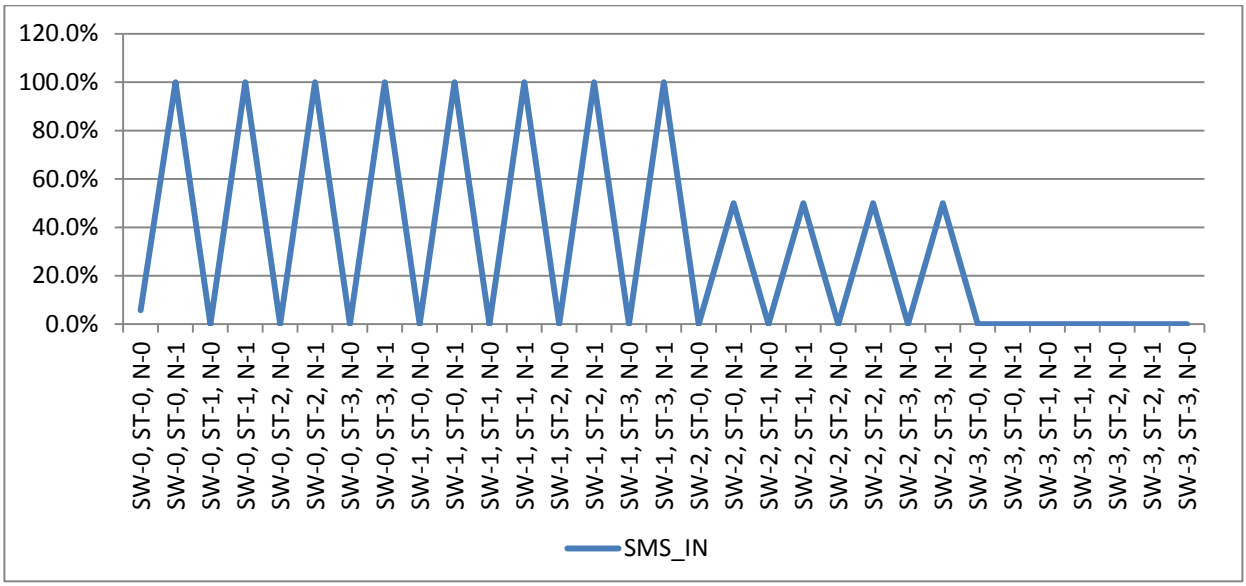

Figure 19 SMS_IN detection rate for anomalies in dataset 1

Dataset 1 is only 2 weeks, therefore we can notice that all SW-2 and above will give no anomaly at all. There was no activity for Call_IN, CALL_OUT, SMS_OUT, thus generated no anomalies, no false positives nor false negatives.

Second observation is the time difference it requires to run algorithm (50ms average) and the normalization function (130ms average). Third observation is the time required for SMS_IN for SW-0, ST-0, N-0 is above 1000 seconds caused by turning off the screen while he algorithm was working. Idle state makes the algorithm process to sleep and will not be executed. This case is only

applied during testing and it's a method used in Android to increase battery size and avoid abusive processes. SMS_IN showed 100% detection rate when we normalize and this detection rate decreases as SW increase since we have a dataset of 2 weeks only. As for false positives it is almost to null in almost all cases, this proves one of the following: either the algorithm is highly effective or it is too aggressive and detects the slightest change. By observing the false negatives we notice that depending of the attributes it is too aggressive. After full analysis we can say that everything over SW-2 is not effective, and anything below ST-1 is also not accurate. This conclusion can also be applied to all other datasets.

6.4.2 Dataset 2:

HSM was reinstalled the same mobile device used in Dataset 1 and without removing any other application for a duration of 4 weeks and used by two persons 'Fawzi Breidi' and 'Khalil Breidi'. During the first week 'Khalil' used the device for general mobile purposes and often light game applications for 1 hour ranging from 9pm to 12pm. The device was then used by 'Fawzi Breidi' for the next 3 weeks and behavior was altered substantially. Mainly the mobile was again for development purposes and the HSM was on and off many times. During the last week the simcard was removed and it was used only for network purposes. Virus was activated twice randomly during the whole process.

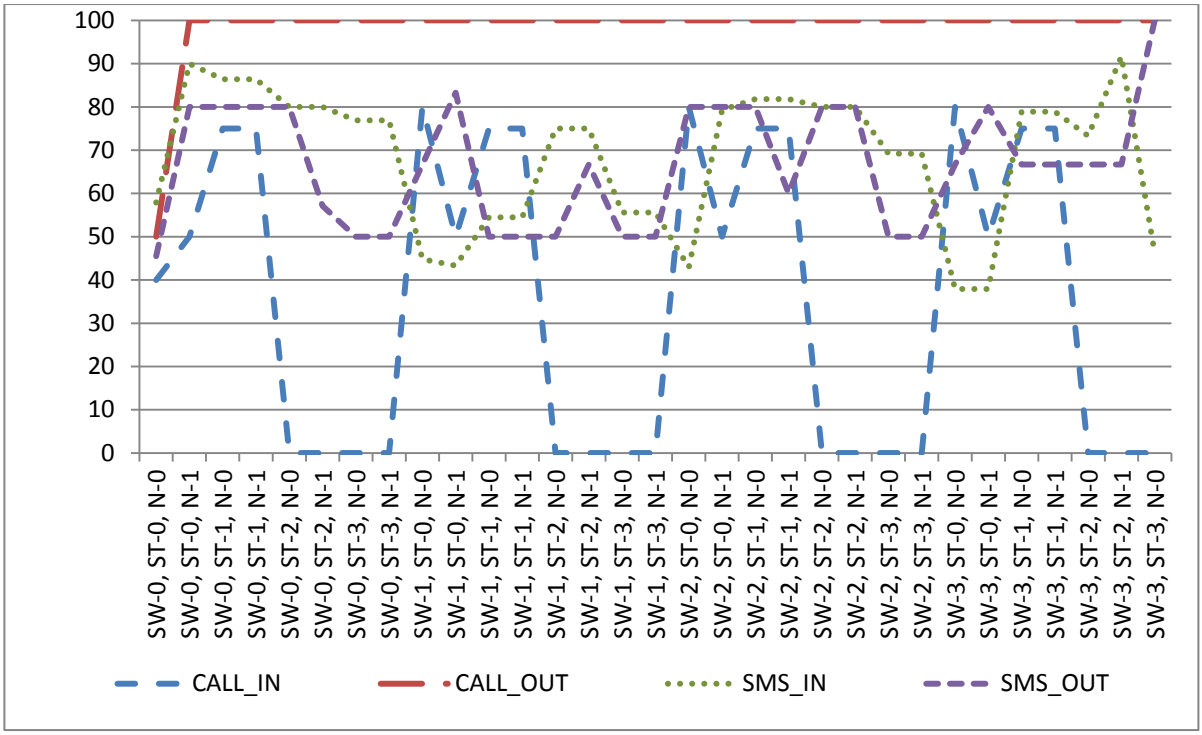

Figure 20 Anomalies detection rate in dataset 2

From the above results we can clearly see a rapid improvement when the dataset duration is longer. But SW didn't do much as the percentage of detected anomalies is slightly decreasing. The criteria that made all the difference is ST, where it tends to zero anomalies in CALL_IN whenever we have ST-3. Normalization did improve the results in SMS_IN and SMS_OUT. As for CALL_OUT it was easily detected in almost all cases. Normalization smooth's user profile and might not be suitable for everyone. The threshold of 2 and above will removed all kinds of false negatives. When the virus was activated it sent 6 messages in less than 2 hour duration; this behavior will mimic most common malware that builds the virus source via SMS.

