Detection of Anomalous Behavior of Wireless Devices using Power Signal and Changepoint Detection Theory.

by

Ricardo Alejandro Manzano

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Anomaly detection has been applied in different fields of science and engineering over many years to recognize inconsistent behavior, which can affect the regular operation of devices, machines, and even organisms. The main goal of the research described in this thesis is to extract the meaningful features of an object's characteristics that allow researchers recognize such malicious behavior.

Specifically, this work is focused on identifying malicious behavior in Android smartphones caused by code running on it. In general, extraneous activities can affect different parameters of such devices such as network traffic, CPU usage, hardware and software resources. Therefore, it is possible to use these parameters to unveil malicious activities. Using only one parameter can not guarantee an accurate model since a parameter may be modified by cybercriminals to act as a benign application. In contrast, using many parameters can produce excessive usage of smartphone's resources, or/and it can affect the time of detection of a proposed methodology. Considering that malicious activities are injected through the software applications that manage the usage of all hardware components, a smartphone's overall power consumption is a better choice for detecting malicious behavior. This metric is considered critical for anomaly analysis because it summarizes the impact of all hardware components' power consumption. Using only one metric is guaranteed to be efficient and accurate methodology for detecting malware on Android smartphones.

This thesis analyzes the accuracy of two methodologies that are evaluated with emulated and real malware. It is necessary to highlight that the detection of real malware can be a challenging task because malicious activities can be triggered only if a user executes the correct combination of actions on the application. For this reason, in the present work, this drawback is solved by automating the user inputs with Android Debug Bridge (ADB) commands and Droidbot. With this automation tool, it is highly likely that malicious behavior can act, leaving a fingerprint in the power consumption.

It should be noted that power consumption consist of time-series data that can be considered non-stationary signals due to changes in statistical parameters such as mean and variance over time. Therefore, the present work approaches the problem by analyzing each signal as a stochastic, using Changepoint detection theory to extract features from the time series. Finally, these features become the input of different machine learning classifiers used to differentiate non-malicious from malicious applications. Furthermore, the efficiency of each methodology is assessed in terms of the time of detection.

Acknowledgements

I would like to thank Dr. Kshirasagar Naik and Abdurhman Albasir who have been my mentors during this two years of research. In addition, I would like to thank Dr. Ivcovic and Dr. Gebotys who have reviewed my thesis and provided me with useful feedback.

Dedication

Firstly, this thesis is dedicated to God Yahweh. Secondly, to my parents Washington Manzano and Sylvia Sanchez who have been my support all my life. Without my parents this cannot be possible. My siblings Gabriela, Paul, Felix, and my nephew Julian who have given support and happiness. To my grandparents Raul and Mariana who are always in my thoughts. To my uncles Danilo, Ramiro, Fernando S., Fernando A., my aunts Nancy, Mayra, Susana, Consuelo, and cousins who have always been there to help me. To my friends Renato, Carlos M., and Carlos C.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Wearable and portable computing devices have become the appliances with the most significant penetration around the world in the last decade, due to power of computing, capacity of connection, capacity of storage, and size of device. According to GSMA Mobile Economy [47], in 2019, 5.13 billion users subscribed to mobile services, and there will be 5.8 billion in 2025. GSMA estimates an annual growth of 1.9%. This statistic takes into consideration smartphones, tablets, and cellular-enabled IoT (Internet of Things) devices. In 2019, Statista has indicated that 3.3 billion corresponds to smartphone users representing 64% of the total number of users subscribing to mobile services worldwide. In fact, mobile devices are clearly an essential tool that people use to work, study, communicate, and entertain.

Due to its different usages and direct interaction with users, mobile devices have also turned into an excellent source of information, that has unfortunately attracted the attention of cybercriminals. These law-breaking developers embed malicious code in applications (Apps) to steal user information, generate undue charges to the users, or provoke denial of service in networks.

Cybercriminals can affect users by stealing user's location, smartphone's IMEI, list of contacts, or list of installed Apps to sell this information in the black market [41]. To illustrate, Zitmo is an Application (App) that captures SMS from banks to provoke Phishing [51]. Geimini is an App that performs calls and sends messages to premium numbers provoking undue charges [18]. Other cybercriminals have designed Apps to affect companies' infrastructures through Denial of Service [93] [83].

The attacks described above generally have been addressed to Android users because this operating system has monopolized the market share in the world, with 71% of the total mobile devices, according to Statcounter in 2019 [91].

To counter these malicious activities, many researchers and companies have developed many methodologies analyzing static, dynamic, and hybrid characteristics of the Apps. Static analysis methodologies inspect specific patterns in the source code of each App [62][99][21]. Dynamic analysis evaluates different parameters such as network traffic, power consumption, or CPU usage to identify malicious activities [19][103][34][61][67]. Finally, hybrid analysis combines static and dynamic approaches to detect malware in an efficient way [22].

This thesis focuses on the dynamic analysis of overall smartphone's power consumption to identify malicious behavior. This parameter is a better choice for detecting malicious behavior because it summarizes the power consumption of all hardware components. Also, the overall power consumption of a malicious App is more difficult for attackers to modify to act as a benign App, because it depends on all of a device's components' power consumption. Furthermore, measuring and storing only one smartphone's parameter is less resource exhausting in terms of memory. In addition, analyzing only one parameter takes less time to detect malicious behavior than analyzing many parameters.

The proposed methodologies measure and store the average power consumption using an external device to avoid affecting the variable measured. Furthermore, the methodologies analyze the measured signals in an external device (off-device detection) because a smartphone is not adequate for handling large amount of data.

This thesis explains two methodologies to detect malware in smartphones using power consumption time-series measurement. The first methodology uses parametric Changepoint. On the other hand, the second methodology utilizes non-parametric Changepoint technique. Changepoint detection theory has been adopted as the signal behaves stochastically. The parametric technique considers different assumptions to fit the model. Conversely, non-parametric technique does not use assumptions, and the features are extracted with density-ratio estimation. Both methods identify collective anomalies in the signal, represented by plausible changes in it [29].

Parametric technique groups datapoints in intervals, maximizing the difference of probability distributions among them. Non-parametric technique ranks every change in two consecutive segments. Each segment is composed of overlapping windows with datasamples. The plausibility of a change is measured without knowing the probability distributions of consecutive segments. The only thing known is the ratio among them.

The features extracted using these techniques have been used to train and test different classifiers such as Support Vector Machine Linear, Logistic Regression, and Naive Bayes, to determine if the smartphone has been running malicious behavior. The classifiers have been chosen because these are deterministic; thus, every time that a classifier is run with

the same input, it will return the same output. In addition, Support Vector Machine and Logistic Regression have been selected because these classifiers are linear. Therefore, they provide better generalization and avoiding overfitting. Finally, Leave One Out Cross Validation has been applied to obtain realistic and precise results of F1-measure, which evaluates all the data points of the dataset.

Both methodologies have been evaluated with emulated and real malware. Emulated malware is used to compare the accuracy of the proposed methodologies with other power-based methodologies in terms of accuracy. Moreover, both methodologies are tested with real malware to verify its validity in real scenarios. Real malware Apps have been chosen from Drebin dataset [21]. Drebin dataset is composed of $\sim$ 5560 applications with real malware of different families such as: KungFu, Plangton, BaseBrid, Opfake, and others. The data collection was done by MobileSandbox Project.

The detection of real malware can be a challenging task considering that Malicious Apps do not trigger malicious behavior without a proper combination of user's inputs. In research, it is manually impossible to look for the combination of actions that trigger malicious behavior [57]. Therefore, the present work tackles this problem using a test input generator to automate the actions in the App with Droidbot [72] and ADB commands.

1.1 Motivation

The increasing number of smartphones and IoT devices, the impact of attacks in the economy of individuals and companies, the increasing amount of mobile malware, and the fields in which cyber-attacks have been released such as smartphones and IoT devices in recent years are the main motivations for the research described in this thesis.

Firstly, the number of mobile subscribers around the world, according to GSMA, shows an important increase from 5.13 billion in 2019 to 5.8 billion in 2025, representing an annual growth of 1.9%, as can be observed in Fig. 1.1. The majority of these devices belong to smartphone users, representing 64% of the total mobile subscribers in 2019 with $\sim$ 3.3 billion, as reported by Statista. Figure 1.1 shows the trend predicted until 2025. Nonetheless, as seen in Fig. 1.2, mobile subscribers are only a small proportion compared with the surge in IoT devices. In 2025, IoT devices will reach 21.5 billion surpassing the 5.8 billion mobile subscribers 3.7 times. In the present work, this statistic is important because the proposed methodologies will evaluate different inputs to prove that these methodologies are data-agnostic. Data-agnostic means that the methodologies

can use different parameters as inputs demonstrating that they can be applied to recognize anomalous behavior in other devices.

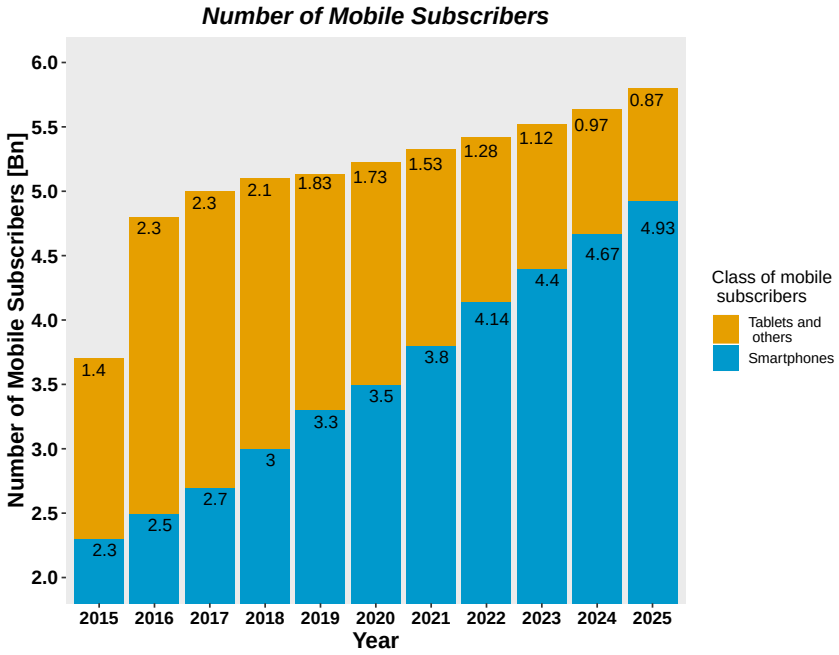


Figure 1.1: Evolution of non-IoT devices from 2015 to 2025 [47].

Secondly, attacks have had an increasingly harmful impact on the economy of individuals and companies. Accenture [4] indicates that Ransomware is the attack that produces the most economic loss, increasing it from $532914 in 2017 to $645920 in 2018, representing 17.49% on average. Another worrying statistic is related to attacks executed by authorized attackers in the networks, called malicious insiders. According to the same source of information, this type of attack is second in generating economic losses, and increased its percentage by 12% from $1415217 to $1621075.

Lastly, McAfee has stated that the number of mobile malware products has increased exponentially. In the last quarter of 2016, the total number of mobile malware reached 15 million. However, in the same quarter of 2017, it surpassed 22 million. In 2018, it exceeded 30 million.

Many authors have demonstrated how threats have expanded to IoT devices. To illustrate, Tierney and Munro [1] proved that ransomware could infect a smart thermostat. Anthi et al. [16] demonstrated that smart home devices such as Amazon Echo Dot, Belkin

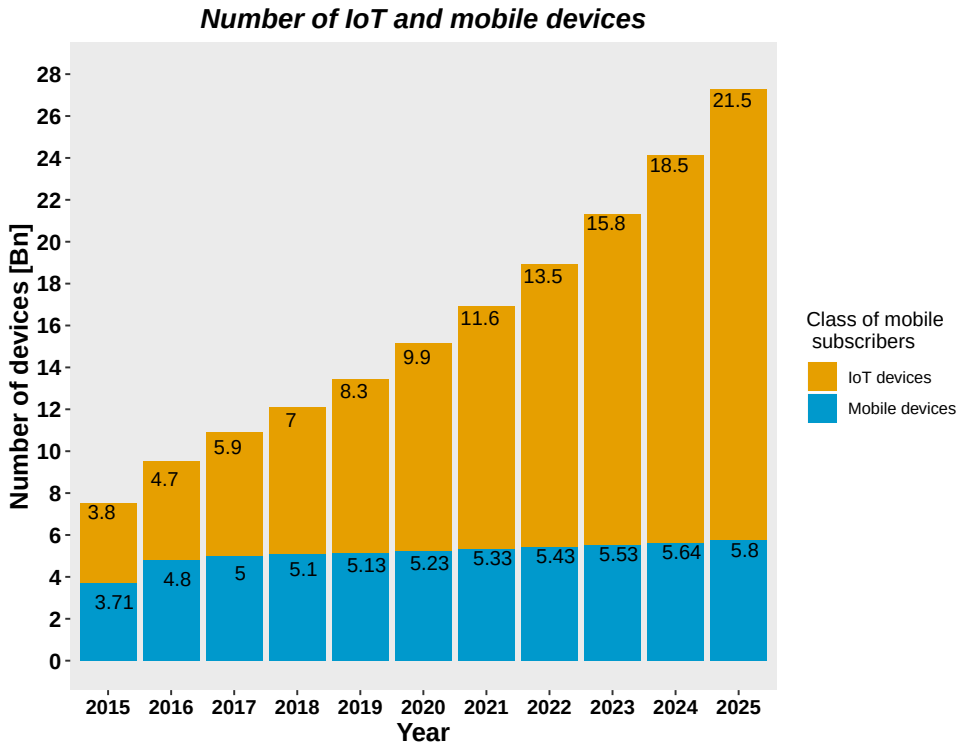


Figure 1.2: Evolution of IoT and non-IoT devices from 2015 to 2025

Netcam, TP-link NC200 camera, Lifx smart lamp, Tp-link smart plug, Samsung smart things hub, Hive Hub, and Apple tv could be prone to reconnaissance, DoS, Distributed Denial of service (DDoS), Man in the middle, Replay, and Spoofing attacks. Albrecht and Mcintyre in [9] described how cybercriminals could control smart IP baby monitors.

Considering threats in wearables devices, Klonoff et al. [64] described how an attack on a wireless glucose monitor could be dangerous for patients in terms of confidentiality and integrity. The author showed how an attacker can retrieve information about a patient, or inject a malicious code to control the device and activate the injection of insulin, which can be deadly for the patient. In reference [69], the authors highlighted how Bluetooth low energy (BLE) could suffer from sniffing attacks. The authors demonstrated how three wearable health bands are hacked, evading authentication protocols and exposing user information.

Taking into account the facts stated above, the main goal of this thesis research is to develop two proof-of-concept methodologies for recognizing anomalous behavior in Android

smartphones using the power consumption. These methodologies will be proved with inputs to demonstrate that they are data agnostic. By proving this fact, we can conclude that these methodologies can be applied to recognize malicious code in other IoT devices.

1.2 Problem statement

In normal conditions, every device must act or behave in the way that the manufacturer designed it to. However, several factors can change the normal behavior of hardware and software components, causing anomalous behavior.

Definition 1. *Anomalous behavior is considered as any change or deviation regardless whether it is small-scale or sizable in a normal data pattern [29] seen from data analysis.*

The main factors that affect normal behavior are hardware and software aging, hardware trojan, firmware updates, and malicious code embedded in new Apps, which are explained as follows.

All electronic devices suffer from hardware aging due to the degradation of performance in transistors. Dogan et al. [37] demonstrated that deviation in CMOS devices could occur mainly due to Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI). These factors affect the transistor switching speed degrading the device's performance.

Software aging, according to references [53][44], is caused by registers' overflow, accumulation of errors in memory, and modification of software by other tools. This kind of anomalous behavior can cause a crash of the system, performance degradation, or data inconsistency.

A Hardware Trojan is related to a device modification in the manufacturing stage to produce attacks. The authors in reference [39] describe how a machine or device can be prone to attacks if one of the parts is replaced by trojan hardware.

Firmware update attacks occur when a user wants to update or change the firmware of a device to activate new functionalities, solve a security issue, fix a malfunctioning operating system, or another cause. Generally, cybercriminals provide firmware with malicious code to steal user information, produce undue charges, or generate a denial of service of the device or network [71][24][74][85].

Finally, smartphones have direct interaction with users. Thus, they can suffer infection due to the installation of infected Apps.

This thesis research focuses on recognizing malicious behavior injected due to malicious Apps installed by the user or other sources in a smartphone using anomaly detection methodologies. The methodologies proposed must recognize emulated and real malware.

1.3 Solution Strategy and Contributions

The general approach to detecting anomalous behavior is shown in Fig. 1.3. This model has 3 main stages: acquisition of the signal, analysis of the signal, and making a decision.

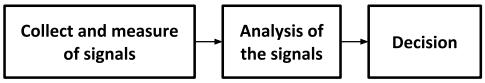


Figure 1.3: General model of generating an anomaly-detection methodology

The first stage in Fig. 1.3 is related to the selection of a signal with different nature to recognize anomalous behavior, and the resource necessary to measure and store this signal. IoT devices are composed of hardware and/or software components such as sensing, actuator, process, power, and communication systems, as shown in Fig. 1.4. These components can be used as sources or metrics to detect anomalous behavior. Most of the parameters described can be measured using an external device (off-device measurement) or using the resources of their own device (on-device measurement).

The second and third blocks of Fig. 1.3 are stages to analyze or evaluate whether the signal is normal or anomalous. This detection analysis can be done on-device or off-device.

Researches use the general model to propose methodologies that unveil malware in smartphones.

Many authors utilize on-device measurement and on-device detection methodologies. The difference among researchers' approaches is the nature of data used. To illustrate, in reference [97], the authors use permissions of an Android App. In reference [81], the researches use system calls.

Most authors use on-device measurement and off-device detection considering network behavior [48], CPU usage [38], and permissions [8].

Finally, some papers approach malware detection taking into account off-device measurement and off-device detection using network packets [94][33], or Radio-frequency emissions [86].

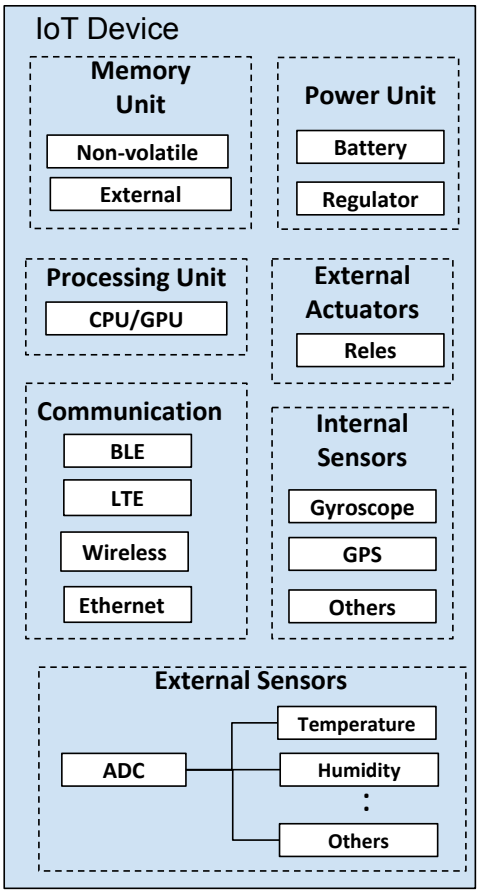


Figure 1.4: Components of an IoT device

It should be noted that most of the parameters used to unveil malware in the cited papers do not summarize the overall behavior of the entire device. For this reason, the power consumption of the entire device is used to discover anomalous behavior. This feature encapsulates the energy consumed by all of the hardware components of the device. Power consumption can be estimated from a smartphone in two ways:

- **On-device measurement**: The power consumption is measured with hardware and software performance counters or the status of the battery extracted from the Kernel of the device [101][105][42][63].
- **Off-device measurement**: The power consumption is measured with an external tool [61][56].

In this work, an external device measures and collects the overall power consumption to avoid affecting the variable with the measuring task. Additionally, an external device (off-device detection) analyses the signal due to the large amount of data to process.

To extract meaningful features from a stochastic time-series signal, Changepoint detection theory is used. The validity of this theory has been proven in other areas of research, guaranteeing its accuracy to detect anomalous behavior [49][40][84][35]. The Changepoint theory identifies collective anomalies in the signal represented by plausible changes in it. The features extracted using Changepoint detection are the inputs to different machine-learning classifiers.