6.4.3 Dataset 3:

HSM was installed on a device that was previously used for development purposes by 'Karim Jahid' run for more than 10 weeks. The device was used for both development and normal mobile usage but HSM was sometimes stopped during testing. During usage of the device an event took place which caused major change his mobile usage, mostly mobile calls, sending and receiving SMS. Virus was not activated and totally dependable on the user behavior. The below results should indicate zero anomaly detection to have best results.

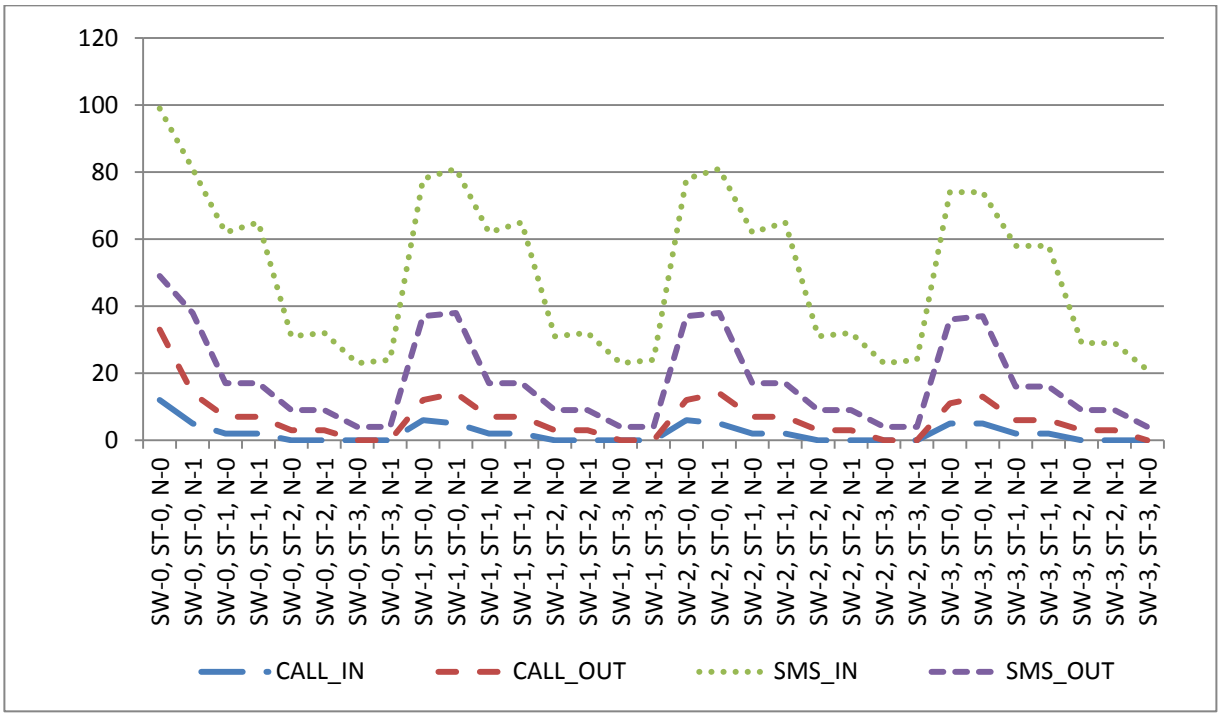

Figure 21 Anomalies detection rate in dataset 3

Since Virus application was not activated during this process, values that shows zero anomalies indicates the best result. Based on the above we notice that number of anomalies decrease to zero in CALL_IN, CALL_OUT, and SMS_OUT when ST-3 and SW-1 and above. Those values get slightly better in SW-2 and

SW-3. On the other hand N-1 and N-0 doesn't show much improvement to the above. SMS showed less accurate result with a minimum of 20 anomalies, best results are when we use ST-3, SW-3 and N-0. The most important observation is the direct relation between all indicators and the number of anomalies being detected. This shows that when an anomaly is detected in on indicator it tends to effect other indicators as well, although in those trust results we do not show this relation and how it functions, but this is left for future work as an idea for collaboration of indicators.

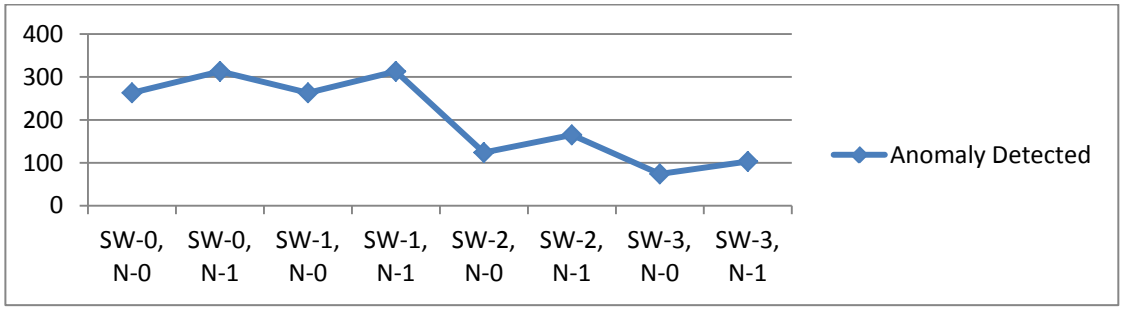

Figure 22 CPU chart diagram for dataset 3 results

CPU was computed (check appendix) and not included since the mobile was used in development and testing thus CPU load was always high during those periods. Yet we can notice when we had SW-3 the anomalies decreased rapidly. Normalization for CPU didn't help at all.

6.5 Common observation:

6.5.1 RAM:

"FREE RAM IS A WASTED RAM" the golden rule with Android. HTC Desire Z device have 512Mb of ram with fixed allocation all the time. Android's memory

management is a little different than other operating systems. Android uses a virtual machine to manage memory allocations for applications, like Java and .NET. But, unlike Java and .NET it also manages the process lifetimes. Android *VM* ensure that each application is responsive; to do that it kills and stops processes as necessary to free as much resources as possible for front end applications labeled as high priority Introducing the Android Computing Platform Introducing the Android Computing Platform [32].

Dalvik is the process virtual machine for android to manage running applications smoothly. Each instance of Dalvik contains its own process that is separate from other instances; thus relinquishing all memory and process management responsibilities to Android runtime environment which handles killing and managing all resources.

As such, Dalvik manages to monopolize the most out of the memory and keeping most used applications in RAM. This leads to constant values for free and used memory by *HSM* making monitoring this criterion useless. The results after further experimenting for over 8 weeks on different datasets show less than 0.5% difference. Therefore using *RAM* will never lead to a solution nor will it help to detect any kind of anomaly or change in behavior with the device with any similar application.

6.5.2 SDCARD:

Android heavily relies on SCARD for its usage and installing new third party applications as well as camera and other similar applications. Running the same algorithm of *HSM* will cause in anomaly every time a new application being installed. Over 280 anomalies where detected after performing 50 actions that involves installing new application. 280 anomalies are often continuous and it requires several cycles until our logs normalizes. Our second approach included normalizing the data at once rather than a ratio for example (20%) this lead to exactly 50 anomalies detected for the 50 newly installed application, although it detected the change but it can never be considered an anomaly since it was a user action requesting for installing such application. In the third attempt, we only computed the increase and decrease speed in space which makes more sense in real life usage, but still the results reflect only the average size of application being installed or new song being downloaded, thus making is a very vague criterion and not useful for detecting any anomalies.

6.5.3 Storage:

Android takes control of the internal disk and allows little interaction with it to avoid tampering with system files, in some Android devices external memory is a must to install applications. In our test device HTC Desire Z we installed a file manager that allows us to check such internal files. The change in internal storage was minimal and we noticed only few changes along our heavy usage on the device; such changes include but not limited to; when the operating system

updated itself twice at two different times and when I configured my email and started receiving email attachments through my email android client. In all those cases *HSM* considered them an anomaly since sudden change was made along a very stable profile for Storage. The same weakness of SDCARD applies here and therefore this criterion is vague and not useful for our tests.

6.5.3 Normalize

In all our data sets being collected we can notice that after normalizing the results we received less false positives whereas in indicators without normalization have this value high. Normalization was a great improvement to our dataset analysis and as such we altered the values for normalization as set 1 and set 2 in *N-1* and *N-2* respectively representing different values to adjust neighboring values more or less often. N-2 shows higher values meaning it will adjust to the new changes more rapidly. Using N-2 was not always the best choice and we noticed little change between the two sets. This highly depends on the user behavior and his actions on the device. Having such hard coded values gives a rough estimation on how often we should adjust to new changes. This gives the idea to dynamically adjust normalized values based on the speed of change of the user as a room for improvement.

6.5.4 CPU

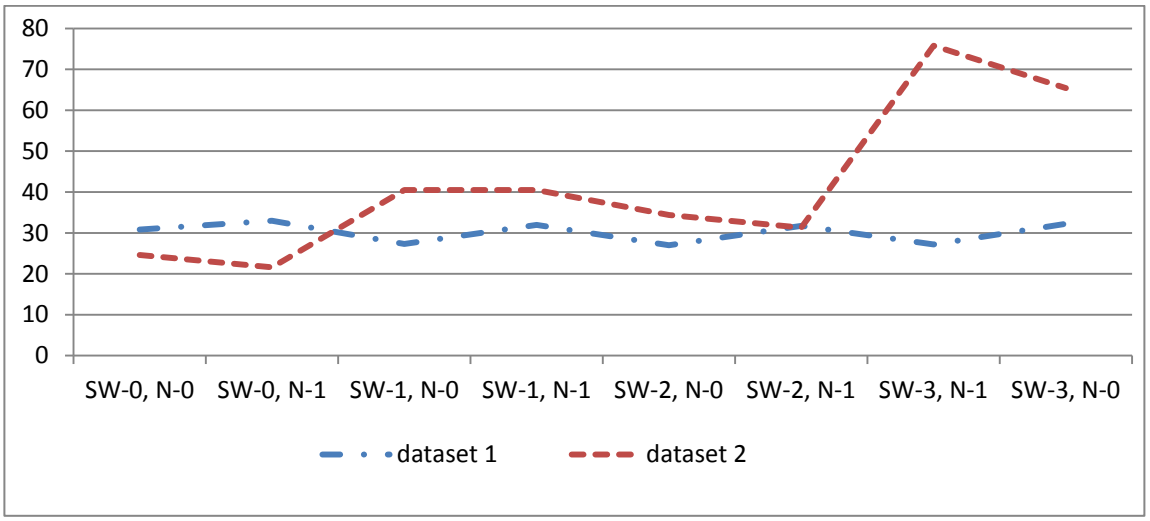

Figure 23 CPU detection rate for dataset 1 and dataset 2

We can directly notice that CPU doesn't function the same in both datasets. On the other hand we notice the great improvement to almost 85% detection rate when we normalize and allow up to safe week of 3 for data set 2.

6.5.4 Runtime

Based on results from data sets we notice that the main difference is each test case is when we normalize as it consumes 3 times more time. CPU consumes more time to compute than rest of the indicators. In all cases the time consumed for 1 cycle (30 minutes) is never more than 130 millisecond which makes HSM a very light weight security software.

Chapter 7

Conclusion

In this research we introduced an effective standalone IDS approach to detect anomalies on the device by monitoring user behavior and creating a highly detailed profile throughout the week. Due to limited resources mobile devices have we computed averages for each 30 minutes and gave it characteristics about the rate change during this slot. The result was a light weight algorithm to detect anomalies effectively and without using any external server for computation. The number of anomalies detected showed a detection rate of over 90% and 0 false negatives and 0 false positives. Typically the algorithm requires 2 weeks at least to have a solid profile and a safe threshold of 2. Normalization function did help in most cases so it is always advised to have it on. More focus should be provided for standalone IDS approach as it is more practical for users who are not connected to the internet. Most server based approaches are not practical and not concert for normal users.

7.1 Future work:

7.1.1 Indicator Collaboration:

All test results were done for one indicator at a time. Relation between one indicator and another is possible for example we noticed high disk IO usage is accompanied with high CPU load and in constant CPU load with all phone calls

made and received. The latter observation can be explained since in almost every case no other action is made during a phone call. Although CPU load increases and battery drain increases depending on the network and its signal strength but it keeps the same trend for each user, thus allowing us to make collaboration between indicators to strengthen detection of anomalies.