As described in Sec. 1.2, the proposed work is focused on the detection of real and emulated malware. Generally, real malware remains dormant if the user's inputs do not follow the correct combination. To solve this problem, we employ an automation tool with Android Debug Bridge (ADB) commands and Droidbot. This tool mimics the actions executed by a user trying different combinations of buttons, links, intents to trigger malicious behavior.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 summarizes previously reported work and background knowledge in anomalous detection. Chapter 3 explains how emulated malware App is designed. Furthermore, this chapter describes the implementation of an automation tool to take power measurements of emulated and real malware. Chapter 4 explains parametric Changepoint detection. Chapter 5 explains non-parametric Changepoint detection. Chapter 6 presents the experiments and results. Finally, Chapter 7 describes the conclusions and future work.

Chapter 2

Background and Related Work

2.1 Introduction

The following chapter briefly introduces anomaly detection and energy-based anomaly detection. Additionally, it explains the challenges of anomaly detection. Finally, the literature review on malware detection in Android smartphones and IoT devices is reviewed.

2.2 Anomaly detection

Anomaly detection has been used for many years to recognize deviations of one or many datapoints from normal patterns or expected behavior [29] in different fields such as fraud, intrusion, hardware or software fault detection. Generally, anomalous behavior is represented as an outlier or group of outliers from a dataset concentrated in specific areas or clusters. Three classes of anomalies exist, according to Chandola et al. [29].

- **Point anomaly**: This abnormality occurs when only an individual anomaly point is diverted from a normal cluster. Generally, the analyzed dataset contains a large number of normal datapoints and only a few extraneous ones.
- **Contextual anomaly**: This kind of irregularity is similar to a point anomaly because an individual abnormal point is diverted from a normal dataset considering a specific context such as a time or position.

- **Collective anomalies**: This anomaly typo is different from point and contextual anomalies, given that the abnormal behavior occurs as a set of datapoints. In this case, only one anomalous point within the collection of datapoints cannot be considered abnormal.

This thesis analyzes collective anomalies using a power consumption time-series. According to Caviglione et al. [28], energy-based approaches are characterized depending on the availability of data as follows:

- **System-Based**: Normal behavior is characterized using power consumption measurements of the smartphone in a clean state. The power consumption can be taken of the entire system or specific hardware components such as a CPU, Wireless module, or other.
- **Application-Based**: Normal behavior is characterized using a unique process running on the smartphone without background processes. While the normal App is running, the power consumption of the entire system or specific hardware components is taken. For example, to characterize normal behavior of games, the power consumption is taken of each single App game running on the smartphone.
- **User-Based**: This energy-based approach uses the actions of a user to characterize normal or anomalous behavior.
- **Attacked-Based**: This type of energy-based solution uses normal and infected Apps to create a dataset. Thus, researches can fit models using supervised learning to identify malicious behavior.

This thesis utilized attacked-based approach because we have available malicious and benign Apps.

2.3 Challenges in detecting anomalous behavior

There are many challenges in detecting anomalous behavior in smartphones, for the following reasons:

- The overall power consumption of the smartphone is a combination of all hardware components' power consumption in the device taken at the same time. Therefore, the overall power consumption can be noisy and inaccurate to unveil malicious behavior if pre-processing and feature extraction stages are not applied with appropriate techniques.
- Boundaries between anomalous and benign classes are hard to define because overlapping or close data samples can exist. Thus, the methodology has to choose optimal classifiers that avoid overfitting and provide acceptable accuracy in terms of F1-measure.
- The scarcity of labelled power consumption measurements is another challenge. Generally, power consumption measurements must be taken manually. Thus, researchers can spend much time in this task. In addition, the power measurements are not consistent due to human interaction. In the present work, it is necessary to develop a tool to take power measurements automatically of real and emulated malware.
- Malware is evolving every second. Thus, defining a general model that recognizes all kinds of malware is difficult.

To address all the challenges highlighted, the present work selects, and develops strategies, techniques, and tools. Moving average filter and Changepoint extract features of a noisy time-series signal. SVM linear, logistic regression, and Naive Bayes classifiers are deterministic linear classifiers selected to avoid overfitting. An automated tool using Droidbot, and ADB commands generates sufficient datasamples to fit and validate the methodologies.

2.4 Related work

As described in Chap. 1, various authors have developed their methodologies considering three main factors: the nature of data, where the measurements are taken, and recognition is executed. In this section, each of the papers cited before is explained in detail.

First, we analyze papers related to on-device detection and on-device measurement.

Martinelli et al. in [81] recognize malware with a tool that evaluates static and dynamic characteristics of each App. The approach analyzes the frequency of opcodes in

		Detection													
		On-device		Off-device											
		Permissions	System calls, Administrator priviliges	Network behavior	Network packets	CPU, Battery, SYN packets	Battery, Position of the user	Battery, CPU usage, permissions	Battery, BTemp, network traffic	Power consumption	Electromagnetic emanations	Timing-based processor	Radio frecuency emmisions	File accesses, network accesses, data leaks, service start	Screen of states, XML files, resources, and classes.dex	Api calls, intents
Measurement	On-device	[97]	[81]	[48]		[38]	[36]	[8] [14]	[67]	[104] [23] [28]						
	Off-device			[5]	[33] [31] [94]					[55], [90] [3],[60] [27],[76]	[82]	[78]	[86]	[102]	[12]	[100]

Table 2.1: Related work

Apps, which is considered a static feature for determining if an App is malicious. If the App contains malicious code, the tool uninstalls the App. Otherwise, the tool runs a dynamic analysis to extract system call invocations, administrator-privilege abuses, and text messages. Using these parameters, the methodology determines if the App is malicious or benign. The methodology evaluates 9804 benign Apps and 2794 malicious Apps from Drebin, Genome Project, and Contagio datasets. The accuracy of detection is 99.7%, which is 1.7% greater than that with the Virus Total approach.

In reference [97], the authors determine if an App has malware with a hot set and a cold set distributed in a group of smartphones. The hot and cold sets have MD5 and regexp signatures from Apps. Each smartphone of a group has a hot set and cold set. All of the smartphones have a hot set that has the most important signatures to detect malware. In contrast, the less important features are distributed in all the devices as a cold set. When a new App is analyzed, signatures are extracted and tested against the hot set. If the tool detects malware, there is a flag to alert other users. Otherwise, if the hot set does not detect malware, the signatures are tested with the cold set. Since the cold set is distributed, if one device cannot recognize the malware, another device will recognize it. This technique is useful in host-based malware which is not resource-intensive.

Other authors have utilized on-device measurement and off-device detection as follows:

Guo et al. [48] analyze the network behavior of benign and malicious Apps to detect malware. This technique uses off-device detection because an external server analyzes the network behavior. The authors consider that malware can execute one or more of the following actions:

- Dissemination: Malware executes this action to propagate itself through the network using MMS, HTTP, FTP, email, or another way.
- Accessing malicious server: Malware carries this action to download a complete version of the malware from a server or execute command control of the device.
- Attacking: Malware performs these actions to steal user information, produce undue charges, or another action.

The authors in this approach extract network patterns of malicious Apps and group them according to the actions exposed before. Also, they obtain network patterns of benign Apps to create a white list. The detection procedure works as follows. In the beginning, the features of network behavior are compared with the white list. If the App shares network patterns with the white list, the signal is benign. Otherwise, the network features

are compared with malicious patterns depending on actions. If the App contains similar features as the three groups exposed before, the signal belongs to one of the groups. Finally, if the patterns do not match with any group, the classification model trains the new features with incremental self-learning method. The smartphone collects and stores network behavior. However, an external server extracts the features and classifies them.

Ali et al. [38] in 2014 used different mobile device resources such as CPU usage, memory, number of ICMP packets, and number of connections to detect malware. This approach collects the data on a device using a "Data Collector". The Data collector retrieves information of the mobile device resources within a pre-defined time interval. The mobile device sends the information collected to a server in the cloud. This server extracts patterns using a Gaussian mixture model to differentiate benign from malicious Apps. This approach tests three malware obtaining excellent results in terms of the ROC curve.

Dixon and Mishra in 2013 [36] used the power consumption of the battery and the geographical position of the device to recognize anomalous behavior in two Apps with emulated malware. The first App imitates real malware, sending SMS to premium numbers. The second App mimics another kind of malware that sends the geographical position of the user through SMS. In this approach, an App collects the power consumption each time that the user moves 1500 meters, or when the battery changes its level. An external server analyses the power consumption with an optimum cutoff value methodology.

Alam and Vuong in 2013 [8] used the battery, CPU usage, and permissions to recognize Apps with anomalous behavior. A mobile device measures and collects the parameters while a Monkey tool is emulating the user's input randomly and automatically. Formally, a Monkey tool is a program that performs random actions in a smartphone such as touches, clicks, calling system-level events, Wifi on-off, Bluetooth on-off, and others. A random forest classifier trains and tests the parameters extracted. The authors evaluate 1330 malicious Apps and 407 benign Apps, obtaining an accuracy of 99.9%.

Amos et al. in 2013, [14] proposed a technique that analyses Apps in cloud infrastructure using a 34-node cluster and one mobile device to detect malicious behavior. Essentially, this platform installs the App in each virtual or physical node. Each node opens the App and runs a Monkey tool, which mimics users' inputs to control the App. While the Monkey tool is executing random actions, features such as the battery, binder, memory, network, and permissions are collected each 5s. These parameters are collected with 432 benign and 1353 malicious Apps. Random Forest, Naive Bayes, Multilayer Perceptron, and Logistic Regression train and test these features reaching a maximum accuracy of 94.53%.

Kurniawan et al. [67] in 2015 used network traffic, battery percentage, and battery

temperature to unveil malware. They collect this information using a Logger App on the device. All the features collected by the Logger are transmitted to a server in the cloud. The server in the cloud trains and tests the following classifiers: Support Vector Machine (SVM), Random forest, Logistic Model Trees(LMT), and J48 Decision Trees. The dataset used in this approach is Genome Project. The authors highlight that the maximum accuracy is 85.6% with Random forest.

Zefferer et al. [104] in 2014 used Power tutor, which collects power consumption measurements on the device, to detect SMS spyware. Power tutor measures the power consumption of different hardware components such as CPU, GPS, Wifi components, and screen. The authors develop a methodology using Mel Frequency Cepstral Coefficients (MFCC) and Gaussian mixture models (GMM) to differentiate four kinds of emulated SMS spyware:

- SMS spyware 1: The spyware forwards an SMS message received to the inbox application of the mobile device.
- SMS spyware 2: The spyware intercepts an incoming SMS, and sends the last geographical location of the device to the same number.
- SMS spyware 3: The spyware intercepts an incoming SMS, and it replies with the current position.
- SMS spyware 4: The spyware delivers the SMS to the inbox application, but at the same time, it sends a copy of this SMS to another number.

They conclude that the patterns of power consumption are dissimilar enough to differentiate them. In addition, the authors validate this methodology with different categories of Apps such as games, internet, idle state, music and multimedia.

In 2018 Azmoodeh et al. [23] also used Power Tutor, which measures the power consumption on the device to recognize Ransomware. In this approach, they utilize a CPU's power consumption to differentiate normal Apps (Facebook, Chrome, Youtube, WhatsApp, Skype, Angrybirds, Maps, Music player, Twitter, Instagram, and Guardian) from six types of Ransomware. They take power traces for 5 minutes for each App. The methodology splits the signal using a window size. Each window is analyzed using Dynamic Time Warping (DTW). Finally, the features extracted are the input of different classifiers such as K-Nearest Neighbor (KNN), Neural Network, Random Forest and Support Vector Machine (SVM). The accuracy reached by the model is 95.5% using KNN with a window size of 7.5s.

In 2016, Caviglione et al. [28] detected attacks related to convert channels using the power consumption of the processes running on the smartphone. This kind of attack occurs when two or more malicious Apps exchange information exploiting different permissions assigned to them. To illustrate, if one App has permission to read phone numbers, and the other App has permission to use the network, both Apps can cooperate to transmit the user's phone numbers through the network. Generally, traditional static or dynamic approaches can fail to recognize these attacks. To prove the validity of the methodology, the authors collected power measurements of the idle state of the smartphone and different scenarios of convert channel attacks using Power Tutor with high-level and middle-level information. Their methodology adopts the power traces to train and test classifiers such as Neural Network and Decision Trees. To generate malicious power traces, a malicious App simulates different convert channels attacks such as type of intent, file lock, system load, volume settings, Unix socket discovery, file size, and memory load attacks. They achieved an accuracy of above 80% with this methodology.

Finally, methodologies that use off-device measurement and off-device detection are described.

Kim et al. [60] are pioneers in identifying emulated malware utilizing power consumption measured through an external device. The authors design and build a device using different low-cost components such as a current sensor, a capacitor, and a micro-controller. Although the device is small and not expensive, it can measure power traces with a high sample rate between 1ms to 50ms, which is sufficient to unveil malicious behavior. The methodology proposed extracts features with a moving average filter and data compression. Furthermore, the dataset trains and tests 11 emulated malicious Apps and eight benign Apps. The malicious Apps are Wifi faker, a dummy App to execute a hard computational task, a combination of the Wifi faker and the dummy program, four emulated mobile worms (Cabir, Mabir, CommW, Lasco), and an App to send a big packet through Wifi. The benign Apps are Windows Media Player, a data sender using Bluetooth and Wifi, and two users who explore a file in the memory of the mobile device. They conclude that the power signatures of Wifi faker, dummy App, Wifi Faker + dummy App are very different compared with the eight benign power signatures, reaching 100% of detection accuracy. Nonetheless, the lowest accuracy reported is 80%, in detecting CommW malware.

Nazari et al., in 2017, [82] developed a methodology for recognizing anomalous behavior in IoT devices, by measuring electromagnetic emanations of the smartphone's processor. They detect two attacks related to program injection into the device's code. The first attack is an injection of an empty shell-code outside of a loop, which can be considered the most common injection attack. The second attack is the injection of 8 instructions inside a loop. The methodology recognizes these scenarios transforming the time series data to the

frequency domain through Short-Term Fourier transform (STFT). This method generates windows that are characterized by a statistical distribution using a non-parametric method called Kolmogorov-Smirnov. The authors use a single-board Linux computer to execute the experiments, detecting both scenarios with an accuracy of $\sim 95\%$.

Lu et al., in 2017, [78] developed a non-intrusive technique to unveil malware in IoT devices using a timing strategy. Most microcontrollers of IoT devices have a trace port that provides useful information about the duration of execution events. In this work, the researches use this port to track the execution time to characterize signals. Lumped timing multi-ranges(LTMR), and sub-component timing intrinsic (ST) extract features from the signals, and hierarchical clustering classifies the signal as benign or malicious. The authors test their methodology in a Smart connected pacemaker, which is composed of a pacer, a sensor to monitor the heart, timers, and an interrupt controller. The controller is an essential component of the system because it senses the heart rate of a patient, and controls the pacer. Generally, these devices are configured by a physician to pace the heart. If there is an anomaly in the heart of the patient, the system sends an alert to the physician to configure it again. If malware is running in the controller, it can modify the cardiac activity log, or it can perform a fuzzing attack that changes the system function and calls. To detect these malicious activities, the authors propose to track the duration of an event with an additional microcontroller that is running LTMR, ST and hierarchical clustering techniques. The authors using this methodology recognize malware in the 12th event with 100 percent accuracy.

Liu et al. [76] in 2016 developed a power consumption side-channel strategy for detecting anomalous behavior in control flow execution applied to IoT microcontrollers. This strategy is called side-channel since the power consumption is measured in the VCC pin of the microcontroller through a resistor. The proposed methodology characterizes the sequence of instructions of a program using the Hidden Markov model (HMM) in the frequency domain. To test the accuracy of the model, the authors test insertion, deletion, and replacement on AES PROGRAM to prove a firmware modification attack. The results show an outstanding accuracy to detect modification of the sequence of a program, almost 99.93%.

In 2017, Li et al. [102] used Droidbot and Droidbox to efficiently discover malicious activities in Apps. Droidbot, a program written in Python, emulates human inputs such as touches, clicks, calls to intents, and other actions over a Test App. Droidbot is different from a Monkey tool because it uses a depth exploration of the Test App, taking into account static characteristics to create efficient test inputs triggering malicious activities faster. Droidbot runs in a computer connected to a mobile device through a wireless or wired connection using ADB (Android Debug Bridge). The authors use Droidbot to

emulate a user's inputs controlling a Test App, whereas Droibox measures file accesses, network accesses, service start, and data leak. The authors conclude that Droidbot can help to trigger sensitive behaviors of the Test App in a few seconds, which is useful in discovering malware efficiently.

Yerima et al., in [100], demonstrate the effectiveness of Droidbot in triggering malicious activities to detect malware in Apps. In this work, the authors compare the accuracy of detection using three tools to trigger malicious behavior: a Monkey Tool (Random-based), Droidbot (State-based), and Hybrid approach (Random + Droidbot). In their testbench, while Monkey, Droidbot, or Monkey+Droidbot is controlling a Test App, a server collects information about API calls and intents. Sequential Minimal Optimization (SMO), Naive Bayes (NB), Simple logistic (SL), Multilayer Perceptron (MLP), Partial Decision Trees (PART), Random Forest, and J48 Decision tree train and test both parameters. The datasets used to verify the accuracy of the methodology are 2444 malicious Apps from the Genome project and 2444 benign Apps from McAfee Labs. They conclude that the best accuracy to detect malware is obtained when Droidbot and Random forest algorithms are used, reaching 94.3%.

In reference [12], the authors used static and dynamic analysis to detect Ransomware. In the beginning, the test App is compared with a database of inspected Apps. If the App is on the database, it is classified as malicious. Otherwise, the App is analyzed using static analysis. In this stage, it extracts images of each screen of the App, XML layout files, resources, and classes.dex. A text extractor is used to retrieve string states from the files denoted before. In addition, it is used as an Image Extractor to obtain patterns from images. While static analysis is executed, dynamic analysis is done using Droidbot. As described in [100] and [102], Droidbot is a user's input generator. Whilst Droibot is controlling the test App; screens of the possible states of the App are extracted in a dynamic state. The images generated by Droidbot are also processed by the Image extractor. Image and string similarity measurements are obtained from all of the analyzed Apps. The authors used 850 ransomware Apps and 500 benign Apps to train the model, and they tested it with 100 samples. They gained an accuracy of 91% using this approach.

Aiolli et al. [5] in 2019 used the network traffic collected by Wireshark to differentiate anomalous actions in cryptocurrency Apps from the ten most used benign Apps from Play Store. The network traces describe the number of bytes transmitted and received during a specific time when Apps are used. The proposed methodology extracts features from these signals using two stages: Traffic bursification and Flows separation. After applying both stages, time-series signals are obtained, denoted as flows. Finally, the length, minimum, maximum, mean, median, mode, variance, skewness, kurtosis, and percentiles for each flow are trained and tested with SVM and RF classifiers. The authors conclude that it is

possible to recognize actions from bitcoin wallet Apps such as an open App, receive and send a bitcoin from benign Apps with an accuracy of 97.7%.

In 2017, Robin et al. [55] was able to identify emulated malware using power consumption measurements taken by an external device and Independent Component Analysis (ICA). They extract two independent components from reference signals (non-malicious App running in the foreground) and suspicious signals (reference signal + emulated malware running in the background). These two independent components are compared with the input signals using correlation. Finally, a threshold of the correlation value between malicious and benign signals is calculated. As a complementary study, the authors extracted the mean and variance of each signal as features to train and test classifiers using SVM. The results presented show a maximum accuracy of 91% when the mean and variance are the features used to train SVM.

In 2018, Robin et al. [90] modified the proposed methodology in [55]; they estimate two independent components from reference signals (a non-malicious App running in the foreground) and suspicious signal (a reference signal + emulated malware running in the background) with two approaches: symmetry and deflation. In addition, the authors add to the methodology four probability distributions tanh, pow3, gauss, and skew, in the Fast ICA. Using all of the combinations between ICA approaches and distributions, the methodology finds a correlation between the independent components and the input signals. The results of the correlation feed three classifiers: Naive Bayes, Support Vector Machine, and Random Forest. The results show acceptable results in recognizing low duty cycles, reaching an average accuracy of 85%.