7.1.2 Using Hierarchical IDS

In the first part of the report we described the different types of IDS introduced and being used, we choose a standalone application to provide a more practical application for end customer usage. A better approach is by using a Hierarchical IDS which is an extended version of distributive and cooperative IDS with multi-layered network infrastructure. The basic idea here is to share resources and the knowledge base being collected by each node and to define trust entity for both sides. For example: Node A trusts node B with x ratio, but B trust A with y ratio; x and y here can be different and A can decide to trust B but not the other way around. Trust ratios can also change with time depending on the consistency of data being provided. A real cooperative manner will not only share the knowledge base, but also share the trust ratio for each of the neighboring nodes to better evaluate each node.

7.1.3 GPS based clustering

GPS logging is already logged by our HSM application but never used the data. One major improvement to this indicator is to log the exact location of the user at all times when GPS is active and consequently creating clusters for possible

locations this user might be in. The idea proposed is called geofencing and it creates a virtual boundary about where the user can be and cannot be, it will also save all possible roads taken. When the user is outside those virtual fences it will be considered an anomaly. Though it might not be accurate to issue an immediate alarm, a better approach is to make a hidden red flag when this occurs and count red flags till a certain threshold before it considers it as an anomaly. If the red flag turned normal again, the algorithm should adapt to the new changes and rewrite the cluster. The same concept of alert system used in HSM can be used where each cluster can have a consistency ratio which defines how fixed user movements are.

7.1.4 Dynamic values

In our research we have hard coded static values for safe weeks and safe threshold values. Though user behavior with the mobile varies significantly from one user to another and by trying out different values some showed better results in some special cases. A better way to solve such cases is to start with a default value of 3 and with time the value will change, either increasing or decreasing. The output result will be faster detection for anomalies.

7.1.5 Multiple rate change for same timeslot

Each slot in the database represent user behavior during 30 minutes, one of the major elements that defines such behavior is rate consistency and rate change. Both values delineate the computed average. This is the case with all of our indicators and the ideal case of on single average such as RAM and CPU. In

other cases such as CALL_IN and CALL_OUT we have 3 averages: number of phone calls, average ratio of calls from phone book, and call duration. Each of those values has its own average computed but they change according to two variables (rate consistency and rate change). A better solution is to use two different variables for each average computed for each slot proving a lot better and more accurate results. This new addition might have a slight increase in the runtime of each slot, yet the positive feedbacks will concealment it.

References

[1] M. Stefik, "The role of NSF's support of engineering in enabling technological innovation - phase II," *SRI International, Arlington*, Final Rep., May 1998

[2] Cell phone history (2012, Mar.) [Online]. *Monash University of Engineering*. Available: http://www.ecse.monash.edu.au/staff/jeana/aboutarmstrong.html

[3] S. Acharya. (2008). Worldwide mobile cellular subscribers to reach 4 billion mark late 2008 [Online]. *International Telecommunication Union, Geneva*. Available: http://www.itu.int/newsroom/press_releases/2008/29.html

[4] "2011 mobile threats report," *Juniper Networks, Sunnyvale, Rep., February 15, 2012*

[5] T. Dyhouse. (2010, Aug.). The growing security threat from mobile phone apps. [Online]. *ComputerWeekly.com*. Available: http://www.computerweekly.com/opinion/The-growing-security-threat-from-mobile-phone-apps

[6] 4 generations of cell phones. (2010, Apr.). *CellPhones.org* [Online]. Available: http://cellphones.org/blog/4-generations-of-cell-phones

[7] K. Peter (2009, Sep). Analysis and comparison of 1G , 2G , 3G ,4G and 5G telecom services [Online]. *Hub Pages*. Available: http://kevin-peter.hubpages.com/hub/3G-and-4G-Mobile-Services

[8] I. Brodsky. (2007, Aug.). New mobile security threats are emerging [Online]. *Network World*. Available: http://features.techworld.com/mobile-wireless/3586/new-mobile-security-threats-are-emerging

[9] B. Suthoff. (2001, Jan.). First impressions matter! 26% of apps downloaded in 2010 were used just once [Online]. *Localytics*. Available: http://www.localytics.com/blog/2011/first-impressions-matter-26-percent-of-apps-downloaded-used-just-once

[10] S. Radwanick, "The 2010 mobile year in review," *comScore, Reston, Rep., March 7, 2011.*

[11] M. Villano. (2005, Mar.). New security threats target cell phones, mobile devices. [Online]. CRN. Available: http://www.crn.com/159901521/printablearticle.htm

[12] E. Oswald. (2005, Feb.). Mobile phone virus surfaces in US [Online]. *Beta News.* Available: http://betanews.com/2005/02/21/mobile-phone-virus-surfaces-in-us

[13] B. Krebs. (2005, May). Paris Hilton hack started with old-fashioned con. [Online]. *Washingtonpost.com Staff Writer.* Available: http://www.washingtonpost.com/wp-dyn/content/article/2005/05/19/AR2005051900711.html

[14] W. Ashford. (2010, July). Millions download suspicious android wallpaper [Online]. *ComputerWeekly.com*. Available: http://www.computerweekly.com/news/1280093401/Millions-download-suspicious-Android-wallpaper

[15] S. McGlaun. (2011, Feb). Cell phone security threats on rise while spam decreases [Online]. *Daily Tech.* Available: http://www.dailytech.com/article.aspx?newsid=20859

[16] M. Hamblen. (2010, May). Smartphone sales shoot up, benefiting Rim, Apple, Android [Online]. *Computer World*. Available: http://www.computerworld.com/s/article/9176972/Smartphone_sales_shoot_up_benefiting_RIM_Apple_Android

[17] Z. *Yongguang, H. Yi-An and L. Wenke.* (2003). "Intrusion Detection Techniques for Mobile Wireless Networks," *Wireless Networks*, vol. 9, no. 5, pp. 545-556, 2003.

[18] M. Pellicer, A. Herrero, E. Corchado and A. Abraham, "Hybrid multi agent-neural network intrusion detection with mobile visualization," *Innovations in Hybrid Intelligent Systems*, vol. 44, pp. 320-328, 2007.

[19] W. Yuy, Y. Sun, K. J. R. Liuy and Z Hang, "Trust modeling and evaluation in ad hoc networks," *GLOBECOM 05 IEEE Global Telecommunications Conference*, vol. 3, pp. 1862-1867, 2005.

[20] M.R. Lyu, J. Liu and X. Li, "A trust model based routing protocol for secure ad hoc networks," *Aerospace Conference, Vol.2, pp 1286 – 1295, 2*004.

[21] E. M. Royer and C. E. Perkins, "Ad hoc on-demand distance vector routing" in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, pp. 90-100 , February 1999.