Cui et al., in 2015, [33] designed a cloud solution for identifying malware analyzing the packets sent by a smartphone. The architecture includes a smartphone connected to a gateway. The gateway is connected to a service-oriented mobile malware detection system (SMMDS) in the cloud. The technique compares the packet information of a new App with previous information extracted from malicious and benign Apps. The SMMDS system uses contraction clustering to train and test features.

In reference [31], the authors analyze the packets of the network traffic transmitted by a device to recognize malicious Apps. The architecture used is a smartphone connected to an access point. The access point is connected to a hub, and one port of the hub is connected to a computer that has Wireshark running on it. The packets of interest are HTTP GET/POST packets with device information such as IMEI, IMSI, device model, software information. Furthermore, the methodology analyzes the legitimacy of the remote server where the packets are sent. The proposed methodology analyses the information described above of each App through a flowchart, which determines if the App is malicious

or non-malicious. To test the model, they use 11 malicious and nine benign Apps, obtaining 100% accurate recognition.

Wei et al. [94] use DNS packets information to analyze the geographical position of malicious servers. Based on the position of the servers, their approach can identify malware. They create a Geographical matrix which contains m rows with Android Apps and n columns with geographical and network features using benign and malicious Apps. They apply Independent Component Analysis(ICA) to this matrix to obtain a latent matrix. This matrix identifies malicious and not malicious Apps.

In reference [77], Loukas et al., in 2018, proposed an innovative methodology for unveiling malicious behavior in autonomous mobile robots to avoid denial of service, spoofing and malware attacks. They use a combination of metrics such as the network incoming, network outgoing, CPU, disk data, encoder, accelerometer, power consumption, current to characterize these kinds of attacks using Recurrent neural networks.

It is necessary to highlight that this thesis can be considered unique for the following reasons:

- The methodologies proposed in this thesis are tested in real and emulated malware, whereas other works cited before [28][104][55][60][77] use only synthetic or emulated malware. Only in reference [23], the authors used real ransomware in the tests. However, they do not explain in detail about the real ransomware used.

- The tool to automate the collection of power consumption measurements uses ADB commands and Droiboit. In contrast, most of the user's input generators use only Monkey or Droidbot [8][14][102][100][12].

- The methodologies proposed in this thesis only use one parameter (power consumption) to identify malware. On the other hand, cited papers [8] [14] [102][100][12] use many parameters to unveil malicious behavior. Thus, these consume more resources in terms of memory, processing, and energy to collect and store the parameters.

Chapter 3

Experiments and Automation Tool

This chapter explains in detail how power consumption measurements have been taken and stored automatically to create two datasets. Specifically, it presents a testbench used to measure the average power consumption of the smartphone with its components. Moreover, the process of designing an emulated malware App is described and the automation tools necessary to generate power traces automatically without human interaction for emulated malware and real malware. Data collection is one of the most important phases in the present work because an extensive evaluation of both methodologies is needed.

3.1 Experimental setup

3.1.1 Testbench

Figure 3.1 shows a general diagram of the testbench used in the present work to take power consumption measurements. The main components of the diagram are the following:

- Monsoon Power Monitor: This reliable device measures different electrical parameters, namely, voltage and current, in the real-time of a device while providing constant direct current voltage to the smartphone. The sample rate to take the measurements is 5000 samples/s. The power monitor terminals are connected to the pins of the smartphone's battery. The smartphone's battery is bypassed to avoid a short circuit. At the same time, the power monitor has a connection with a CPU or server through a USB cable.

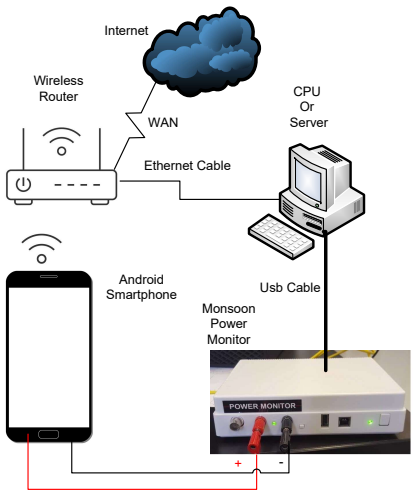


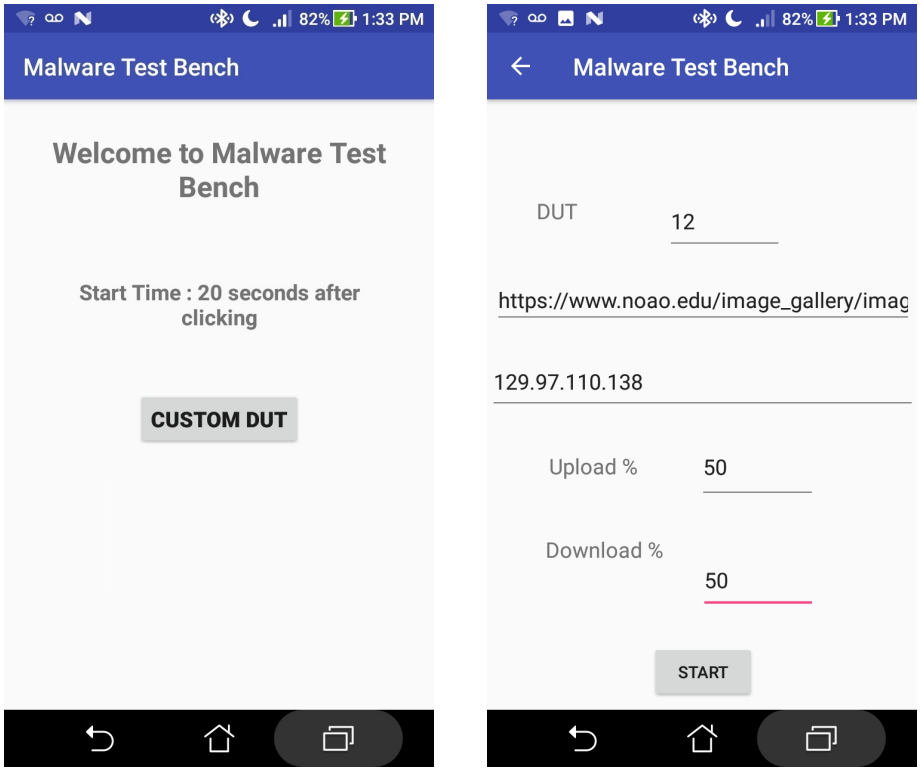
Figure 3.1: Testbench to take power consumption measurements

- Computer or server: This device is responsible for triggering power monitor, initiating a benign or malicious App on mobile device through wifi. In addition, this cpu stores the power consumption measurements. The code that integrates all the applications is developed in Python.
- Wireless router: This device provides a wireless connection between the computer or server and the smartphone. In a parallel connection, it provides internet to the smartphone.

3.1.2 Emulated Malware Android Application

A customized App that emulates real malware has been developed. This App mimics real malware downloading or uploading files from or to a malicious server, depending on its configuration.

Explicitly, this App starts a service that runs in the background of the smartphone, and it executes the actions of downloading and uploading. During a downloading action, the App downloads a video from the Internet. On the other hand, the uploading action consists of the generation of a 20 Mb random image that is uploaded to a specific server on the Internet. The App has two screens, as shown in Fig. 3.2(a) and Fig. 3.2(b). The first screen has one button called CUSTOM DUT. After touching this button, the App



(a)Interface application screen 1 (b) Interface application screen 2

Figure 3.2: Interfaces application emulated malware

redirects the user to the second screen. In the second screen, there are three configurable parameters:

- Duty cycle: This parameter denotes the "time of activeness" of the downloading or/and uploading actions in 1 minute of the total duration, as described in Fig. 3.3. To illustrate, if the duty cycle is 1%, the emulated malware will act for 0.6s each minute. If it is 12%, it will act for 7.2s each minute.
- Percentage of downloading: This represents amount of time within the duty cycle when the App executes the downloading task. Figure 3.4 shows a configuration example of the App's parameters. When the duty cycle is 12% and 100% downloading, the emulated malware will act during 7.2s each minute, and the App will download the video all the time. For example in another case, if a duty cycle is 12%, a percentage of uploading is 75%, and downloading 25%, the malware will act 7.2s each

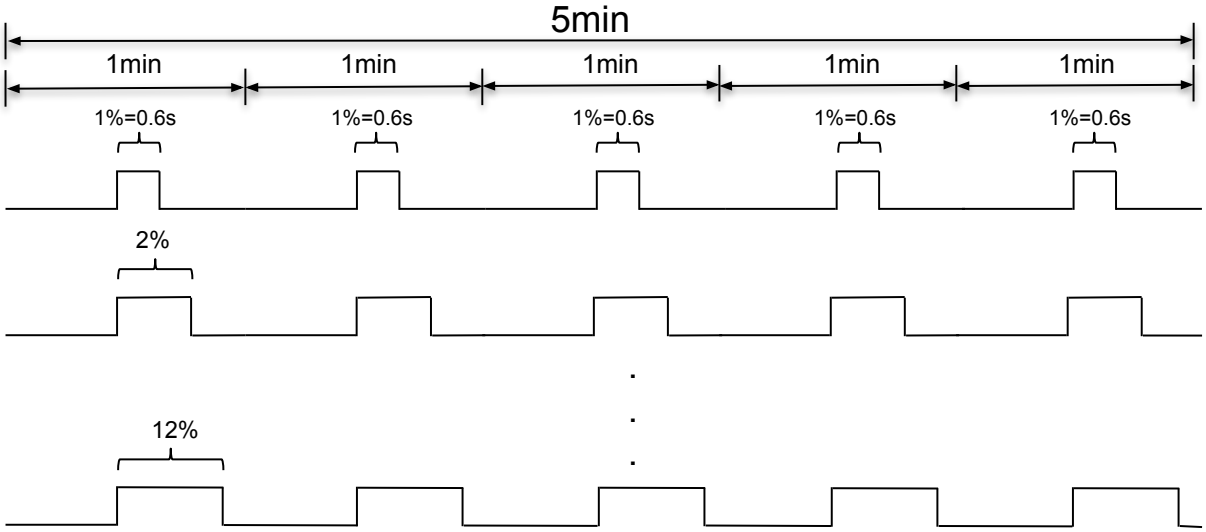


Figure 3.3: Duty cycle

minute. The emulated malware will upload the image during 5.4s of 7.2s, and it will download a file during 1.8s of 7.2s. It is necessary to highlight that App does not download the video entirely in the "time of activeness". The App only downloads a little percentage of the file. Thus, every time that the emulated malware acts, the App reinitializes the action of downloading or uploading from the beginning.

- Percentage of uploading: This represents the uploading stage time within the duty cycle.
- Additional fields: There are two optional parameters to configure. The first describes the URL to download the image. Therefore, the user can download an image, video, or file from another site on the Internet. The second is the public IP of the server used to upload the image.
- Start button: After configuring the parameters described above, the user can press the button to initialize the emulated malware actuation.

3.1.3 Emulated Malware Dataset

The emulated malware dataset is composed of two kinds of signals: reference and malicious.

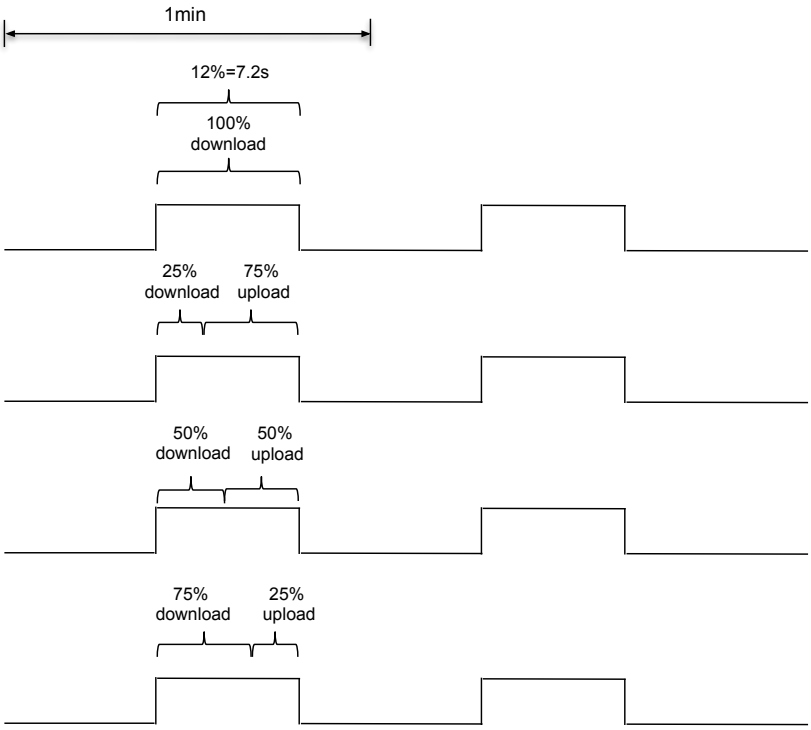


Figure 3.4: Duty cycle with percentages of activeness

Reference signals are power consumption measurements taken for five consecutive minutes while a Youtube App is running a specific video in the foreground, and there is no another App running in the background.

Each power trace is taken while the following procedure is running on the smartphone

- The Youtube App is opened.
- The video is chosen from Youtube.
- The video plays.

The total time of each power measurement is 5 minutes. This experiment is repeated 15 times to acquire sufficient data power measurements for the dataset.

It is important to point out that, for consistency, each experiment has been executed under the same smartphone's configuration, such as screen brightness, no sound, and the version of the operating system.

Malicious signals are power consumption measurements taken for five consecutive minutes while Youtube App is running in the foreground, and an emulated malware is running in the background.

The emulated malware runs cyclically for 5 minutes with a pre-set duty cycle, and percentages of download and upload as explained in Subsection 3.1.2.

Before power consumption measurements begins, the App initializes the emulated malware to run in the background of the smartphone. Then, the power consumption is taken while the Youtube App runs a video in the foreground as it was described in Reference signals.

3.1.4 Automation emulated malware

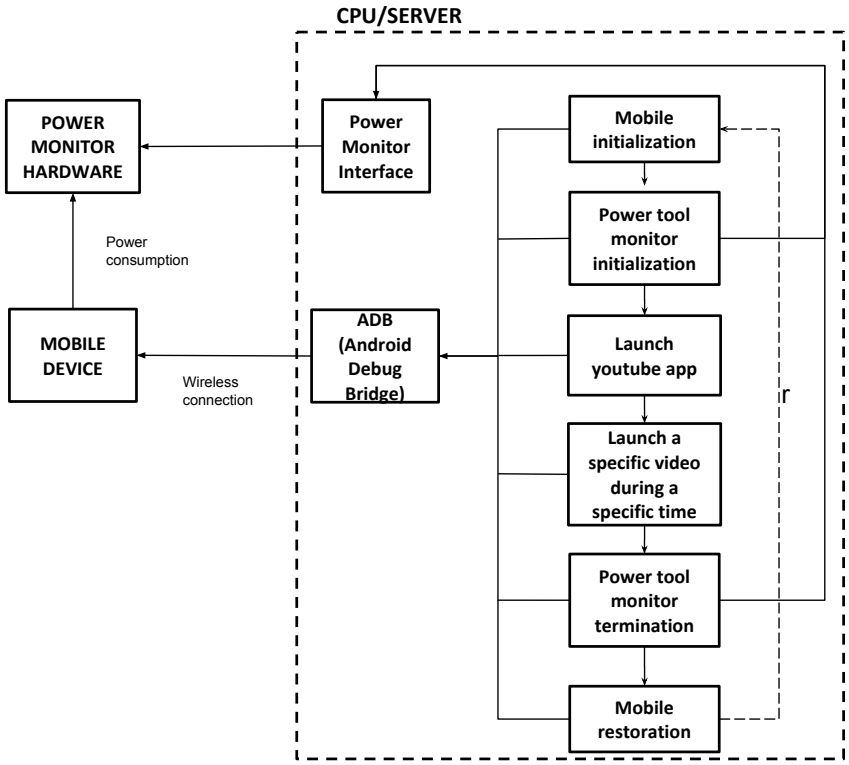


Figure 3.5: Automated generation of reference signals to create emulated malware dataset

Android Debug Bridge (ADB) commands and different APIs integrated in Python are

used to take power consumption measurements without human interaction.

ADB is a console terminal tool used to communicate a computer or server with an Android smartphone. A user or a tool can execute different commands through the console to send simple or complex tasks to the smartphone, such as touches, clicks, unlock, lock/unlock, turn on/off the background light, open Apps, and others.

The flow chart designed to take power consumption measurements automatically for reference signals is shown in Fig. 3.5. The stages of the flow are described below:

1 **Mobile initialization**: ADB commands turn on the smartphone's back-light screen and unlock the smartphone.

2 **Power tool monitor initialization**: Python initializes an API connection with Monsoon Power Monitor. In addition, Monson Power Monitor starts the power consumption sampling.

3 **Launch Youtube**: ADB commands open the Youtube App.

4 **Launch a specific video during a specific time**: ADB commands launch a specific video and keep running the video for 5 minutes.

5 **Power tool monitor termination**: Python stops the power consumption sampling, and it stores a .csv with the results of the power consumption trace.

6 **Mobile restoration**: ADB commands close the Youtube App, lock the mobile device, and turn off the back-light of the screen.

Python repeats the stages described above in a loop for r times, depending on how many measurements the user wants to generate. In the present work, 15 measurements of reference signals have been taken.

Figure 3.6 shows the flow chart used to record malicious signals.

The stages are similar to the flow chart for reference signals in Fig. 3.5. However, one stage is added to the flow chart, called configuration of emulated malware. In this stage, ADB commands fill out the text boxes of the emulated malware App described in Sec. 3.1.1. These fields are the duty cycle, percentage of downloading, and percentage of uploading. Finally, the emulated malware is begun by pushing the button START in the App. The emulated malware runs in the smartphone's background while the power consumption measurement is taken, following the same stages used to take reference signals.

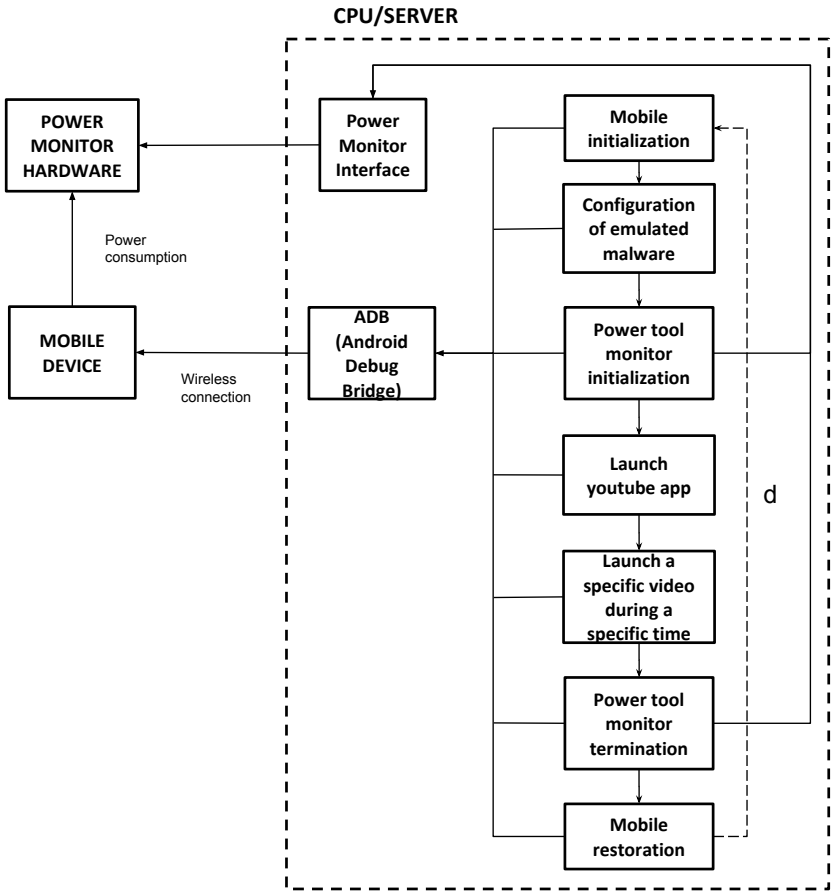


Figure 3.6: Automated generation of malicious signals to create emulated malware dataset

Python repeats the experiment d times, keeping the same conditions on the smartphone. In this thesis, 15 measurements have been taken for each of the following percentages of duty cycles: $Dut = 1\%$, $Dut = 2\%$, $Dut = 3\%$, $Dut = 4\%$, $Dut = 8\%$, and $Dut = 12\%$, with 5 additional scenarios considering different percentages of downloading and/or uploading as follows:

- Scenario 1: The emulated malware downloads a file during 100% of the duty cycle time. For example, if Dut=1% is set, the emulated malware downloads a file from the Internet for 0.6s of each minute.
- Scenario 2: The emulated malware downloads a file during 25% of the duty cycle time, and for the remaining 75%, it uploads a file. To illustrate, if Dut=1% is set,

the emulated malware downloads a file for 0.15s, and for the remaining 0.45s, it uploads an image to a server on the Internet. This process is repeated each minute.