[22] K. Wang and S. Stolfo, "Anomalous Payload-Based Network Intrusion Detection," *Recent Advances in Intrusion Detection*, vol. 3224, pp. 203-222, 2004.

[23] J. Sobral, M. Notare, K. Juca and A. Boukerche, "Human immune anomaly and misuse based detection for computer system operations: Part II," in *Parallel and Distributed Processing Symposium*, pp. 8, April 2003.

[24] W. Sanders, H. Khurana and R. Berthier, "Intrusion detection for advanced metering infrastructures: Requirements and architectural directions," in S*mart Grid Communications (SmartGridComm), 2010 First IEEE International Conference, pp. 350 – 355*, October 2010.

[25] T. Anantvalee and J. Wu, "A Survey on Intrusion Detection in Mobile Ad Hoc Networks," *Wireless Network Security*, Springer US, pp. 159-180, 2007.

[26] Y. Zhang, W. Lee and Y. A. Huang, "Intrusion Detection Techniques for Mobile Wireless Networks," *Wireless Networks*, vol. 9, no. 5, pp. 545-556, 2003.

[27] A. Patcha, K. Nadkarni and A. Mishra, "Intrusion Detection in Wireless Ad Hoc Networks," *Wireless Communications, IEEE*, vol. 11, no. 1, pp. 48-60, 2004.

[28] A. L. Shimpi and B. Klug. (2011, Nov). iPhone 4S preliminary benchmarks: ~800MHz A5, slightly slower GPU than iPad 2, still very fast [Online]. *Anand Tech*. Available: http://www.anandtech.com/show/4951/iPhone-4s-preliminary-benchmarks-800mhz-a5-slightly-slower-gpu-than-ipad-2

[29] D. Goldman. (2011, Mar.). Windows phone to eclipse iPhone sales by 2015 – forecast [Online]. *CNN Money*. Available: http://money.cnn.com/2011/03/29/technology/windows_phone_7_forecast/index.htm

[30] Android. (2007). Platform versions [Online]. *Android Developers – Resources. Creative Commons Attribution 2.5.* Available: http://developer.android.com/resources/dashboard/platform-versions.html

[31] A. Shabtai, U. Kanonov and Y. Elovici, "Intrusion Detection for Mobile Devices using the knowledge-based, temporal abstraction method," *J. Syst. Softw.*, vol. 83, no. 8, pp. 1524-1537, August 2010.

[32] S. Komatineni and D. MacLean, "Introducing the Android Computing Platform," *Pro Android 2*, APress, pp. 6-8, March 2012.

Appendix

CALL_IN	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	0	0	0	29.589
SW-0, ST-0, N-1	0	0	0	31.481
SW-0, ST-1, N-0	0	0	0	28.834
SW-0, ST-1, N-1	0	0	0	29.593
SW-0, ST-2, N-0	0	0	0	27.004
SW-0, ST-2, N-1	0	0	0	29.693
SW-0, ST-3, N-0	0	0	0	27.083
SW-0, ST-3, N-1	0	0	0	28.975
SW-1, ST-0, N-0	0	0	0	27.427
SW-1, ST-0, N-1	0	0	0	29.967
SW-1, ST-1, N-0	0	0	0	26.220
SW-1, ST-1, N-1	0	0	0	29.456
SW-1, ST-2, N-0	0	0	0	26.784
SW-1, ST-2, N-1	0	0	0	28.523
SW-1, ST-3, N-0	0	0	0	25.353
SW-1, ST-3, N-1	0	0	0	29.954
SW-2, ST-0, N-0	0	0	0	26.050
SW-2, ST-0, N-1	0	0	0	28.539
SW-2, ST-1, N-0	0	0	0	27.266
SW-2, ST-1, N-1	0	0	0	29.129
SW-2, ST-2, N-0	0	0	0	26.075
SW-2, ST-2, N-1	0	0	0	28.274
SW-2, ST-3, N-0	0	0	0	26.598
SW-2, ST-3, N-1	0	0	0	27.776
SW-3, ST-0, N-0	0	0	0	27.859
SW-3, ST-0, N-1	0	0	0	28.909
SW-3, ST-1, N-0	0	0	0	27.627
SW-3, ST-1, N-1	0	0	0	29.494
SW-3, ST-2, N-0	0	0	0	26.635
SW-3, ST-2, N-1	0	0	0	31.021
SW-3, ST-3, N-0	0	0	0	26.012

Table 8 CALL_IN simulation details for dataset 1

CALL_OUT	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot

SW-0, ST-0, N-0	0	0	0	7345.851
SW-0, ST-0, N-1	0	0	0	46.261
SW-0, ST-1, N-0	0	0	0	32.274
SW-0, ST-1, N-1	0	0	0	44.734
SW-0, ST-2, N-0	0	0	0	32.830
SW-0, ST-2, N-1	0	0	0	43.278
SW-0, ST-3, N-0	0	0	0	31.805
SW-0, ST-3, N-1	0	0	0	42.975
SW-1, ST-0, N-0	0	0	0	34.315
SW-1, ST-0, N-1	0	0	0	45.191
SW-1, ST-1, N-0	0	0	0	30.606
SW-1, ST-1, N-1	0	0	0	43.012
SW-1, ST-2, N-0	0	0	0	31.477
SW-1, ST-2, N-1	0	0	0	43.705
SW-1, ST-3, N-0	0	0	0	32.307
SW-1, ST-3, N-1	0	0	0	43.141
SW-2, ST-0, N-0	0	0	0	30.909
SW-2, ST-0, N-1	0	0	0	42.448
SW-2, ST-1, N-0	0	0	0	32.942
SW-2, ST-1, N-1	0	0	0	43.232
SW-2, ST-2, N-0	0	0	0	32.788
SW-2, ST-2, N-1	0	0	0	43.477
SW-2, ST-3, N-0	0	0	0	31.610
SW-2, ST-3, N-1	0	0	0	43.129
SW-3, ST-0, N-0	0	0	0	32.776
SW-3, ST-0, N-1	0	0	0	42.253
SW-3, ST-1, N-0	0	0	0	32.714
SW-3, ST-1, N-1	0	0	0	43.461
SW-3, ST-2, N-0	0	0	0	31.062
SW-3, ST-2, N-1	0	0	0	42.846
SW-3, ST-3, N-0	0	0	0	30.166

Table 9 CALL_OUT simulation details for dataset 1

SMS_OUT	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	0	0	0	30.162
SW-0, ST-0, N-1	0	0	0	30.689
SW-0, ST-1, N-0	0	0	0	29.058
SW-0, ST-1, N-1	0	0	0	31.100
SW-0, ST-2, N-0	0	0	0	26.573