- Scenario 3: The emulated malware downloads a file during 50% of the duty cycle time, and for the remaining 50%, it uploads a file. To illustrate, if Dut=1% is set, the emulated malware downloads a file for 0.3s, and for the remaining 0.3s, it uploads an image. This process is repeated each minute.

- Scenario 4: The emulated malware downloads a file during 75% of the duty cycle time, and for the remaining 25%, it uploads a file. To illustrate, if Dut=1% is set, the emulated malware downloads a file for 0.45s, and for the remaining 0.15s, it uploads an image. This process is repeated each minute.

- Scenario 5: The emulated malware uploads a file during 100% of the duty cycle time. For example, if Dut=1% is set, the emulated malware uploads an image to the Internet for 0.6s of each minute.

3.1.5 Real Malware Dataset

The real malware dataset has been created using Apps with malware from **Drebin** repository and Apps without malware from **Play Store** repository. A pair of Apps with equivalent characteristics and user interface has been selected from both Apps repositories.

3.1.6 Automation real malware

ADB commands, API tools, and Droidbot integrated in Python are used to obtain power consumption measurements automatically.

Droidbot is undoubtedly the most important component in the automation tool developed because it triggers sensitive behaviors of each App emulating user's inputs. Droidbot acts as a user reproducing touches on the screen, intent calls, uploading and downloading actions, or usage of sensors of the smartphone to trigger sensitive behaviors.

Li et al., in [73], highlights some advantages of Droidbot over other user's input generators such as:

- Droidbot does not need instrumentation of each App because it is based on a GUI model. Thus, Droidbot can control all Android Apps.

- Droidbot creates a path on the fly to execute the next action monitoring if the input performed causes a change in the current state. Thus, Droidbot can trigger sensitive behavior efficiently.

The flow chart in Fig. 3.7 shows the main stages to take power consumption measurements automatically of real malware. This flow chart shows the main stages for malicious and benign real Apps automation.

1 **Power tool monitor initialization**: Python initializes an API connection with Monsoon Power Monitor. In addition, Monson Power Monitor starts power consumption sampling.

2 **Mobile initialization**: ADB commands turn on the smartphone's back-light screen, unlock the smartphone, and install the App.

3 **Droidbot**: Droidbot generates user inputs on the fly. It explores the events in the App in a greedy depth strategy.

4 **Mobile restoration**: ADB commands uninstall the App, close Apps and windows opened by Droidbot during the test. Finally, the mobile device is locked, and the back-light of the screen is turned off.

5 **Power tool monitor termination**: Python stops the power consumption sampling, and it stores a .csv with the results of the power consumption trace.

Python repeats the flow chart described r times in a loop for benign Apps and d times for malicious Apps. In this thesis, we repeat the experiment 15 times for each App.

3.1.7 Dataset

The automation tools have been used to generate two large datasets with the following characteristics

- Dataset Emulated Malware
  - Benign signals: 15 measurements
  - Malicious signals: 15 measurements x 6 duty cycles x 5 scenarios=450 power consumption measurements

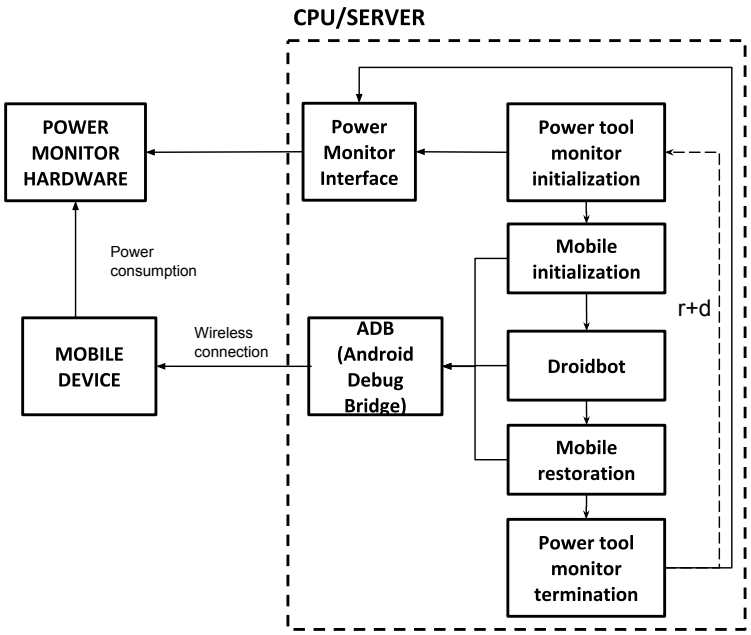


Figure 3.7: Automated generation of benign and malicious signals to create real malware dataset

- Dataset Real Malware
  - Benign signals: 15 measurements x 5 applications= 75 power consumption measurements.
  - Malicious signals: 15 measurements x 5 applications=75 power consumption measurements.

3.1.8 Preliminary analysis

A preliminary analysis has been done using emulated and real malware power consumption datasets. These datasets have been collected manually without the automation tool described in Subs. 3.1.4 and 3.1.6.

Emulated Malware dataset

In the dataset of emulated malware, reference signals are power consumption measurements taken for five consecutive minutes while a Youtube App is running a specific video in the foreground, and there is no App running in the background. Each power measurement is taken as soon as the Youtube App is opened, the name of the video is typed, and the video is played. This procedure was repeated 15 times, keeping the same configurations in the smartphone in terms of the brightness of the screen and version of the operating system.

Malicious signals are power consumption measurements taken for five consecutive minutes while a Youtube App is running in the foreground and emulated malware is running in the background. The emulated malware has been configured to run cyclically in the total time with a pre-set duty cycle. The action that the emulated malware executes during the duty cycle is to start a connection with a remote server, and only download specific files imitating the behavior of real malware. The emulated malware running in the background with distinct percentages of activeness have been collected and named $Dut = 1\%$, $Dut = 2\%$, $Dut = 3\%$, $Dut = 4\%$, $Dut = 8\%$, and $Dut = 12\%$. The cycle length considered is 60 seconds. Fifteen power measurements for each percentage of emulated malware have been collected. Therefore, 90 measurements that represent malicious signals and 15 measurements that represent non-malicious signals are available to construct the model.

This dataset has been evaluated with two techniques:

- Statistical analysis: The mean and the variance of each of the signals are obtained. Then, the results are plotted in Fig. 3.8. An extension of the results can be found in Appx. A. As we can see, malicious power measurements are concentrated in one cluster, while reference in another cluster. However, the features are not sufficient to discriminate both classes clearly with clusters separated by large distances. In [55], the authors used these features to create a classification problem, obtaining an average F1-measure of 91%.

- Signal processing: Cross-validation is applied between signals of the same class, as can be seen in Fig. 3.9. Cross-validation measures the similarity of two signals convolving one signal with another one. This operation returns a unique value, between 0 and 1. Where 1 means that both signals are very similar, while 0 signifies that both signals are very different. A Cross-validation value has been found among all the possible combinations of power consumption measurements of the same class. Finally, histograms are plotted for each malicious class against a benign class. The results obtained have been plotted in Figs. 3.10, 3.11, 3.12, 3.13, 3.14, 3.15. As

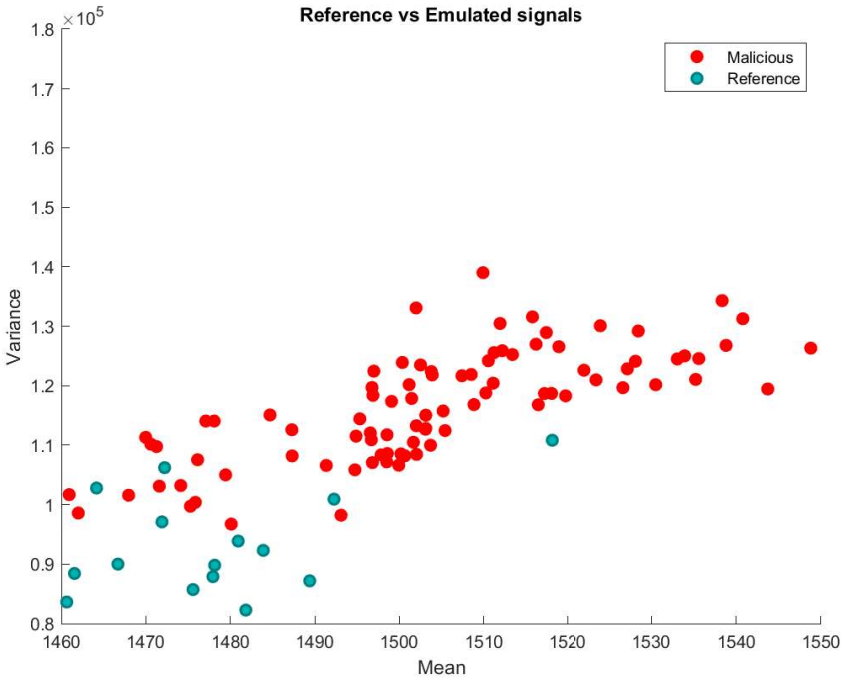


Figure 3.8: Means and Variances of reference and emulated malware with percentages 1,2,3,4,8,12

we can see, the mean in all the scenarios using cross-validation approach is ∼0.9. Therefore, this characteristic is not useful in differentiating references from emulated malware.

Real Malware dataset

This dataset has 15 power consumption measurements for one game App with malware from Drebin Dataset, and 15 measurements for a similar benign App from Play store. The power consumption measurements are collected without the automation of the App. Therefore, the power trace is taken while the following procedure is running:

- The App is opened.
- The App runs for 5 min without any user interaction.

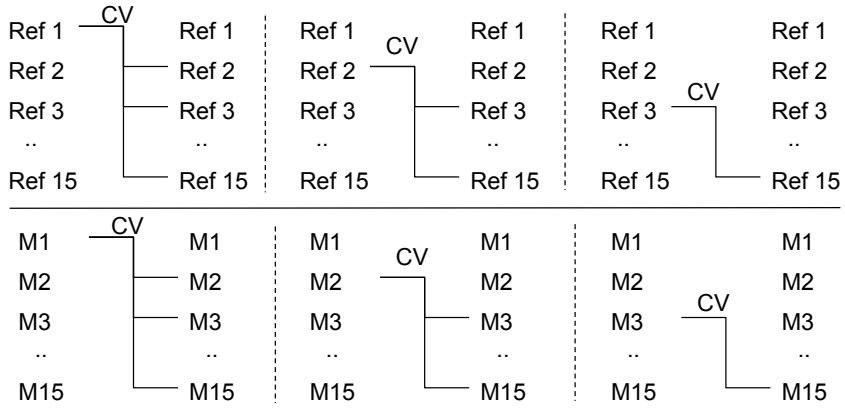


Figure 3.9: Cross-validation operation applied to all possible combinations of same class

The dataset has been evaluated considering the same techniques as in the emulated dataset.

- Statistical analysis: The mean and variance have been calculated for each signal. The results are plotted in Fig. 3.16. Nonetheless, the mean and variance are not satisfactory characteristics for separating malware and non-malware Apps.
- Signal processing: A Cross-validation value has been found among all the possible combinations of power consumption measurements of the same class. The results are plotted as a histogram in Fig. 3.17. As shown, the values of the autocorrelations are similar for non-malicious and malicious signals. Thus, the autocorrelation is not a desirable feature for classifying signals.

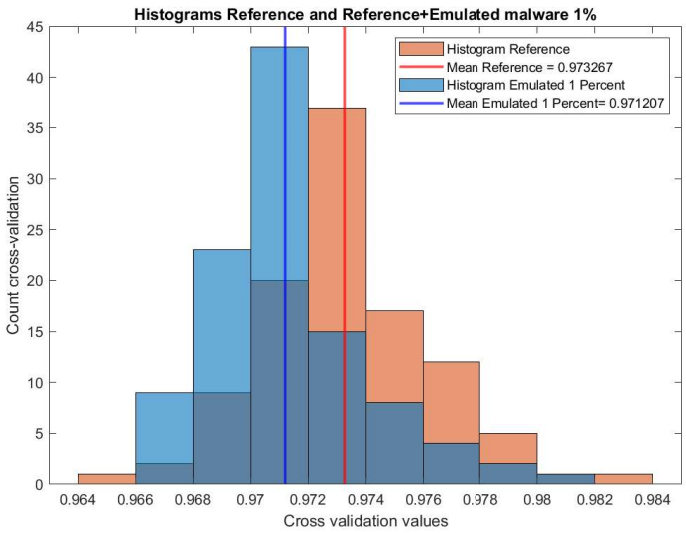


Figure 3.10: Histogram of cross-validation of Reference and Emulated malware 1%

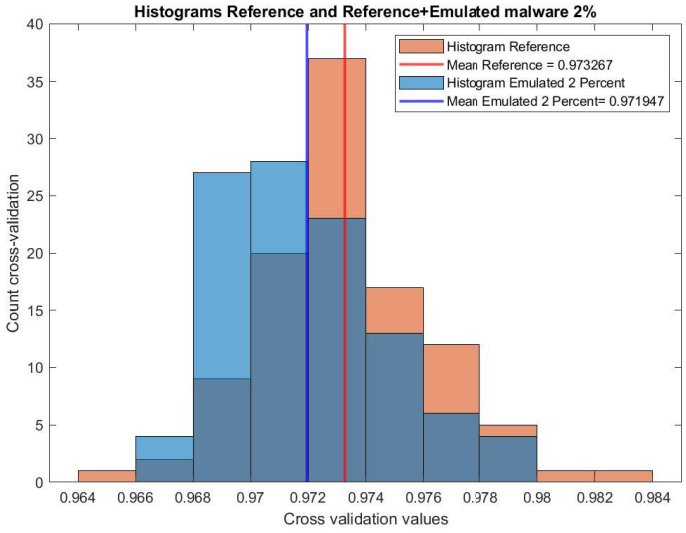


Figure 3.11: Histogram of cross-validation of Reference and Emulated malware 2%

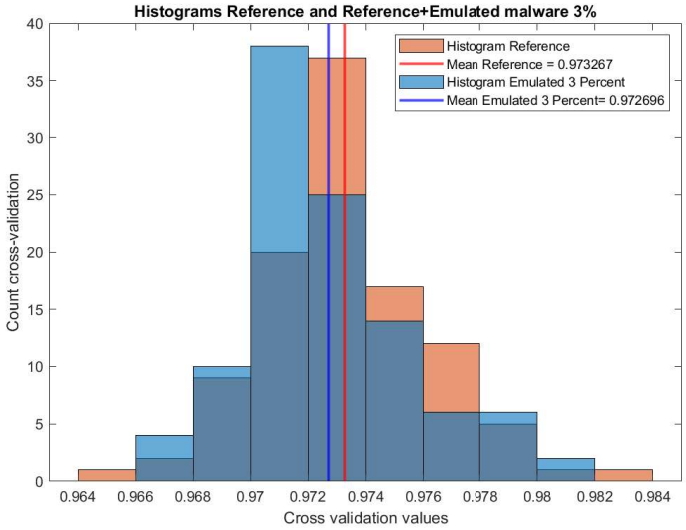


Figure 3.12: Histogram of cross-validation of Reference and Emulated malware 3%

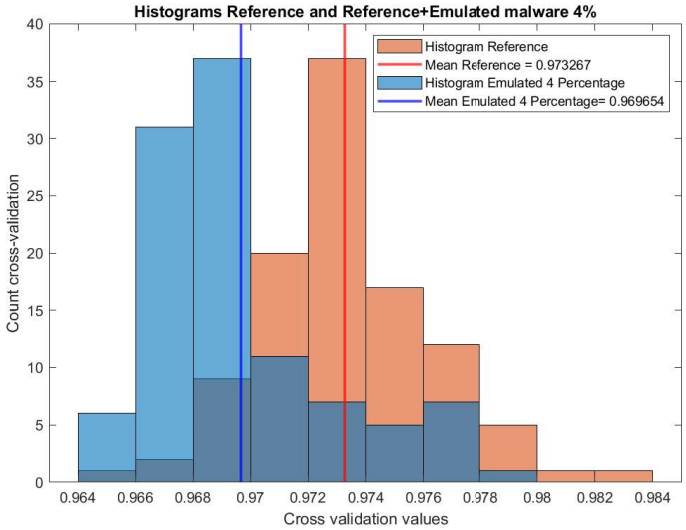


Figure 3.13: Histogram of cross-validation of Reference and Emulated malware 4%

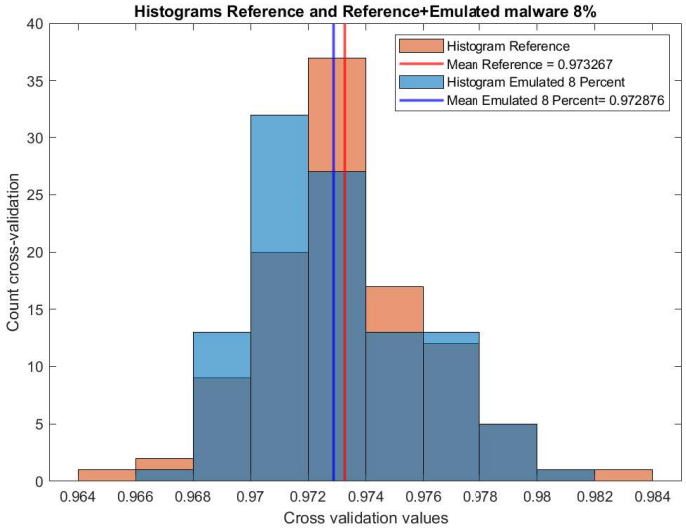


Figure 3.14: Histogram of cross-validation of Reference and Emulated malware 8%

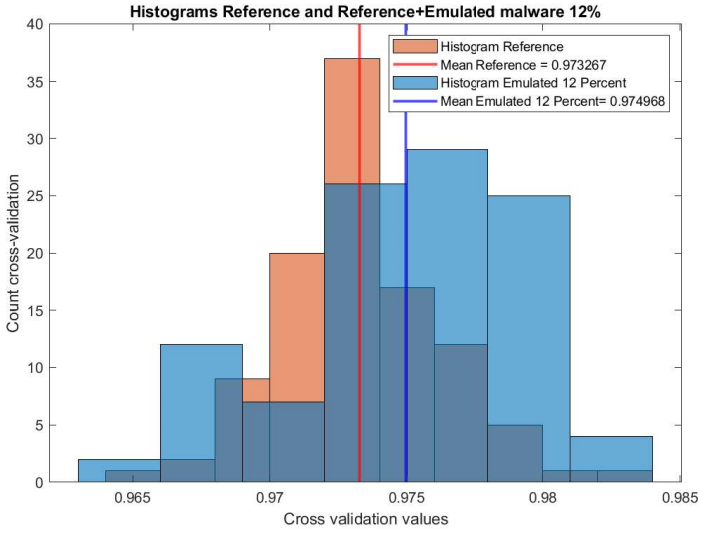


Figure 3.15: Histogram of cross-validation of Reference and Emulated malware 12%

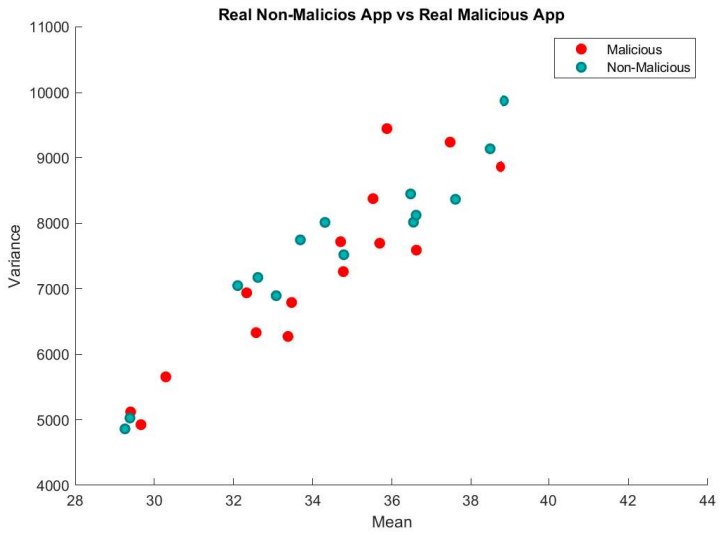


Figure 3.16: Mean and Variance of Real Malware and Non-malware Apps

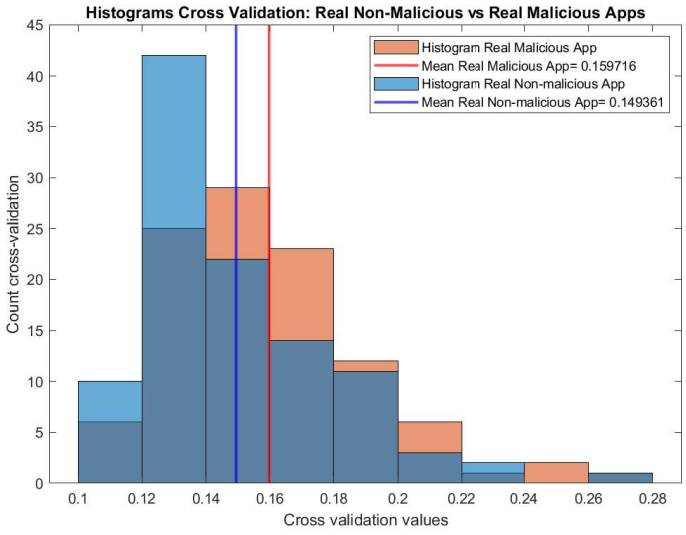


Figure 3.17: Histogram of Cross-validation of Real Malware and Non-Malware Apps

Chapter 4

Methodology

This chapter explains the methodology proposed to unveil malicious behavior using parametric Changepoint detection in Feature Extraction Stage. Chapter 5 describes a similar methodology, in which non-parametric Changepoint detection is used in the same stage. Parametric Changepoint detection uses two assumptions to extract meaningful features from a time-series signal.