SW-0, ST-2, N-1	0	0	0	28.730
SW-0, ST-3, N-0	0	0	0	26.730
SW-0, ST-3, N-1	0	0	0	30.900
SW-1, ST-0, N-0	0	0	0	27.137
SW-1, ST-0, N-1	0	0	0	28.983
SW-1, ST-1, N-0	0	0	0	26.398
SW-1, ST-1, N-1	0	0	0	29.581
SW-1, ST-2, N-0	0	0	0	27.440
SW-1, ST-2, N-1	0	0	0	29.660
SW-1, ST-3, N-0	0	0	0	27.784
SW-1, ST-3, N-1	0	0	0	29.104
SW-2, ST-0, N-0	0	0	0	27.627
SW-2, ST-0, N-1	0	0	0	30.780
SW-2, ST-1, N-0	0	0	0	27.846
SW-2, ST-1, N-1	0	0	0	29.610
SW-2, ST-2, N-0	0	0	0	27.058
SW-2, ST-2, N-1	0	0	0	29.909
SW-2, ST-3, N-0	0	0	0	27.017
SW-2, ST-3, N-1	0	0	0	29.963
SW-3, ST-0, N-0	0	0	0	27.295
SW-3, ST-0, N-1	0	0	0	29.573
SW-3, ST-1, N-0	0	0	0	26.660
SW-3, ST-1, N-1	0	0	0	30.585
SW-3, ST-2, N-0	0	0	0	27.730
SW-3, ST-2, N-1	0	0	0	29.174
SW-3, ST-3, N-0	0	0	0	26.685

Table 10 SMS_OUT simulation details for dataset 1

CPU	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, N-0	4	1	0	48.485
SW-0, N-1	4	1	0	136.212
SW-1, N-0	4	1	0	45.261
SW-1, N-1	4	1	0	131.112
SW-2, N-0	2	0	1	43.693
SW-2, N-1	2	0	1	134.739
SW-3, N-1	0	0	3	129.751
SW-3, N-0	0	0	3	44.867

Table 11 CPU simulation details for dataset 1

SMS_IN	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	35	33	0	9014.863
SW-0, ST-0, N-1	2	0	0	65.714
SW-0, ST-1, N-0	0	0	2	39.826
SW-0, ST-1, N-1	2	0	0	64.855
SW-0, ST-2, N-0	0	0	2	37.880
SW-0, ST-2, N-1	2	0	0	63.747
SW-0, ST-3, N-0	0	0	2	38.124
SW-0, ST-3, N-1	2	0	0	63.581
SW-1, ST-0, N-0	0	0	2	39.369
SW-1, ST-0, N-1	2	0	0	63.515
SW-1, ST-1, N-0	0	0	2	37.942
SW-1, ST-1, N-1	2	0	0	64.805
SW-1, ST-2, N-0	0	0	2	39.046
SW-1, ST-2, N-1	2	0	0	63.689
SW-1, ST-3, N-0	0	0	2	39.975
SW-1, ST-3, N-1	2	0	0	64.340
SW-2, ST-0, N-0	0	0	2	38.734
SW-2, ST-0, N-1	1	0	1	64.647
SW-2, ST-1, N-0	0	0	2	39.149
SW-2, ST-1, N-1	1	0	1	62.643
SW-2, ST-2, N-0	0	0	2	38.195
SW-2, ST-2, N-1	1	0	1	63.759
SW-2, ST-3, N-0	0	0	2	38.967
SW-2, ST-3, N-1	1	0	1	63.705
SW-3, ST-0, N-0	0	0	2	38.842
SW-3, ST-0, N-1	0	0	2	64.340
SW-3, ST-1, N-0	0	0	2	38.079
SW-3, ST-1, N-1	0	0	2	65.158
SW-3, ST-2, N-0	0	0	2	38.523
SW-3, ST-2, N-1	0	0	2	63.490
SW-3, ST-3, N-0	0	0	2	38.050

Table 12 SMS_IN simulation details for dataset 1

CALL_IN	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	10	6	0	33.462
SW-0, ST-0, N-1	8	4	0	43.663

SW-0, ST-1, N-0	3	0	1	31.529
SW-0, ST-1, N-1	3	0	1	43.720
SW-0, ST-2, N-0	0	0	4	31.342
SW-0, ST-2, N-1	0	0	4	44.451
SW-0, ST-3, N-0	0	0	4	31.099
SW-0, ST-3, N-1	0	0	4	44.052
SW-1, ST-0, N-0	5	1	0	30.815
SW-1, ST-0, N-1	8	4	0	42.540
SW-1, ST-1, N-0	3	0	1	31.335
SW-1, ST-1, N-1	3	0	1	43.102
SW-1, ST-2, N-0	0	0	4	31.135
SW-1, ST-2, N-1	0	0	4	42.700
SW-1, ST-3, N-0	0	0	4	31.082
SW-1, ST-3, N-1	0	0	4	43.410
SW-2, ST-0, N-0	5	1	0	31.619
SW-2, ST-0, N-1	8	4	0	43.213
SW-2, ST-1, N-0	3	0	1	31.105
SW-2, ST-1, N-1	3	0	1	43.599
SW-2, ST-2, N-0	0	0	4	30.877
SW-2, ST-2, N-1	0	0	4	43.785
SW-2, ST-3, N-0	0	0	4	31.337
SW-2, ST-3, N-1	0	0	4	43.711
SW-3, ST-0, N-0	5	1	0	31.291
SW-3, ST-0, N-1	8	4	0	42.954
SW-3, ST-1, N-0	3	0	1	31.344
SW-3, ST-1, N-1	3	0	1	43.242
SW-3, ST-2, N-0	0	0	4	31.020
SW-3, ST-2, N-1	0	0	4	43.458
SW-3, ST-3, N-0	0	0	4	31.067

Table 13 CALL_IN simulation details for dataset 2

CALL_OUT	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	2	1	0	28.166
SW-0, ST-0, N-1	1	0	0	26.592
SW-0, ST-1, N-0	1	0	0	25.646
SW-0, ST-1, N-1	1	0	0	26.708
SW-0, ST-2, N-0	1	0	0	25.796
SW-0, ST-2, N-1	1	0	0	26.820
SW-0, ST-3, N-0	1	0	0	26.144