4.1 Methodology

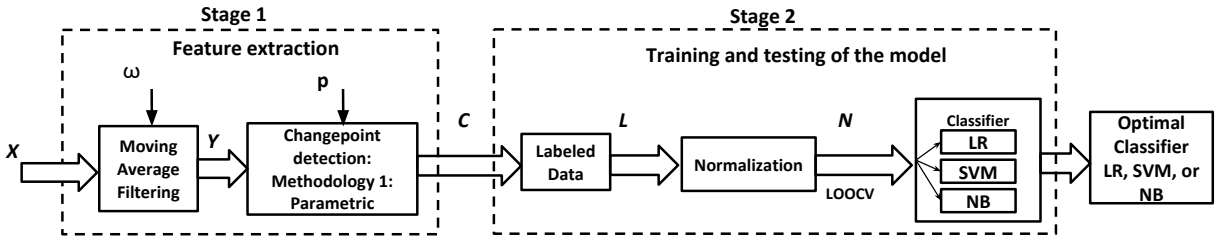
This methodology has been divided into Procedure 1 and Procedure 2, as shown in Fig. 4.1 and Fig. 4.2, respectively. Procedure 1 trains and tests the model with a dataset, and Procedure 2 describes the process that a new measurement without a label has to follow to be classified as anomalous or not. To represent each of the input and output variables in the stages of both procedures, the following notation is used. Matrices are represented by a bold capital letter (e.g., $\mathbf{X}$). Vectors are denoted by a capital letter in italics (e.g., X), and each element of a vector is represented by a lower case letter in italics (e.g., x).

Procedure 1 uses two kinds of signals to train and test the model: benign and malicious. A benign signal is defined as a power consumption measurement taken over a specific time from a smartphone while a verified non-malicious App is running on it. This signal is represented by a vector B with the dimension n. The value of n depends on the time duration of the measurement multiplied by the sample rate. Thus, $B_i = (b_{i_1}, b_{i_2}, b_{i_3}, ..., b_{i_n})$. A benign signal is taken r times keeping the same time duration and conditions in terms of the configuration of the smartphone. The matrix $\mathbf{B}$ concatenates r benign signals. Thus,

this matrix has dimensions (r x n). Each row of matrix $\mathbf{B}$ is represented by B_i.

In contrast, a malicious signal is defined as a power measurement signal taken over a specific time while an App with malicious code is running on a smartphone. As described in Chapter 3, two kinds of malware will be analyzed: emulated and real. When we are analyzing emulated malware, a malicious signal can be interpreted as a benign App running in the foreground and an emulated malware running in the background. When we are analyzing real malware, a malicious signal is just real malware App running in the foreground of the smartphone. A malicious signal is denoted by the vector $M_i = (m_{i_1}, m_{i_2}, m_{i_3}, ..., m_{i_n})$. This signal is taken d times. Thus, a matrix denoted as $\mathbf{M}$ concatenates d malicious signals and has dimensions (dxn). Each row of matrix $\mathbf{M}$ is denoted by a vector M_i.

The matrices $\mathbf{B}$ and $\mathbf{M}$ have been concatenated in a matrix $\mathbf{X}$ with dimensions ((r+d)xn). Each row of matrix $\mathbf{X}$ is denoted by X_i, which can represent a benign or a malicious signal depending on the value of i.

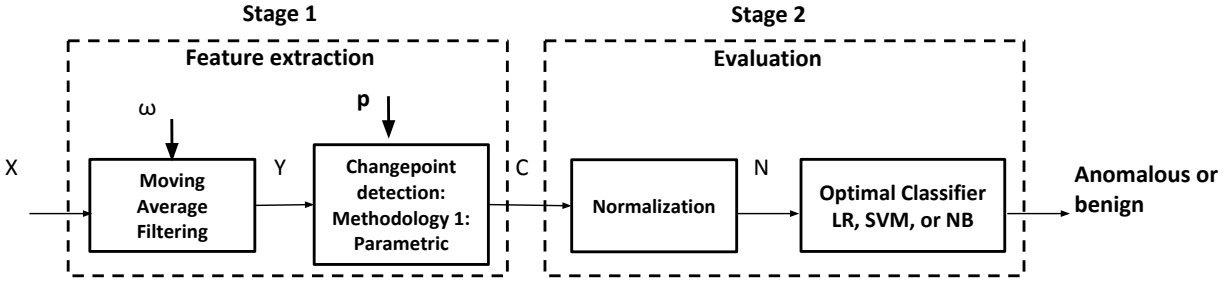


Legend:
r: number of reference signals; d: number of malicious signals; n: number of observed values.
Methodology 1: ***X****:(r+d) x n;* ***Y****:(r+d)x(n/ω);* ***C****:(r+d) x f;* ***L****:(r+d)x(f+1);* ***N****:(r+d)x(f+1); ω: window size; f: features;* p: penalty value
LOOCV: Leave one out cross validation; LR: Logistic regression; SVM Linear: Support Vector Machine Linear; NB: Naive Bayes

Figure 4.1: Procedure 1: Data Preparation and Model Training and Testing

Procedure 1 uses supervised and offline learning methods because labelled power measurement signals obtained beforehand are used to train and test the models. The first stage of Procedure 1, as is seen in Fig. 4.1, is Feature Extraction. This stage is composed of two sub-stages: Filtering and Changepoint detection. Filtering is used because the sample rate in the data acquisition (5000 samples/s) is relatively high; hence, different kinds of noise generated by some sources can be affecting the real behavior of the signal. To prevent this disturbance, a moving average filter is used. Parametric Changepoint detection is the second sub-stage, and it has the goal to recognize meaningful features in each of the signals.

The second stage of Procedure 1 comprises training and testing of the model. This stage contains three sub-stages: Labeling, Normalization, Training and Testing of the



Legend:
n: number of observed values
Methodology 1: *X: Suspicious Signal (1 x n); Y:(1 x (n/ω)); C:(1 x f); N:(1 x f)*; ω: window size; f: features; p: penalty value
LOOCV: Leave one out cross validation; LR: Logistic regression; SVM Linear: Support Vector Machine Linear; NB: Naive Bayes

Figure 4.2: Procedure 2: Detection procedure for classifying an unlabeled signal as malicious or not malicious

classifier. A labelling sub-stage labels the data features extracted by the Changepoint detection algorithm. The label is 0 when the features are extracted from a vector B_i, and 1 when the features are extracted from vector M_i. The second sub-stage is data normalization, in which each feature is scaled between 0 to 1, considering all of the rows of matrix $\mathbf{X}$ using the min-max method. This sub-stage is applied to increase the performance of the classifier's optimization problem [7]. Finally, the last sub-stage has the purpose of choosing an optimal classifier, which should be the method that provides the best accuracy and F1-measure using the optimal hyperparameters. Support Vector Machine (SVM) linear, Logistic Regression, and Naive Bayes are evaluated as possible classifiers. After this procedure is executed, a classification model is obtained.

Procedure 2 shown in Fig. 4.2 is called detection, and it comprises two stages: Feature Extraction and Evaluation. The Feature Extraction stage includes the same sub-stages as in Procedure 1, and the evaluation stage uses only normalization and evaluation of the features.

In Subsections 5.1.1 and 5.1.2, Procedure 1 is described in detail, while in Subsection 5.1.3 Procedure 2 is defined.

4.1.1 Procedure 1: Feature Extraction.

As described before, the Feature Extraction stage is composed of two sub-stages: Filtering and Changepoint detection.

Filtering.

The moving average filters the signal to eliminate noise of each of the signals of matrix $\mathbf{X}$. This kind of filter belongs to the class of finite impulse response filters, which are used for smoothing and waveform shaping of signals. In addition, this filter can be considered as one of the simplest filters that provides the lowest noise if compare with other linear filters [89]. Moving average is applied to each of the rows X_i of the matrix $\mathbf{X}$ using Eq. (5.1)

$$Y_{i_j} = \frac{1}{w} \sum_{g=0}^{w-1} X_{i_{w*j+g}} \quad , \tag{4.1}$$

where:

- w window size: This parameter determines how many values of the signal X_i are averaged. To choose the optimal value of the window size, a range of values of the window size have been tested together with other parameters of Procedure 1. The optimal combination of these parameters that provides the best accuracy and a normality error less than a constraint for all of the scenarios is chosen as the final value for the window size w.
- X_i input power consumption: This signal represents the raw signal power consumption with dimensions $(1xn)$.
- Y_i output filtered signal: This signal is the resultant of Eq.5.1 with dimensions $(1x(n/w))$.
- g: This index is the current position of signal X_i.
- j: This index is used for the new signal filter signal Y_i. j takes values between 0 and $trunc(n/w)$.

A matrix $\mathbf{Y}$ with dimensions $(r+d)$x(n/w) is obtained after this sub-stage. Throughout this paper, the term *window* with the symbol w denotes the number of observed values of the signal X_i, grouped and averaged to obtain a smooth signal Y_i. The term "interval" describes the set of values of the smooth signal Y_i, grouped through the theory of parametric changepoint detection. The term "interval" is used in the next sub-stage.

Parametric Changepoint

The Changepoint detection sub-stage extracts meaningful features of each of the filtered signals Y_i. This chapter uses the theory of parametric Changepoint detection, which can identify sudden abrupt changes in each of the filtered signals Y_i. This theory groups datapoints with similar statistical properties in intervals. Figure 4.3 shows a set of data points of the signal with similar statistical parameters grouped in many intervals [59]. All the intervals must follow the same mass probability function. Parametric changepoint theory considers two assumptions:

1 The random variables in each interval follow a Gaussian distribution.

2 Each of the random variables contained in each interval must be independent of one another.

Considering these assumptions, Changepoint theory can extract relevant features of time-series signals. The features selected as important in this methodology are:

- Feature 1: Number of changepoints
- Feature 2: Mean of the changepoint intervals' mean
- Feature 3: Mean of the changepoint intervals' variance

The number of changepoints is an important feature because we assume that the actions of emulated or real malware in the smartphone will affect the signal as a changepoint due to the change in the signal's statistical properties.
Features 2 and 3 are significant for the model because they characterize every interval according to statistical properties. Feature 2 and 3 are calculated after Feature 1 has been found.

To explain the theory of Changepoint detection, we have considered an appropriate notation, which will be used in Eqs. (4.2), (4.3), and (4.4). These equations describe the cost function used to find the optimal position of the changepoints in a signal Y_i.
The notation used is as follows:

- k : This variable represents the number of intervals in which the signal Y_i will be divided, as can be seen in Fig. 4.4. k can take values between 0 to K.

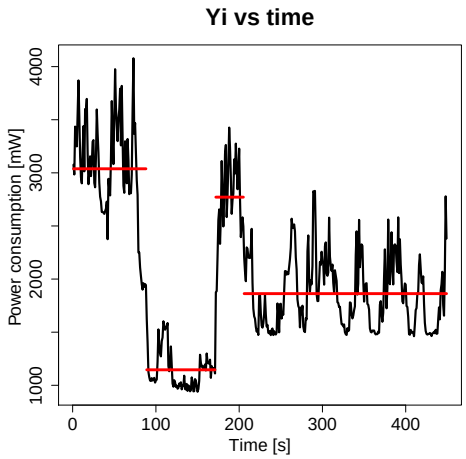


Figure 4.3: Changepoint detection applied to one part of the signal Y_i

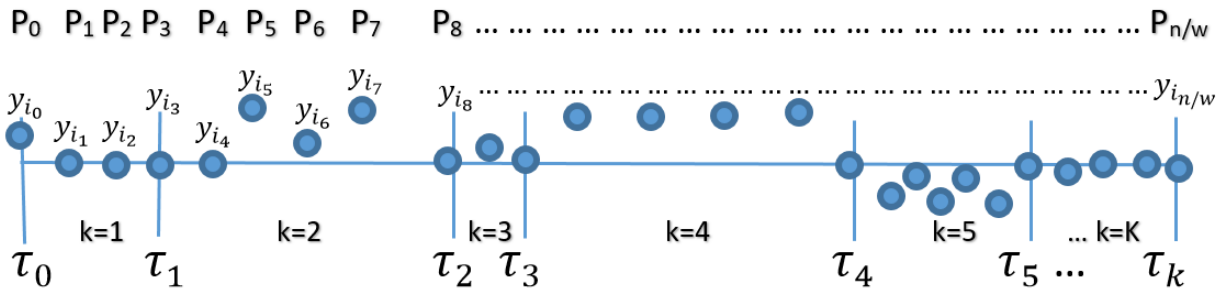


Figure 4.4: Datapoints of the signal Y_i group in intervals through the Changepoint theory.

- $K-1$: This variable denotes the number of changepoints in the signal Y_i. Each changepoint characterizes the transition between intervals with distinct statistical properties.
- τ_k : This variable defines the position of each changepoint. The vector $\tau = (\tau_0, \tau_1, ..., \tau_K)$ contains all the positions of the changepoints in the signal Y_i. We have used the notation τ'_k to denote τ_{k-1}.
- y_{i_j} : This variable corresponds to each element of the vector Y_i after the sub-stage of filtering.
- Interval : After applying the theory of Changepoint detection, each interval will con-

tain a set of data points between $(y_{i_{\tau'_k}}, y_{i_{\tau_k}}]$. To illustrate, if we analyze the second interval k=2 in Fig. 4.4, the interval will include all the data points between $(y_{i_{\tau_1}}, y_{i_{\tau_2}}]$. Hence, the data points with positions P4, P5, P6, P7 and P8 will be considered in the interval.

To understand the theory of changepoint detection, we assume that the number of changepoints $K-1$ of the signal Y_i is known. Thus, we can assign a cost to group a set of y_{i_j}'s in each interval, using a Gaussian mass probability function. Then, we can sum up all the intervals' costs to obtain a total cost for each signal Y_i. This total cost function is described in Eq. (4.2). To find the best interval grouping, the total cost function is minimized by grouping datapoints y_{i_k}'s among sequential intervals. After this minimization problem, we know the optimal position of each changepoint that separates intervals that contain data points of the signal Y_i with similar statistical properties.

$$J(\tau, Y_i) = \frac{1}{n/w} \sum_{k=1}^{K} G((y_{i_{\tau'_k}}, y_{i_{\tau_k}}]) \tag{4.2}$$

Considering that each interval of the signal Y_i follows a Gaussian distribution with changes in the mean and variance, the function G can be modelled by the twice negative Gaussian log-likelihood with changes in the mean and variance [70][50]. Hence,

$$G((y_{i_{\tau'_k}}, y_{i_{\tau_k}}]) = (\tau_k - \tau'_k) \log(\hat{\sigma}^2_{(\tau'_k, \tau_k]}) \ , \tag{4.3}$$

where

$$\hat{\sigma}^2_{(\tau'_k, \tau_k]} = \frac{1}{(\tau_k - \tau'_k)} \sum_{i=(\tau'_k+1)}^{\tau_k} (y_i - \overline{y}_{(\tau'_k, \tau_k]})^2 \ , \tag{4.4}$$

and $\overline{y}_{(\tau_{k'}, \tau_k]}$is the empirical mean.

The assumption to understand this theory is not true in real Apps. Hence, we add to the cost function an additional penalty value *pen* to obtain the number and position of changepoints automatically [70][50], as shown in Eq. (4.5):

$$H(\tau, Y_i) = J(\tau, Y_i) + pen. \tag{4.5}$$

The value of the penalty must be chosen to obtain the highest F1-measure and lowest normality error. Different approaches such as Schwarz's information criterion [88], Akaike's information criteria [6], or an adaptive choice of the penalization parameter [70] have been

used to accomplish this requirement, but in most cases, these theories cannot reach the optimal solution. For this reason, in the present work, we test a range of penalty values. If the accuracy of the entire model is maximized under all the scenarios and the normality error is minimum within an imposed constraint, the value of the penalty is selected as a unique value for the whole model.

After finding the number of changepoints $K-1$ with the optimal penalty value, the position τ of each changepoint in every signal Y_i and the data points of each interval are known. Using this information, we find the second feature, which is the mean of the changepoint intervals' mean. To find this feature as a unique value, we calculate the mean of each interval. Subsequently, we obtain the mean of all the intervals' mean. The mathematical expression is described in Eq. (4.6):

$$\mu_{Y_i} = \frac{1}{K}\sum_{k=1}^{K} \overline{y}_{(\tau'_k,\tau_k]}. \tag{4.6}$$

Finally, the third feature is calculated, called the mean of the changepoint intervals' variance. We obtain the variance of each of the intervals of the signal Y_i. Afterwards, the mean of the intervals' variance is calculated using Eq. (4.7).

$$\hat{\sigma}^2_{Y_i} = \frac{1}{K}\sum_{k=1}^{K} \hat{\sigma}^2_{y_{(\tau'_k,\tau_k]}}. \tag{4.7}$$

For each measurement or row Y_i of the matrix $\mathbf{Y}$, three features represented by three single values are extracted. The matrix obtained after this sub-stage is represented by the letter $\mathbf{C}$ with dimensions $((r+d)$ x f), where f represents the number of features extracted for each vector Y_i. Each vector of the matrix $\mathbf{C}$ is denoted as C_i.
Equations (4.5), (4.6), and (4.7) extract features in the sub-stage of Changepoint detection. It is important to highlight that PELT (Pruned Exact Linear Time) algorithm has been used in this paper to find the number of changepoints and their optimal position for each signal. This algorithm guarantees exactness and relatively fast computational time because it is based on an iterative calculation. This algorithm uses dynamic programming with pruning to find the exact global optimum [54].

In this thesis, the assumptions to extract meaningful features using parametric changepoint have been verified. The first assumption is that each interval of a signal Y_i should follow a Gaussian distribution. To verify this assumption, Kolmogorov testing is applied

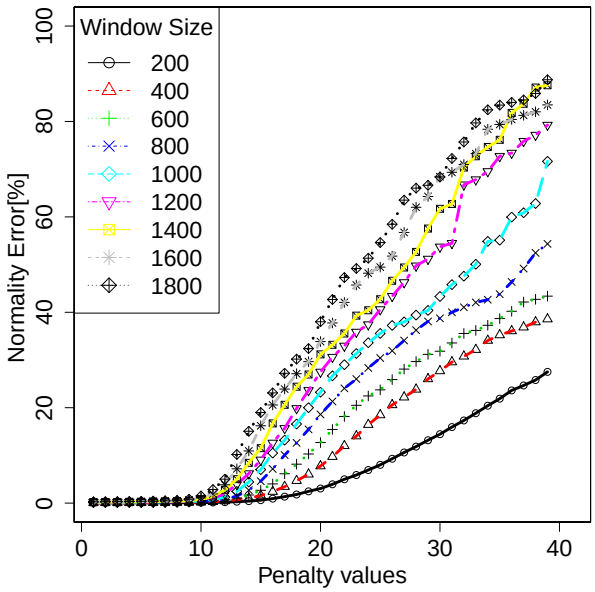
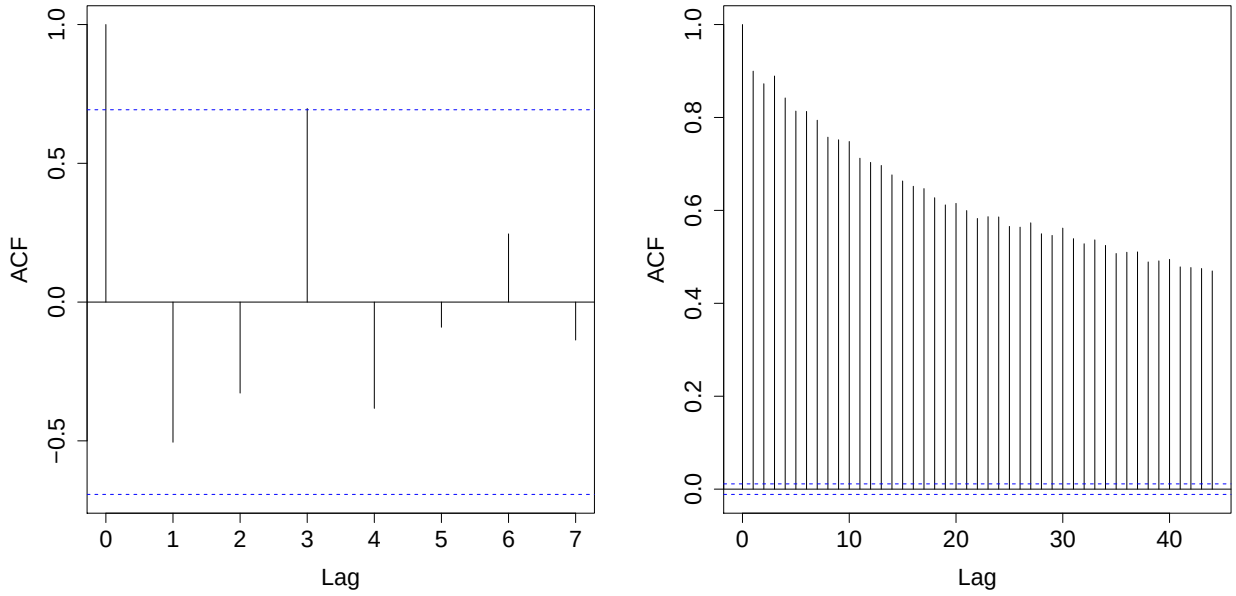


Figure 4.5: Normality error check

to all of the intervals, and we calculate an error that indicates how many intervals of the signal Y_i do not pass the test. This characteristic is used to select the penalty value and the window size. The validation is important because while the window size and penalty value increase, the error increases exponentially, as in Fig. 4.5. If the error is high, the assumption of extracting features with Changepoint detection is not valid.

To verify the second assumption, autocorrelation testing is applied to each interval found with Changepoint detection theory for each signal Y_i. In this case, we find a correlation between an original interval and a duplicated lagged signal of itself. A lagged signal means that the signal is offset $(1, 2, 3.., n/w)$. To illustrate, if lag 1 is taken, the signal will be offset 1 sample. With the lagged signal, we want to analyze whether the present sample depends on past samples. The mathematical expression of autocorrelation applied to an interval is given by Eq. 4.8, while the expression of autocorrelation applied to the entire signal is given by Eq. 4.9

$$ACF = \frac{\sum_{i=l+1+\tau'_k}^{\tau_k} (y_i - \overline{y}_{(\tau'_k,\tau_k]})(y_{i-l} - \overline{y}_{(\tau'_k,\tau_k]})}{\sum_{p=1+\tau'_k}^{\tau_k} (y_p - \overline{y}_{(\tau'_k,\tau_k]})^2} \tag{4.8}$$



(a) Interval Autocorrelation test

(b) Signal Autocorrelation test

Figure 4.6: Analysis of normality error, interval autocorrelation, and whole signal autocorrelation

$$ACF = \frac{\sum_{i=l+1}^{n} (y_i - \overline{y}_{[1,n]})(y_{i-l} - \overline{y}_{[1,n]})}{\sum_{p=1}^{n} (y_p - \overline{y}_{[1,n]})^2} \tag{4.9}$$

where l is the lag analyzed.

Figure 4.6(a) shows the results after applying autocorrelation to one of the intervals found with Changepoint theory, chosen randomly, of a Benign signal. The random variables included in this interval are independent of one another. In contrast, if autocorrelation is applied to the raw signal X_i directly without applying the Filtering and Changepoint detection sub-stages, a correlation is found to exist among all the variables that constitute the signal, as shown in Fig. 4.6(b).

4.1.2 Procedure 1: Training and Testing of the model.

This stage is composed of three sub-stages: Labeling, Normalization, and Training and Testing of the model.

Labeling

This sub-stage labels the features represented in the matrix **C**. The first r rows of **C** represent power measurements of benign signals, labelled 0. The rest of the rows of **C** depict malicious signals which, labelled 1. The resultant matrix after labeling is denoted by **L**, and it has dimension $(r + d)$x(f+1). **L** has an additional column for the labels. Each labeled vector of **L** is denoted by L_i.

Normalization

The first three columns of the matrix **L**, which represent the features, are normalized primarily to increase the algorithm's performance in the next sub-stage. The min-max normalization is executed per feature of **L**, and is expressed by Eq. (5.18)

$$N_j = \frac{L_i - min(L_j)}{max(L_j) - min(L_j)}, \tag{4.10}$$

where i represents each of the rows of matrix L while j denotes each column of L. The index j can take values between 1 to f, and i can take values between 1 to $(r+d)$.

After this sub-stage, we find a matrix N with dimensions $(r+d)$x$(f+1)$. Each row of the matrix N is represented as N_i

Training and testing

Until this sub-stage, the measurements of benign and malicious signals have been processed through stages and sub-stages, obtaining meaningfully normalized and labelled features.

In this sub-stage, supervised learning transforms this data into a classification model. In supervised learning, the data is separated into two datasets called the training and testing datasets. The training dataset trains a model, while the testing dataset verifies the performance of the created model.

In the present work, it is difficult to define optimal percentages by which to divide the data into training and testing datasets, since the number of samples is reduced. For this reason, Leave One Out Cross Validation (LOOCV) is used [95]. This technique uses all of the samples available to train and test the model. At the beginning, LOOCV excludes 1 sample of the whole data available, and we train the model with the rest of the data. The excluded sample is evaluated in the model, and the error is calculated with the label available. This process is repeated with all of the samples, to calculate the total error, which gives us the accuracy of the model [20]. This method is sensitive to outliers, but it is guaranteed that the model will learn features for each of the samples of the data.

Different classification algorithms, namely Support Vector Machine linear, Logistic Regression, and Naive Bayes, have been evaluated using LOOCV to obtain the accuracy and F1-measure of each model. In the present work, accuracy and F1-measure have the same values because the dataset is balanced.

A confusion matrix helps to calculate both accuracy metrics. This matrix has the outcomes of each classifier trained and tested. The dimension of this matrix depends on the number of classes of the problem according to $(numberofclasses)^2$ components. In this case, we have two classes available: malicious and benign samples. Hence, the confusion matrix has four components, as shown in Table 4.1.

	Predicted benign (0)	Predicted malicious(1)
Actual benign(0)	True positives(TP)	False Positives(FP)
Actual malicious(1)	False negatives(FN)	True negatives(TN)

Table 4.1: Confusion Matrix

Accuracy is defined by Eq. (4.11). This equation does not take into consideration False Positives and False Negatives for the score. Thus, accuracy is not sufficient to characterize a model because if the classifier predicts all the labels erroneously, the accuracy will be 100%.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.11}$$

F1-Measure is defined by Eq. (4.12). This metric is an indicator of the exactness and sensitivity of the classifier. Precision evaluates the proportion of True Positives and the sum of True positives and True Negatives, showing the exactness of the classifier. The recall relates the number of true positives over the number of true positives and false negatives. Thus, it shows the completeness of the classifier.

$$Accuracy = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.12}$$

F1-measure is a better metric than accuracy because almost all the components of the confusion matrix are used to calculate it.

4.1.3 Procedure 2: Detection procedure

This procedure shown in Fig. 4.2 tests unlabeled new measurements. In this case, we do not known if the measurement has malware running on it.

The feature extraction stage comprises two sub-stages. The sub-stage of Filtering uses the same value for w obtained in Procedure 1 to apply the moving average filter to the new signal. After this sub-stage, a vector Y is obtained with dimensions n/w.

Subsequently, the sub-stage of Changepoint detection extracts f features of the input signal using the same penalty value as in Procedure 1. It obtains a vector C with dimension f, where f=3.

The next stage is called Evaluation, and it only considers two sub-stages: Normalization and Evaluation. The min-max method is used to normalize the features of the new signal. After the features are normalized, the optimal classifier chosen in Procedure 1 is used to predict whether the normalized features of the new signal indicate anomalous behavior.

Chapter 5

Methodology 2

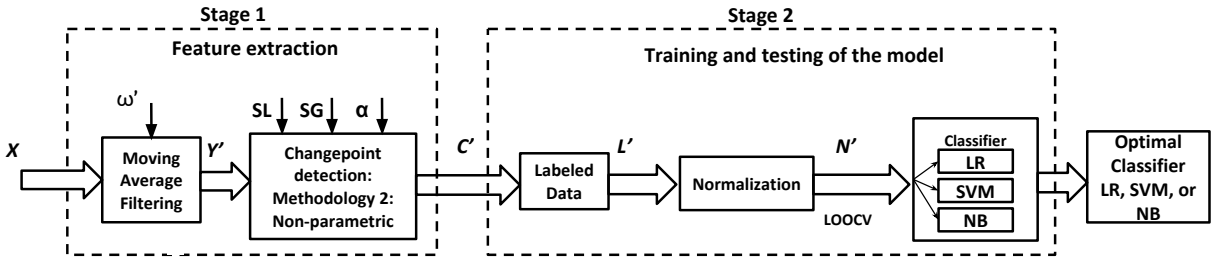
This chapter describes the non-parametric changepoint detection methodology used to detect malicious behavior. This methodology has the same stages and sub-stages as Methodology 1, as is seen in Figs. 5.4 and 5.1 . However, the way the features are extracted in the sub-stage of feature extraction is different to that in Methodology 1.

5.1 Methodology

This methodology has two Procedures as Methodology 1. Procedure 1 trains and tests the model, while Procedure 2 shows the flow that a new measurement has to follow to be classified as a malicious or non-malicious signal. Furthermore, the same notation has been used to name the signals as in Methodology 1. Each benign signal is denoted as follows, $B_i = (b_{i_1}, b_{i_2}, b_{i_3}, ..., b_{i_n})$, and this experiment is repeated r times. On the other hand, a malicious signal is denoted as $M_i = (m_{i_1}, m_{i_2}, m_{i_3}, ..., m_{i_n})$, and the experiment is repeated d times. All of the signals have been concatenated in a matrix $\mathbf{X}$ with dimensions $(r + d)xn$.

The stages and sub-stages of Methodology 2 are similar to those presented in Chapter 4. Hence, they are presented in a compact manner.

The first stage of Procedure 1 is called feature extraction, and it is composed of Filtering and non-parametric Changepoint Detection. The second stage is denoted as training and testing of the model. This stage fits three machine learning algorithms Support Vector Machine Linear, Logistic regression, and Naive Bayes with filtered and meaningful



Legend:
r: number of reference signals; d: number of malicious signals; n: number of observed values.
***Methodology 2: X**:(r+d) x n; **Y'**:(r+d)x(n/ω'); **C'**:(r+d) x f1; **L'**:(r+d)x(f1+1); **N'**:(r+d)x(f1+1); ω': window size; f1: features; SL:slide;*
SG:segment; α:alpha; LOOCV: Leave one out cross validation; LR: Logistic regression; SVM Linear: Support Vector Machine Linear; NB: Naive Bayes

Figure 5.1: Procedure 1: Data Preparation and Model Training and Testing

features. This stage is composed of Labeling, Normalization, and Training and Testing of the classifier.

Procedure 2, named Detection, has the goal of classifying a new unseen signal as malicious or benign. In this procedure, the sub-stage of labelling is omitted, and the current sample is tested in the optimal classifier chosen in Procedure 1.

The following sub-sections explain all the stages and sub-stages in detail.

5.1.1 Procedure 1: Feature Extraction.

Feature extraction is the most important stage in the methodology since this stage has a decisive impact on the final accuracy and F1-measure of the models. At the beginning of this stage, moving average filter eliminates noise from the original signal. Next, non-parametric changepoint detection obtains relevant training features.

Filtering.

This sub-stage is useful in eliminating interference introduced in the signal due to external sources such as electromagnetic or thermal noise. Additionally, the sampling rate of the power consumption is relatively high (near to 5000 samples/s). Thus, these noises can amplify its effect in the measurement. Each of the power consumption measurements represented by X_i passes through this sub-stage using Eq. (5.1)

$$Y_{ij} = \frac{1}{w'} \sum_{g=0}^{w'-1} X_{i_{w'*j+g}} \quad , \tag{5.1}$$

where:

- w' window size: This parameter determines how many values of the signal X_i are averaged. This parameter has been chosen considering the maximum F1-measure and the minimum time of detection reached with all of the stages of Procedure 1. It is noteworthy that in Methodology 1, the time of detection has not been considered in selecting the window size, because as is shown in [80], it is relatively low near to 1.1 seconds.
- X_i input power consumption: This signal represents the raw signal power consumption with dimensions $(1xn)$.
- Y_i output filtered signal: This signal is the resultant of Eq.5.1 with dimensions $(1x(n/w))$.
- g: This index is the current position of signal X_i.
- j: This index is used for the new signal filter signal Y_i. j takes values between 0 and $trunc(n/w)$.

The filtered signals are concatenated in a matrix **Y'** with dimensions $(r + d)$x(n/w'). In the present chapter, the term "window" describes the set of values grouped to filter the signal. The term "slice (SL)" describes the datapoints grouped in non-parametric changepoint detection. Finally, the term "segment (SG)" corresponds to consecutive groups of slices.

Non-parametric Changepoint detection

The theory of non-parametric changepoint detection assigns scores to the value of sudden changes in a stochastic time-series signal, using divergence-based dissimilarity and relative Pearson direct density-ratio theories. If there is a significant change in the signal, the value obtained is high. Conversely, if the change is not important, it is ranked with a low value. Figure 5.2 shows how each point of the signal is ranked in a new scale depending on its strength. The limits are bounded in a range.

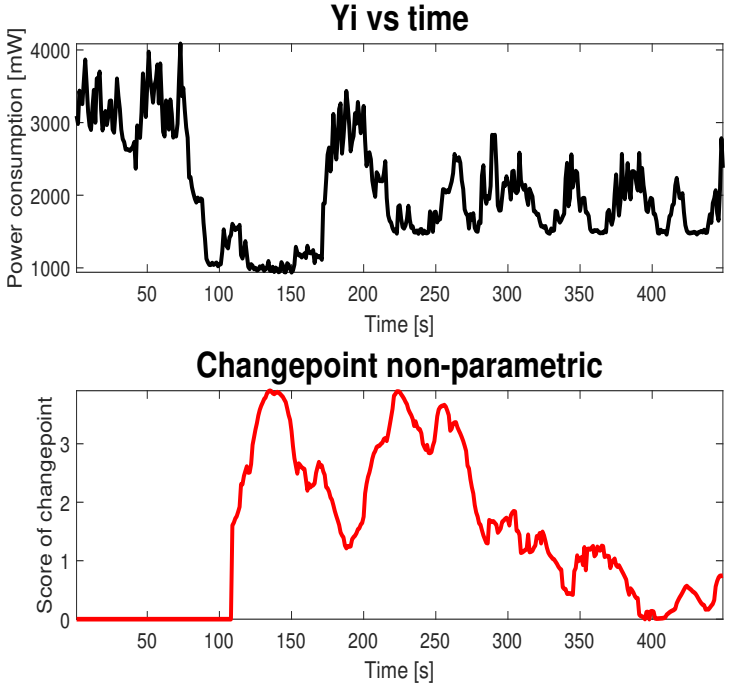


Figure 5.2: Non-Parametric changepoint detection applied to one part of the signal Y_i named the Resultant signal

This theory is called non-parametric changepoint because this theory does not assume any distribution or independence of random variables of retrospective segments. In the present work, the following features have been found for each signal using non-parametric changepoint detection:

- $Feature1'$: Cumulative sum of changepoints' scores.
- $Feature2'$: Mean of the changepoints' scores.
- $Feature3'$: Variance of the changepoints' scores.
- $Feature4'$: Mean of the whole signal.
- $Feature5'$: Variance of the whole signal.

We select $Feature1'$ as relevant because we assume that malware acts using different hardware resources of the smartphone. This usage will be reflected as a high cumulative

sum of changepoints' scores. $Features2'$ and $Feature3'$ have been selected because these summarize the main statistical properties of the Resultant Signal after we have applied non-parametric changepoint detection. Finally, $Feature4'$ and $Feature5'$ are also useful for the analysis because they show the general statistical properties for the whole signal.

To understand how the Resultant Signal is generated, we review the theory of divergence-based dissimilarity and relative Pearson direct density-ratio.

Divergence-based calculates the dissimilarity between two retrospective segments using the ratio of their unknown probability distributions [75].

The ratio of two retrospective segments $f(Y)$ and $f'(Y)$ is defined by Eq. (5.2). It is necessary to highlight that knowing the ratio of both segments does not mean that we can infer the distribution of $f(Y)$ and $f'(Y)$, because the relation does not guarantee a unique solution.

$$ratio = \frac{f(Y)}{f'(Y)} \tag{5.2}$$

To explain how the non-parametric Changepoint detection technique works, we use the following notation to describe Fig. 5.3:

- Slice SL: set of datapoints of the signal X_i grouped according to the value of this variable.
- Segment SG: set of slices associated in a group.
- o: defined as n/w'.
- Z: vector containing all datapoints grouped by the value of SL.
- **Z**: matrix that has all vectors Z.

To illustrate, Fig. 5.3 shows a time-series signal composed of $o = n/w$ elements. If the value of the slice SL=4, four datapoints are associated in a group. The datapoints y_{i_0}, y_{i_1}, y_{i_2}, and y_{i_3} are grouped in a vector denoted as Z_{i_0}. The slice advances one position at a time to the right. Therefore, the next element will be named Z_{i_1}, and it contains the elements y_{i_1}, y_{i_2}, y_{i_3}, and y_{i_4}. This process is repeated until all of the samples are grouped. All vectors Z are concatenated in a matrix **Z**.
A Segment is a set of ordered slices grouped. In Fig. 5.3 the value of the segment is 2.