SW-0, ST-3, N-1	1	0	0	26.732
SW-1, ST-0, N-0	1	0	0	25.590
SW-1, ST-0, N-1	1	0	0	26.962
SW-1, ST-1, N-0	1	0	0	27.103
SW-1, ST-1, N-1	1	0	0	26.969
SW-1, ST-2, N-0	1	0	0	25.838
SW-1, ST-2, N-1	1	0	0	26.785
SW-1, ST-3, N-0	1	0	0	26.174
SW-1, ST-3, N-1	1	0	0	27.003
SW-2, ST-0, N-0	1	0	0	26.175
SW-2, ST-0, N-1	1	0	0	26.541
SW-2, ST-1, N-0	1	0	0	25.333
SW-2, ST-1, N-1	1	0	0	26.508
SW-2, ST-2, N-0	1	0	0	26.143
SW-2, ST-2, N-1	1	0	0	26.746
SW-2, ST-3, N-0	1	0	0	25.533
SW-2, ST-3, N-1	1	0	0	26.685
SW-3, ST-0, N-0	1	0	0	25.389
SW-3, ST-0, N-1	1	0	0	26.461
SW-3, ST-1, N-0	1	0	0	25.803
SW-3, ST-1, N-1	1	0	0	25.227
SW-3, ST-2, N-0	1	0	0	24.544
SW-3, ST-2, N-1	1	0	0	26.448
SW-3, ST-3, N-0	1	0	0	24.482

Table 14 CALL_OUT simulation details for dataset 2

CPU	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, N-0	199	150	0	114.886
SW-0, N-1	111	87	0	126.701
SW-0, N-0	84	50	0	47.551
SW-1, N-0	84	50	0	47.567
SW-2, N-0	61	40	0	47.026
SW-2, N-1	80	55	0	129.979
SW-3, N-1	31	6	2	133.060
SW-3, N-0	19	2	7	47.453

Table 15 CPU simulation details for dataset 2

SMS_IN	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	38	16	0	39.006
SW-0, ST-0, N-1	30	3	0	48.764
SW-0, ST-1, N-0	22	3	0	34.851
SW-0, ST-1, N-1	22	3	0	49.029
SW-0, ST-2, N-0	14	2	1	34.041
SW-0, ST-2, N-1	14	2	1	50.209
SW-0, ST-3, N-0	10	0	3	34.497
SW-0, ST-3, N-1	10	0	3	50.312
SW-1, ST-0, N-0	29	16	0	33.509
SW-1, ST-0, N-1	30	17	0	47.171
SW-1, ST-1, N-0	22	10	0	33.106
SW-1, ST-1, N-1	22	10	0	47.573
SW-1, ST-2, N-0	14	2	2	33.000
SW-1, ST-2, N-1	14	2	2	49.370
SW-1, ST-3, N-0	10	0	8	32.942
SW-1, ST-3, N-1	10	0	8	49.324
SW-2, ST-0, N-0	28	16	0	32.861
SW-2, ST-0, N-1	29	6	0	47.107
SW-2, ST-1, N-0	21	3	1	33.177
SW-2, ST-1, N-1	21	3	1	48.003
SW-2, ST-2, N-0	13	1	2	32.911
SW-2, ST-2, N-1	13	1	2	48.788
SW-2, ST-3, N-0	9	0	4	32.935
SW-2, ST-3, N-1	9	0	4	49.416
SW-3, ST-0, N-0	26	15	3	32.167
SW-3, ST-0, N-1	26	15	3	47.150
SW-3, ST-1, N-0	19	4	0	33.055
SW-3, ST-1, N-1	19	4	0	47.519
SW-3, ST-2, N-0	11	0	4	32.924
SW-3, ST-2, N-1	11	0	1	49.372
SW-3, ST-3, N-0	7	1	6	32.480

Table 16 SMS_IN simulation details for dataset 2

SMS_OUT	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	11	6	0	30.096
SW-0, ST-0, N-1	5	1	0	31.815
SW-0, ST-1, N-0	5	1	0	27.880

SW-0, ST-1, N-1	5	1	0	31.527
SW-0, ST-2, N-0	5	1	0	27.812
SW-0, ST-2, N-1	5	1	2	31.306
SW-0, ST-3, N-0	2	0	2	27.459
SW-0, ST-3, N-1	2	0	2	32.073
SW-1, ST-0, N-0	5	1	1	27.664
SW-1, ST-0, N-1	5	0	1	31.402
SW-1, ST-1, N-0	5	2	1	27.583
SW-1, ST-1, N-1	5	2	1	31.269
SW-1, ST-2, N-0	5	2	1	27.157
SW-1, ST-2, N-1	5	1	1	30.921
SW-1, ST-3, N-0	2	0	2	27.637
SW-1, ST-3, N-1	2	0	2	31.281
SW-2, ST-0, N-0	5	1	0	27.388
SW-2, ST-0, N-1	5	1	0	31.475
SW-2, ST-1, N-0	5	1	0	27.771
SW-2, ST-1, N-1	5	2	0	31.209
SW-2, ST-2, N-0	5	1	0	27.367
SW-2, ST-2, N-1	5	1	0	29.427
SW-2, ST-3, N-0	2	0	2	26.047
SW-2, ST-3, N-1	2	0	2	29.726
SW-3, ST-0, N-0	4	0	2	26.065
SW-3, ST-0, N-1	4	0	1	29.761
SW-3, ST-1, N-0	4	0	2	25.644
SW-3, ST-1, N-1	4	0	2	29.699
SW-3, ST-2, N-0	4	0	2	26.021
SW-3, ST-2, N-1	4	0	2	29.345
SW-3, ST-3, N-0	1	0	0	26.425

Table 17 SMS_OUT simulation details for dataset 2

CALL_IN	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	12	12	0	30.805
SW-0, ST-0, N-1	5	5	0	32.968
SW-0, ST-1, N-0	2	2	0	27.305
SW-0, ST-1, N-1	2	2	0	31.965
SW-0, ST-2, N-0	0	0	0	26.999
SW-0, ST-2, N-1	0	0	0	31.741
SW-0, ST-3, N-0	0	0	0	27.175
SW-0, ST-3, N-1	0	0	0	32.221

SW-1, ST-0, N-0	6	6	0	26.834
SW-1, ST-0, N-1	5	5	0	31.441
SW-1, ST-1, N-0	2	2	0	27.016
SW-1, ST-1, N-1	2	2	0	31.857
SW-1, ST-2, N-0	0	0	0	27.112
SW-1, ST-2, N-1	0	0	0	31.966
SW-1, ST-3, N-0	0	0	0	27.153
SW-1, ST-3, N-1	0	0	0	31.362
SW-2, ST-0, N-0	6	6	0	26.599
SW-2, ST-0, N-1	5	5	0	31.325
SW-2, ST-1, N-0	2	2	0	26.705
SW-2, ST-1, N-1	2	2	0	31.390
SW-2, ST-2, N-0	0	0	0	26.736
SW-2, ST-2, N-1	0	0	0	32.094
SW-2, ST-3, N-0	0	0	0	26.740
SW-2, ST-3, N-1	0	0	0	31.483
SW-3, ST-0, N-0	5	5	0	26.785
SW-3, ST-0, N-1	5	5	0	30.510
SW-3, ST-1, N-0	2	2	0	26.746
SW-3, ST-1, N-1	2	2	0	31.039
SW-3, ST-2, N-0	0	0	0	26.730
SW-3, ST-2, N-1	0	0	0	31.383
SW-3, ST-3, N-0	0	0	0	26.537