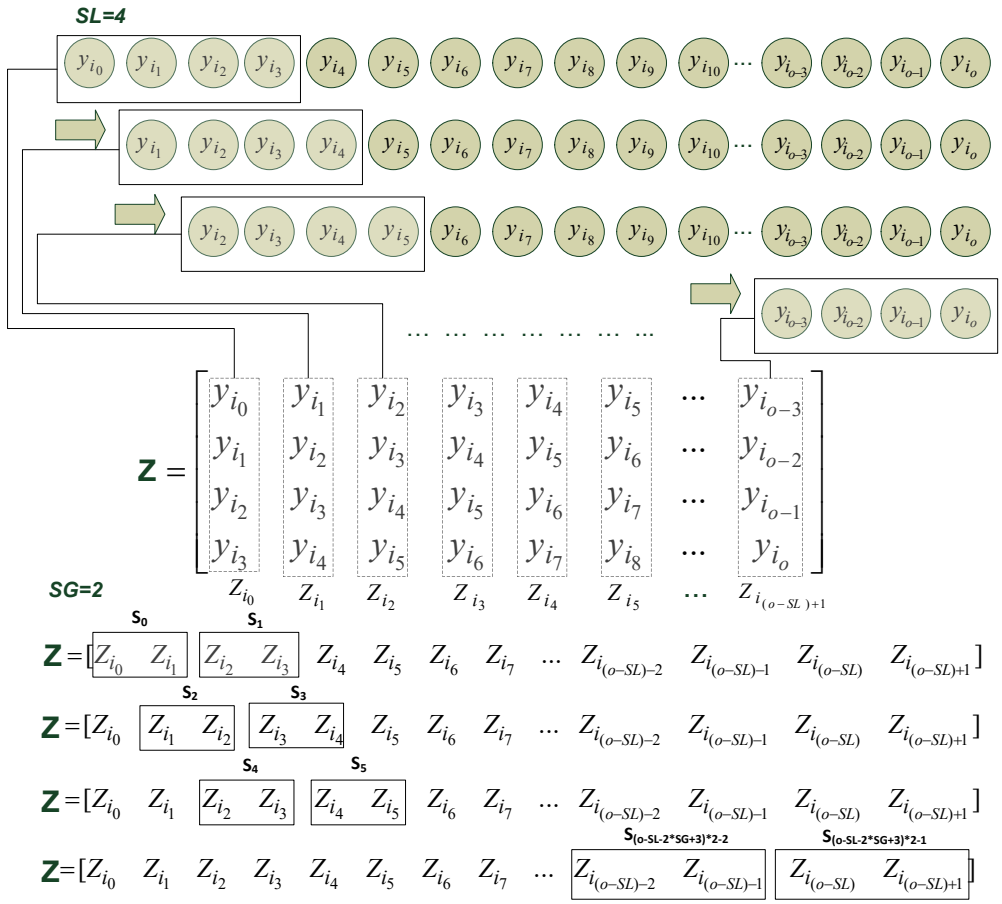


Figure 5.3: Notation and example of non-parametric changepoint detection

Hence, 2 slices Z_{i_0} and Z_{i_1} are grouped and denoted with seg_0. Next, 2 slices Z_{i_2} and Z_{i_3} form another group named seg_1. The process to obtain the scores of a changepoint is based on finding a dissimilarity measure between pairs of probability segments, using symmetric divergence-based approach given by Eq. (5.3).

$$D(P(\mathbf{S}_q)||P(\mathbf{S}_{q+1})) + D(P(\mathbf{S}_{q+1})||P(\mathbf{S}_q)) \tag{5.3}$$

The concept of divergence according to Ali et al. [10] is shown in Eq. (5.4). It is defined

as f-divergence.

$$D(P(\mathbf{S}_q)||P(\mathbf{S}_{q+1})) = \int P(\mathbf{S}_{q+1})\, f\left(\frac{P(\mathbf{S}_q)}{P(\mathbf{S}_{q+1})}\right) d\mathbf{S} \tag{5.4}$$

In Eq. (5.4), we assume that $P(\mathbf{S}_q)$ and $P(\mathbf{S}_{q+1})$ are strictly positive, and the function f is a convex function where $f(1) = 0$ [75]. In the present paper, we use Pearson Divergence shown in Eq. (5.5), which replaces to the function f in Eq. (5.4).

$$PE(P(\mathbf{S}_q)||P(\mathbf{S}_{q+1})) = \frac{1}{2}\int P(\mathbf{S}_{q+1}) \left(\frac{P(\mathbf{S}_q)}{P(\mathbf{S}_{q+1})} - 1\right)^2 d\mathbf{S} \tag{5.5}$$

$P(\mathbf{S}_q)$ and $P(\mathbf{S}_{q+1})$ can be estimated with estimation theory using a Naive approach. Nonetheless, this estimation can be considered a complex task. Therefore, many authors such as Sugiyama et al. [92] and Kanamori et al. [58] develop a direct estimation of the ratio without the need to estimate the probabilities separately. Both methods are unbounded; hence, the ratio of the segments' probabilities can be infinite in some cases. In this paper, to tackle this problem, we use a method called RuLSIF proposed by Yamada et al. [75]. It uses a parameter alpha to bound the ratio, as shown in Eq. (5.6)

$$\begin{aligned} PE(P(\mathbf{S}_q)||P(\mathbf{S}_{q+1})) := \\ PE(P(\mathbf{S}_q)||\alpha P(\mathbf{S}_q) + (1-\alpha)P(\mathbf{S}_{q+1})) = \\ \int P_\alpha(\mathbf{S}_{q+1}) \left(\frac{P(\mathbf{S}_q)}{P_\alpha(\mathbf{S}_{q+1})} - 1\right)^2 d\mathbf{S} \end{aligned} \tag{5.6}$$

where

$$P_\alpha(\mathbf{S}_{q+1}) = \alpha P(\mathbf{S}_q) + (1-\alpha)P(\mathbf{S}_{q+1}) \tag{5.7}$$

If α takes the value of 0, the divergence PE is the same as in Eq. (5.5). Equation (5.8) defines the relationship between two retrospective probability segments

$$r_\alpha(\mathbf{S}) = \frac{P(\mathbf{S}_q)}{\alpha P(\mathbf{S}_q) + (1-\alpha)P(\mathbf{S}_{q+1})} \tag{5.8}$$

According to [75], we can model the ratio $\frac{P(\mathbf{S}_q)}{P(\mathbf{S}_{q+1})}$ using a kernel model given by Eq. (5.9). The ratio is denoted as the function $g(\mathbf{S};\theta)$ in Eq. (5.9),

$$g(\mathbf{S},\theta) := \sum_{l=1}^{cv} \theta_l K(\mathbf{S},\mathbf{S}_l) \tag{5.9}$$

where K is defined as a Gaussian kernel shown in Eq. (5.10)

$$K(\mathbf{S}_q,\mathbf{S}_{q+1}) = exp\left(-\frac{||\mathbf{S}_q - \mathbf{S}_{q+1}||^2}{2\sigma^2}\right) \tag{5.10}$$

To approximate $g(\mathbf{S};\theta)$ using the samples available, it is necessary to minimize the square loss shown in Eq. (5.11)

$$J(\mathbf{S}) = \frac{1}{2}\int P_\alpha(\mathbf{S}_{q+1})(r_\alpha(\mathbf{S}) - g(\mathbf{S};\theta))^2 d\mathbf{S} \tag{5.11}$$

After minimizing Eq. 5.11, we calculate the estimator denoted by Eq. 5.12

$$g(\hat{\mathbf{S}}) := \sum_{l=1}^{cv} \theta_l K(\mathbf{S},\mathbf{S}_l) \tag{5.12}$$

where cv means folds of cross-validation.

Finally, the estimator is replaced in Eq. (5.6), and we find the changepoint score for the segments $\mathbf{S}_q$ and $\mathbf{S}_{q+1}$. As mentioned, we use symmetric divergence to find the score of each changepoint denoted by Eq. 5.4. Thus, it is necessary to find the changepoint score from $\mathbf{S}_{q+1}$ to $\mathbf{S}_q$. Both scores are sumed up. This process is repeated with all of the segments from $\mathbf{S}_0$ to $\mathbf{S}_{(o-SL-2*SG+3)*2-1}$. The result is a vector denoted as $SC = (sc_0, sc_1, sc_2, sc_3, ..., sc_{n/a})$ with the scores of changepoints for each datapoint of the signal Y_i. To find feature 1', 2', 3', 4', and 5, we use Eqs. (5.13), (5.14), (5.15), (5.16), (5.17).

$$f1' = \sum_{i=0}^{n/a} sc_i \tag{5.13}$$

$$f2' = \frac{1}{n/a} \sum_{i=0}^{n/a} sc_i \tag{5.14}$$

$$f3' = \sum_{i=0}^{n/a} \frac{(sc_i - f2')^2}{n/a} \tag{5.15}$$

$$f4' = \frac{1}{n} \sum_{i=0}^{n} y_i \tag{5.16}$$

$$f5' = \sum_{i=0}^{n} \frac{(y_i - f4)^2}{n} \tag{5.17}$$

The values of the slice SL, α, and segment SG have been chosen with the wrapper approach, which means considering the optimal value to obtain the best F1-measure and time of detection for the entire model.

If r benign and d malicious signals pass through the sub-stage filtering and changepoint methodology 2, we obtain a matrix named **C'**, with the dimensions $(r+d)x5$. Each power consumption measurement is denoted by C_i' after this sub-stage.

5.1.2 Procedure 1: Training and Testing of the model.

The sub-stages of training and testing the model are the same as in Methodology 1. These sub-stages are: Labeling, Normalization, and evaluation of the classifier.

Labeling

We label each element of the matrix C' in this sub-stage. There are two labels possible to characterize each signal C_i'. If the signal is benign, it is labelled 0. In contrast, if the signal is malicious, it is labelled 1. The resultant matrix after labeling is denoted by **L'**, with dimension $(r + d)$x(6).

Normalization

To increase the performance of the classifiers of the next sub-stage, we use min-max normalization. The first five columns of matrix **L'** are normalized according to max and min value for each feature or column. Equation (5.18) shows the mathematical formula applied to each feature.

$$N'_j = \frac{L'_i - min(L'_j)}{max(L'_j) - min(L'_j)}, \tag{5.18}$$

- i: represents each row of the matrix **L'**.
- j: represents each feature or column of **L'**

After normalization, a matrix denoted as **N'** is calculated.

Training and testing

In the present paper, we use supervised learning to train and test 3 classifiers: Support Vector Machine Linear, Naive Bayes, and Logistic Regression. These classifiers are deterministic. Therefore, every time that the classifier runs, we obtain the same result in the output. Deterministic classifiers were chosen to provide solid and trustworthy results for the methodologies.

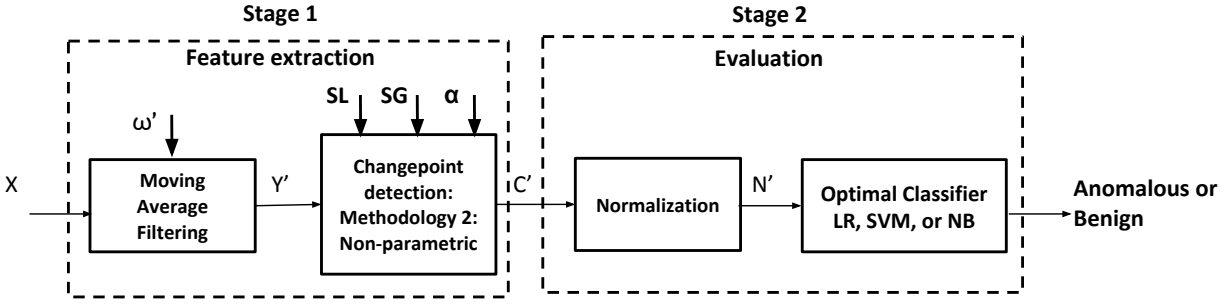
To train and test each classifier, we use Leave One Out Cross Validation (LOOCV) because this cross-validation technique evaluates all data samples providing realistic and precise results of F1-measure of each model.

In Methodology 2, we select the optimal classifier which performs better in terms of F1-measure and in the least detection time.

5.1.3 Procedure 2: Detection procedure

This procedure has the goal of classifying a new unlabeled power consumption measurement as benign or malicious. This Procedure is shown in Fig. 4.2.

Procedure 2 has the same sub-stages as Procedure 1 in feature extraction stage. We filter the power trace with the window size w' selected in Procedure 1. The filtered signal



Legend:
n: number of observed values
Methodology 2: *X: Suspicious Signal (1 x n); Y':(1 x (n/ω')); C':(1 x f1); N':(1 x f1); ω': window size; f1: features; SL:slide;* SG:segment; α:alpha; LOOCV: Leave one out cross validation; LR: Logistic regression; SVM Linear: Support Vector Machine Linear; NB: Naive Bayes

Figure 5.4: Procedure 2: Detection procedure used to classify an unlabeled signal as malicious or not malicious

passes through the non-parametric changepoint detection sub-stage to extract meaningful features. These features are the inputs to the stage named Evaluation.
The evaluation stage only has two sub-stages, normalization and evaluation, of the optimal classifier. In this case, the features extracted in stage 1 are normalized using min-max normalization. The normalized features are evaluated in the optimal classifier chosen in Procedure 1. After this stage, the signal can be classified either a malicious or benign signal.

Chapter 6

Results

This chapter presents the results for Methodology 1 and 2 in terms of F1-Measure applied to dataset 1 and 2. Also, this section describes the normality error for Methodology 1 and the time of execution for both methodologies. Finally, we compare the results of both methodologies with those from previous work.

In Methodology 1, the window size and the penalty value are the tuning parameters to obtain the lowest normality error and the best F1-measure. Reference [80] describes the optimal values of both parameters. The values are:

- Moving average filtering sub-stage: Window size w=50
- Parametric changepoint sub-stage: Penalty value p=1

The results of F1-measure applied for dataset one are shown in Fig. 6.1. The lowest F1-measure value is ∼78, when duty cycle is 12%, and the emulated malware is only uploading the image to a server. The highest average F1-measure is 99.46% for scenario 1, in which the emulated malware downloads a file during the entire duty cycle. The worst average F1-measure is 95.01%, when the emulated malware uploads an image 25% of the duty cycle time and downloads a file 75% of the same time. The normality error is ∼0 in all the scenarios, and there is no correlation among random variables into an interval. Therefore, we can conclude that the assumptions are accomplished.

In Methodology 2, the window size, alpha, SL, SG, and cross-validation are the tuning parameters to obtain the best F1-measure and the less detection time. The following parameters' values have been evaluated in Methodology 2:

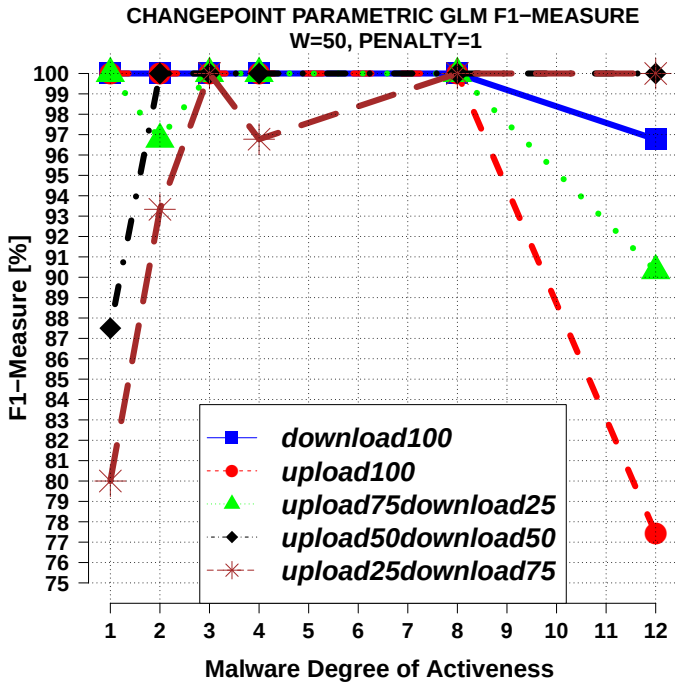


Figure 6.1: F1-measure results for Methodology 1 evaluating dataset 1

- Moving average filtering sub-stage: Window size $w' = [50, 100, 500, 1000]$
- Non-parametric changepoint sub-stage: $\alpha = [0.2, 0.4, 0.6, 0.8]$, $SL = [10, 50, 100]$, $SG = [10, 30, 50]$, and $cv = 5$

The optimal parameters selected are w'=1000, $\alpha = 0.4$, $SL = 10$, $SG = 10$, and $cv = 5$ to obtain the best F1-measure and the least detection time.

Figure 6.2 shows the results after applying Methodology 2 to dataset 1. The highest F1-measure value is 100% when the actions of downloading and uploading are equally distributed in time during duty cycle. The lowest F1-measure value is 96.66% when the emulated malware uploads an image 100% of duty cycle time.

Parametric and non-parametric methodologies perform similarly in terms of average F1-measure in all the scenarios of dataset 1. The average F1-measure in all the scenarios is 97.29% for the parametric and 97.82% for the non-parametric approaches, demonstrating that both are appropriate methodologies for detecting emulated malware.

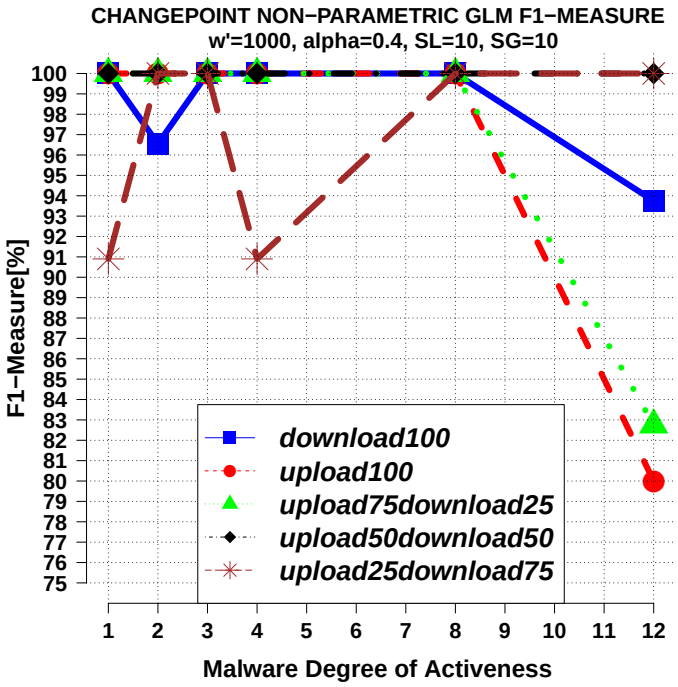


Figure 6.2: F1-measure results Methodology 2 evaluating dataset 1.

Neither Methodology 1 nor Methodology 2 performs with high F1-measure in 12% duty cycle. The average F1-measure for the parametric is 92%, and for the non-parametric is 91.3% in this duty cycle.

The results of both methodologies have been compared with the methodologies in [104], [55], and [56]. This analysis only compares the results in which the emulated malware is downloading a file from the Internet with Dut=1%, 2%, 3%, 4%, 8% and 12% because these papers do not analyze the five additional scenarios described in Figs. 6.1 and 6.2.

A down-sampling stage reduces the dimension of each signal of the dataset of scenario 1 to train and test the methodology proposed in [104]. Afterwards, MFCC extracts 12 coefficients for each signal. Finally, GMM discriminates between non-malicious and malicious signals training these coefficients. Figure 6.3 shows the results for each methodology.

The average F1-measure accuracy reached for methodology 1 is ∼99.45%, for methodology 2 ∼98.55%, for reference [55] ∼91%, for reference [56] ∼85%, and for reference [104] ∼69%. The proposed methodologies clearly surpass all other methodologies in terms of average F1-measures. It is noteworthy that [56] and [55] achieve better accuracy than Methodology 1 and 2 when emulated malware has a degree of activeness of 12%. However,

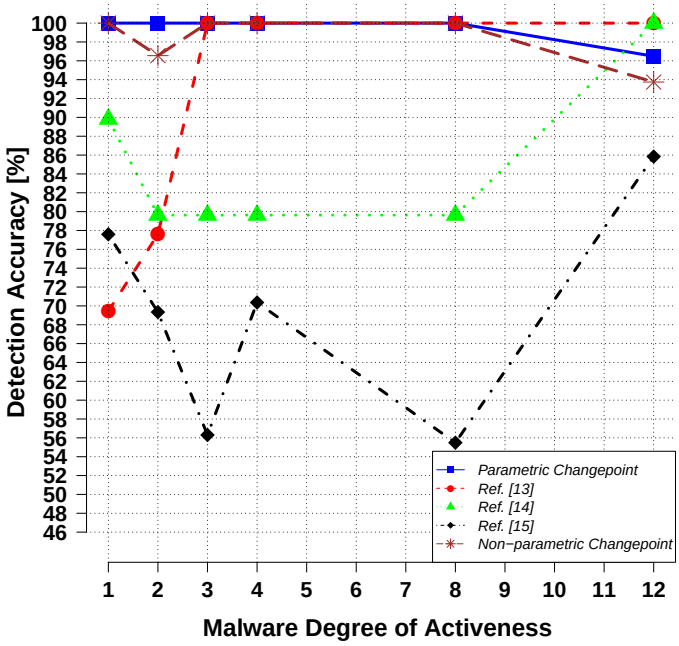


Figure 6.3: Comparison.

the difference is less than 4%. We are interested in a model that can generalize well in all the scenarios. Therefore, we can state that methodologies 1 an d 2 surpass the F1-measure results of the methodologies in [55], [56], and [104].