Table 18 CALL_IN simulation details for dataset 3

CALL_OUT	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	33	33	0	29.087
SW-0, ST-0, N-1	14	14	0	44.383
SW-0, ST-1, N-0	7	7	0	31.560
SW-0, ST-1, N-1	7	7	0	45.144
SW-0, ST-2, N-0	3	3	0	31.326
SW-0, ST-2, N-1	3	3	0	45.339
SW-0, ST-3, N-0	0	0	0	31.063
SW-0, ST-3, N-1	0	0	0	44.392
SW-1, ST-0, N-0	12	12	0	30.903
SW-1, ST-0, N-1	14	14	0	43.553
SW-1, ST-1, N-0	7	7	0	31.169
SW-1, ST-1, N-1	7	7	0	42.105
SW-1, ST-2, N-0	3	3	0	29.898

SW-1, ST-2, N-1	3	3	0	42.295
SW-1, ST-3, N-0	0	0	0	30.177
SW-1, ST-3, N-1	0	0	0	42.919
SW-2, ST-0, N-0	12	12	0	30.174
SW-2, ST-0, N-1	14	14	0	42.470
SW-2, ST-1, N-0	7	7	0	30.770
SW-2, ST-1, N-1	7	7	0	43.083
SW-2, ST-2, N-0	3	3	0	30.809
SW-2, ST-2, N-1	3	3	0	43.883
SW-2, ST-3, N-0	0	0	0	31.072
SW-2, ST-3, N-1	0	0	0	44.094
SW-3, ST-0, N-0	11	11	0	30.791
SW-3, ST-0, N-1	13	13	0	42.628
SW-3, ST-1, N-0	6	6	0	30.276
SW-3, ST-1, N-1	6	6	0	42.956
SW-3, ST-2, N-0	3	3	0	29.911
SW-3, ST-2, N-1	3	3	0	42.846
SW-3, ST-3, N-0	0	0	0	30.008

Table 19 CALL_OUT simulation details for dataset 3

CPU	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, N-0	263	N/A	N/A	65.573
SW-0, N-1	313	N/A	N/A	157.235
SW-1, N-0	263	N/A	N/A	62.416
SW-1, N-1	313	N/A	N/A	150.444
SW-2, N-0	124	N/A	N/A	62.708
SW-2, N-1	165	N/A	N/A	163.355
SW-3, N-0	74	N/A	N/A	61.176
SW-3, N-1	103	N/A	N/A	165.823

Table 20 CPU simulation details for dataset 3

SMS_IN	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	99	99	0	32.373
SW-0, ST-0, N-1	81	81	0	38.547
SW-0, ST-1, N-0	62	62	0	31.542
SW-0, ST-1, N-1	65	65	0	39.382
SW-0, ST-2, N-0	31	31	0	32.080
SW-0, ST-2, N-1	32	32	0	42.350

SW-0, ST-3, N-0	23	23	0	31.679
SW-0, ST-3, N-1	24	24	0	43.334
SW-1, ST-0, N-0	78	78	0	31.326
SW-1, ST-0, N-1	81	81	0	38.376
SW-1, ST-1, N-0	62	62	0	31.926
SW-1, ST-1, N-1	65	65	0	39.465
SW-1, ST-2, N-0	31	31	0	31.952
SW-1, ST-2, N-1	32	32	0	42.146
SW-1, ST-3, N-0	23	23	0	31.434
SW-1, ST-3, N-1	24	24	0	42.733
SW-2, ST-0, N-0	78	78	0	31.285
SW-2, ST-0, N-1	81	81	0	37.423
SW-2, ST-1, N-0	62	62	0	31.092
SW-2, ST-1, N-1	65	65	0	39.096
SW-2, ST-2, N-0	31	31	0	31.294
SW-2, ST-2, N-1	32	32	0	41.223
SW-2, ST-3, N-0	23	23	0	31.175
SW-2, ST-3, N-1	24	24	0	42.761
SW-3, ST-0, N-0	74	74	0	30.977
SW-3, ST-0, N-1	74	74	0	37.829
SW-3, ST-1, N-0	58	58	0	31.016
SW-3, ST-1, N-1	58	58	0	39.307
SW-3, ST-2, N-0	29	29	0	31.072
SW-3, ST-2, N-1	29	29	0	41.991
SW-3, ST-3, N-0	21	21	0	31.538

Table 21 SMS_IN simulation details for dataset 3

SMS_OUT	Anomaly Detected	False Positives	False Negatives	Avg runtime per slot
SW-0, ST-0, N-0	49	49	0	29.737
SW-0, ST-0, N-1	38	38	0	31.750
SW-0, ST-1, N-0	17	17	0	29.964
SW-0, ST-1, N-1	17	17	0	34.355
SW-0, ST-2, N-0	9	9	0	29.120
SW-0, ST-2, N-1	9	9	0	35.353
SW-0, ST-3, N-0	4	4	0	29.135
SW-0, ST-3, N-1	4	4	0	35.390
SW-1, ST-0, N-0	37	37	0	29.777
SW-1, ST-0, N-1	38	38	0	32.154
SW-1, ST-1, N-0	17	17	0	28.792

SW-1, ST-1, N-1	17	17	0	34.470
SW-1, ST-2, N-0	9	9	0	29.069
SW-1, ST-2, N-1	9	9	0	34.938
SW-1, ST-3, N-0	4	4	0	29.499
SW-1, ST-3, N-1	4	4	0	34.503
SW-2, ST-0, N-0	37	37	0	28.510
SW-2, ST-0, N-1	38	38	0	31.303
SW-2, ST-1, N-0	17	17	0	28.528
SW-2, ST-1, N-1	17	17	0	33.541
SW-2, ST-2, N-0	9	9	0	28.453
SW-2, ST-2, N-1	9	9	0	34.179
SW-2, ST-3, N-0	4	4	0	28.878
SW-2, ST-3, N-1	4	4	0	34.633
SW-3, ST-0, N-0	36	36	0	28.656
SW-3, ST-0, N-1	37	37	0	31.334
SW-3, ST-1, N-0	16	16	0	28.060
SW-3, ST-1, N-1	16	16	0	32.499
SW-3, ST-2, N-0	9	9	0	27.336
SW-3, ST-2, N-1	9	9	0	33.498
SW-3, ST-3, N-0	4	4	0	27.858

Table 22 SMS_OUT simulation details for dataset 3