Finally, Methodologies 1 and 2 evaluate dataset 2. The results are shown as a bar chart for each App in Fig. 6.4. It is clear that non-parametric changepoint detection, with an average F1-measure of 96.09% for all Apps, surpasses the parametric approach by ∼8%.

Methodology 2 F1-measure surpasses Methodology 1 in both datasets. In the emulated malware dataset, non-parametric methodology outperforms by ∼ 0.53% Methodology 1. In real malware dataset, non-parametric surpasses parametric methodology by ∼8%. However, the detection time for the parametric approach is 1s, while that of non-parametric is 15s. The difference in detection time between the two methodologies is small. Therefore, we suggest using non-parametric changepoint detection to detect malware in smartphones. Furthermore, non-parametric changepoint detection does not use any assumptions.

In Chapters 1 and 2, we highlight that both methodologies would be tested with a different input from the power consumption to justify that they are data agnostic. Data agnostic means that these can use a different input to detect anomalous behavior. For

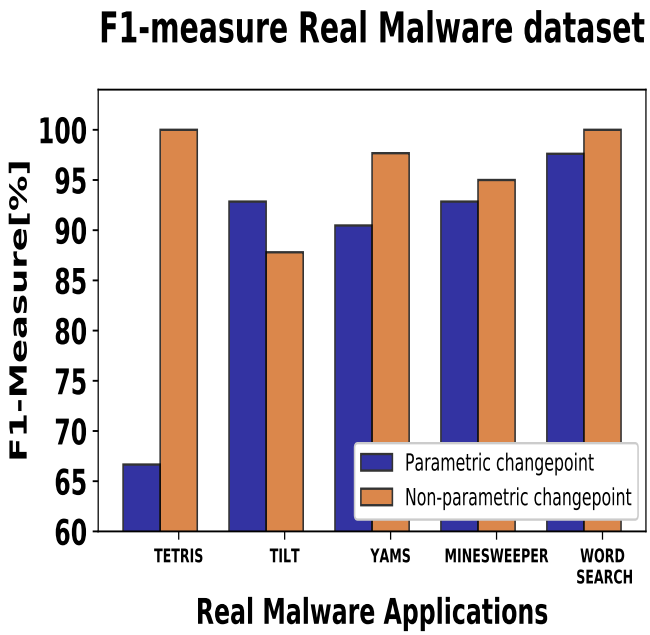


Figure 6.4: Results of F1-measure for Methodology 1 and 2 applied to dataset 2 of real malware

this purpose, methodologies 1 and 2 evaluate a small network traffic dataset created using Wireshark. A malicious App called Tilt from Drebin and a benign App named Tilt from Play Store are used to create the dataset. While Wireshark is collecting the network traffic, Droidbot and ADB commands are controlling the App emulating user's inputs. Fifteen measurements of a benign App and 15 of a malicious App have been taken.

Figure 6.5 shows the results for Tilt App using the power consumption and the network traffic as inputs. There is no a result for non-parametric changepoint detection of the input network traffic because the network traffic time signal is sparse(it contains many zeros). When Methodology 2 evaluates the network traffic signal, there is a mathematical error because the relation of two consecutive slices can be infinite or zero due to the fact that the divisor of the relation can be 0. Therefore, we can conclude that Methodology 2 is not useful in detecting malicious behavior with sparse signals.

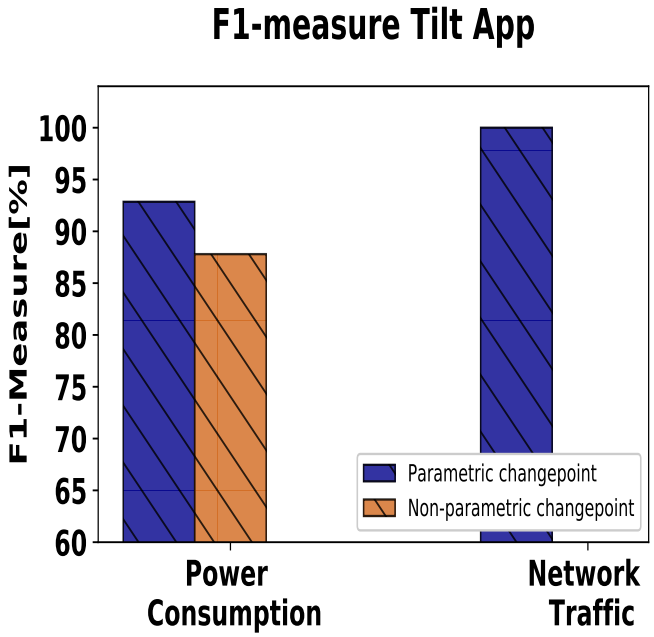


Figure 6.5: Results of F1-measure for Methodologies 1 and 2 applied to dataset 2 of real malware

Chapter 7

Conclusions and Future Work

This thesis has presented two methodologies to detect emulated and real malware in Android smartphones using power consumption. Both methodologies show satisfactory accuracy in terms of F1-measure over ∼95% to detect malware, although non-parametric methodology shows slightly better results without assumptions but with a greater detection time than a parametric approach. It is noteworthy that both methodologies have a weakness in detecting malware with the highest percentage of duty cycle 12%. For this reason, it is necessary to tackle this problem as future work. When the duty cycle is greater than 12%, the statistical properties of the whole signal will change drastically. Thus, we can check at the beginning only the mean and the variance to detect malware. If the signal is classified as benign, we can run the Methodology 1 and 2 to detect malware with lower duty cycles. Therefore, we can conclude that malware with duty cycle over 12% are easier to detect.

We can highlight that the present work is unique in terms of validation datasets if we compare it with other works such as [61] [55] [104] [28], which use only emulated malware to validate their methodologies. Furthermore, if we compare the present work with previous works [80], [55] that use emulated malware, we can conclude that we use 75% more power measurements to test the methodologies with emulated malware. However, the number of real Apps is still a small power consumption dataset. Therefore, as future work, we can use a hundred or more real malware Apps.

Finally, we verify that parametric changepoint detection methodology can handle sparse and not sparse signals to detect malicious behavior. Hence, Methodology 1 can be considered a data-agnostic methodology. However, the non-parametric methodology exhibits an issue handling sparse signals. Thus, we can look for a method to solve this issue.

References

[1] Proof-of-concept ransomware for smart thermostats demoed at defcon. *[Online]. Available: https://boingboing.net/2016/08/08/proofof-concept-ransomware-fo.html*, 2016.

[2] J. Abawajy and A. Kelarev. Iterative classifier fusion system for the detection of android malware. *IEEE Transactions on Big Data*, pages 1–1, 2018.

[3] Zeinab Abbasi, Mehdi Kargahi, and Morteza Mohaqeqi. Anomaly detection in embedded systems using simultaneous power and temperature monitoring. In *2014 11th International ISC Conference on Information Security and Cryptology*, pages 115–119. IEEE, 2014.

[4] Accenture. People-based attacks have increased the most).

[5] Fabio Aiolli, Mauro Conti, Ankit Gangwal, and Mirko Polato. Mind your wallet's privacy: identifying bitcoin wallet apps and user's actions through network traffic analysis. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1484–1491. ACM, 2019.

[6] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[7] Luai Al Shalabi, Zyad Shaaban, and Basel Kasasbeh. Data mining: A preprocessing engine. *Journal of Computer Science*, 2(9), 2006.

[8] M. S. Alam and S. T. Vuong. Random forest classification for detecting android malware. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 663–669, Aug 2013.

[9] K. Albrecht and L. Mcintyre. Privacy nightmare: When baby monitors go bad [opinion]. *IEEE Technology and Society Magazine*, 34(3):14–19, Sep. 2015.

[10] Syed Mumtaz Ali and Samuel D Silvey. A general class of coefficients of divergence of one distribution from another. *Journal of the Royal Statistical Society: Series B (Methodological)*, 28(1):131–142, 1966.

[11] Aljawharah Alnasser, Hongjian Sun, and Jing Jiang. Cyber security challenges and solutions for v2x communications: A survey. *Computer Networks*, 151:52–67, 2019.

[12] A. Alzahrani, A. Alshehri, H. Alshahrani, R. Alharthi, H. Fu, A. Liu, and Y. Zhu. Randroid: Structural similarity approach for detecting ransomware applications in android platform. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*, pages 0892–0897, May 2018.

[13] Mohammed K Alzaylaee, Suleiman Y Yerima, and Sakir Sezer. Improving dynamic analysis of android apps using hybrid test input generation. In *2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*, pages 1–8. IEEE, 2017.

[14] B. Amos, H. Turner, and J. White. Applying machine learning classifiers to dynamic android malware detection at scale. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1666–1671, July 2013.

[15] IoT analytics. State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating).

[16] Eirini Anthi, Lowri Williams, Małgorzata Słowińska, George Theodorakopoulos, and Pete Burnap. A supervised intrusion detection system for smart home iot devices. *IEEE Internet of Things Journal*, 2019.

[17] Alessio Antonini, Federico Maggi, and Stefano Zanero. A practical attack against a knx-based building automation system. In *ICS-CSR*, 2014.

[18] Abdullahi Arabo and Bernardi Pranggono. Mobile malware and smart device security: Trends, challenges and solutions. In *2013 19th international conference on control systems and computer science*, pages 526–531. IEEE, 2013.

[19] K. Ariyapala, H. G. Do, H. N. Anh, and et. all. A host and network based intrusion detection for android smartphones. In *30th Int. Conf. on Advanced Info. Net. and Apps Workshops (WAINA)*, March 2016.

[20] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

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

[22] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu. Samadroid: A novel 3-level hybrid malware detection model for android operating system. *IEEE Access*, 6:4321–4339, 2018.

[23] Amin Azmoodeh, Ali Dehghantanha, Mauro Conti, and Kim-Kwang Raymond Choo. Detecting crypto-ransomware in iot networks based on energy consumption footprint. *Journal of Ambient Intelligence and Humanized Computing*, 9(4):1141–1152, Aug 2018.

[24] Meriem Bettayeb, Qassim Nasir, and Manar Abu Talib. Firmware update attacks and security for iot devices: Survey. In *Proceedings of the ArabWIC 6th Annual International Conference Research Track*, page 4. ACM, 2019.

[25] Tamara Bonaci, Jeffrey Herron, Tariq Yusuf, Junjie Yan, Tadayoshi Kohno, and Howard Jay Chizeck. To make a robot secure: An experimental analysis of cyber security threats against teleoperated surgical robots. *arXiv preprint arXiv:1504.04339*, 2015.

[26] Andreas Brauchli and Depeng Li. A solution based analysis of attack vectors on smart home systems. In *2015 International Conference on Cyber Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pages 1–6. IEEE, 2015.

[27] R. Bridges, J. Hernández Jiménez, J. Nichols, K. Goseva-Popstojanova, and S. Prowell. Towards malware detection via cpu power consumption: Data collection design and analytics. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1680–1684, Aug 2018.

[28] L. Caviglione, M. Gaggero, J. Lalande, W. Mazurczyk, and M. Urbański. Seeing the unseen: Revealing mobile malware hidden communications via energy consumption and artificial intelligence. *IEEE Transactions on Information Forensics and Security*, 11(4):799–810, April 2016.

[29] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[30] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, volume 4, pages 447–462. San Francisco, 2011.

[31] Patrick Shicheng Chen, Shu-Chiung Lin, and Chien-Hsing Sun. Simple and effective method for detecting abnormal internet behaviors of mobile devices. *Information Sciences*, 321:193–204, 2015.

[32] Luigi Coppolino, Valerio DAlessandro, Salvatore DAntonio, Leonid Levy, and Luigi Romano. My smart home is under attack. In *2015 IEEE 18th International Conference on Computational Science and Engineering*, pages 145–151. IEEE, 2015.

[33] Baojiang Cui, Haifeng Jin, Giuliana Carullo, and Zheli Liu. Service-oriented mobile malware detection system based on mining strategies. *Pervasive and Mobile Computing*, 24:101–116, 2015.

[34] M. Curti, A. Merlo, M. Migliardi, and S. Schiappacasse. Towards energy-aware intrusion detection systems on mobile devices. In *Int. Conf. on High Performance Computing Simulation (HPCS)*, July 2013.

[35] R. G. d. S. Ramos, P. R. L., and J. V. d. M. Cardoso. Anomalies detection in wireless sensor networks using bayesian changepoints. In *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 384–385, Oct 2016.

[36] B. Dixon and S. Mishra. Power based malicious code detection techniques for smartphones. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 142–149, July 2013.

[37] H. Dogan, D. Forte, and M. M. Tehranipoor. Aging analysis for recycled fpga detection. In *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 171–176, Oct 2014.

[38] Ali El Attar, Rida Khatoun, and Marc Lemercier. A gaussian mixture model for dynamic detection of abnormal behavior in smartphone applications. In *2014 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 1–6. IEEE, 2014.

[39] R. Elnaggar, K. Chakrabarty, and M. B. Tahoori. Hardware trojan detection using changepoint-based anomaly detection techniques. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 1–14, 2019.

[40] Rana Elnaggar, Krishnendu Chakrabarty, and Mehdi B Tahoori. Hardware trojan detection using changepoint-based anomaly detection techniques. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

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

[42] Jason Flinn and Mahadev Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–10. IEEE, 1999.

[43] Huber Flores, Jonatan Hamberg, Xin Li, Titti Malmivirta, Agustin Zuniga, Eemil Lagerspetz, and Petteri Nurmi. Evaluating energy-efficiency using thermal imaging. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 147–152. ACM, 2019.

[44] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi. A methodology for detection and estimation of software aging. In *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257)*, pages 283–292, Nov 1998.

[45] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. In *Annual Cryptology Conference*, pages 444–461. Springer, 2014.

[46] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[47] GSMA. The mobile economy 2018, 2018.

[48] Dai-Fei Guo, Ai-Fen Sui, Yi-Jie Shi, Jian-Jun Hu, Guan-Zhou Lin, and Tao Guo. Behavior classification based self-learning mobile malware detection. *JCP*, 9(4):851–858, 2014.

[49] Zhongyuan Hau and Emil C Lupu. Exploiting correlations to detect false data injections in low-density wireless sensor networks. In *Proceedings of the 5th on Cyber-Physical System Security Workshop*, pages 1–12. ACM, 2019.

[50] Kaylea Haynes, Idris A Eckley, and Paul Fearnhead. Efficient penalty search for multiple changepoint problems. *arXiv preprint arXiv:1412.3617*, 2014.

[51] Daojing He, Sammy Chan, and Mohsen Guizani. Mobile application security: malware threats and defenses. *IEEE Wireless Communications*, 22(1):138–144, 2015.

[52] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks–practical examples and selected short-term countermeasures. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2008.

[53] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton. Software rejuvenation: analysis, module and applications. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 381–390, June 1995.

[54] Brad Jackson, Jeffrey Scargle, David Barnes, and et. all. An algorithm for optimal partitioning of data on an interval. *IEEE Signal Processing Letters*, 12(2):105–108, 2005.

[55] R Soundar Raja James, Abdurhman Albasir, Kshirasagar Naik, Mohamed-Yahia Dabbagh, Prajna Dash, M Zamani, and Nishith Goel. Detection of anomalous behavior of smartphones using signal processing and machine learning techniques. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7. IEEE, 2017.

[56] Robin James, Abdurhman Albasir, Kshirasagar Naik, and et. all. A power signal based dynamic approach to detecting anomalous behavior in wireless devices. In *Proceedings of the 16th ACM Int. Symposium on Mobility Management and Wireless Access MobiWac'18*, 2018.

[57] Xuxian Jiang and Yajin Zhou. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*, pages 95–109. IEEE, 2012.

[58] Takafumi Kanamori, Shohei Hido, and Masashi Sugiyama. A least-squares approach to direct importance estimation. *Journal of Machine Learning Research*, 10(Jul):1391–1445, 2009.

[59] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

[60] H. Kim, K. G. Shin, and P. Pillai. Modelz: Monitoring, detection, and analysis of energy-greedy anomalies in mobile handsets. *IEEE Transactions on Mobile Computing*, 10(7):968–981, July 2011.

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

[62] TaeGuen Kim, BooJoong Kang, Mina Rho, and et. all. A multimodal deep learning method for android malware detection using various features. *IEEE Trans. on Info. Forensics and Security*, 14(3), 2019.

[63] Mikkel Baun Kjærgaard and Henrik Blunck. Unsupervised power profiling for mobile devices. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 138–149. Springer, 2011.

[64] David C Klonoff. Cybersecurity for connected diabetes devices. *Journal of diabetes science and technology*, 9(5):1143–1147, 2015.

[65] Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[66] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462. IEEE, 2010.

[67] Harry Kurniawan, Yusep Rosmansyah, and Budiman Dabarsyah. Android anomaly detection system using machine learning classification. In *Int. Conf. on Electrical Engineering and Informatics (ICEEI)*. IEEE, 2015.

[68] Leslie Lamport. *LATEX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[69] M. Langone, R. Setola, and J. Lopez. Cybersecurity of wearable devices: An experimental analysis and a vulnerability assessment method. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 304–309, July 2017.

[70] Marc Lavielle. Using penalized contrasts for the change-point problem. *Signal processing*, 85(8):1501–1510, 2005.

[71] Boohyung Lee and Jong-Hyouk Lee. Blockchain-based secure firmware update for embedded devices in an internet of things environment. *The Journal of Supercomputing*, 73(3):1152–1167, 2017.

[72] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 23–26. IEEE, 2017.

[73] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: A lightweight ui-guided test input generator for android. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, pages 23–26, Piscataway, NJ, USA, 2017. IEEE Press.

[74] Zhen Ling, Junzhou Luo, Yiling Xu, Chao Gao, Kui Wu, and Xinwen Fu. Security vulnerabilities of internet of things: A case study of the smart plug system. *IEEE Internet of Things Journal*, 4(6):1899–1909, 2017.

[75] Song Liu, Makoto Yamada, Nigel Collier, and Masashi Sugiyama. Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, 43:72–83, 2013.

[76] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. On code execution tracking via power side-channel. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1019–1031. ACM, 2016.

[77] George Loukas, Tuan Vuong, Ryan Heartfield, Georgia Sakellari, Yongpil Yoon, and Diane Gan. Cloud-based cyber-physical intrusion detection for vehicles using deep learning. *Ieee Access*, 6:3491–3508, 2017.

[78] S. Lu, R. Lysecky, and J. Rozenblit. Subcomponent timing-based detection of malware in embedded systems. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 17–24, Nov 2017.

[79] Ricardo Manzano, Abdurhman Albasir, Kshirasagar Naik, Nishith Goel, and A Kozlowski. Detection of anomalous behavior in wireless devices using changepoint analysis. In *2019 IEEE International Congress on Internet of Things (ICIOT)*. IEEE, 2019.

[80] Ricardo Manzano, Abdurhman Albasir, Kshirasagar Naik, Jim Kozlowski, and Nishith Goel. Detection of anomalous behavior in wireless devices using changepoint

analysis. In *2019 IEEE International Congress on Internet of Things (ICIOT)*, pages 82–90. IEEE, 2019.

[81] Fabio Martinelli, Francesco Mercaldo, and Andrea Saracino. Bridemaid: An hybrid tool for accurate detection of android malware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 899–901. ACM, 2017.

[82] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic. Eddie: Em-based detection of deviations in program execution. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 333–346, June 2017.

[83] J. Oh, M. Park, and T. Chung. The solution of denial of service attack on ordered broadcast intent. In *16th International Conference on Advanced Communication Technology*, pages 397–400, Feb 2014.

[84] Habeeb Olufowobi, Uchenna Ezeobi, Eric Muhati, Gaylon Robinson, Clinton Young, Joseph Zambreno, and Gedare Bloom. Anomaly detection approach using adaptive cumulative sum algorithm for controller area network. In *Proceedings of the ACM Workshop on Automotive Cybersecurity*, pages 25–30. ACM, 2019.

[85] Jakob Rieck. Attacks on fitness trackers revisited: A case-study of unfit firmware security. *arXiv preprint arXiv:1604.03313*, 2016.

[86] R. A. Riley, J. T. Graham, R. M. Fuller, R. O. Baldwin, and A. Fisher. A new way to detect cyberattacks: Extracting changes in register values from radio-frequency side channels. *IEEE Signal Processing Magazine*, 36(2):49–58, March 2019.

[87] M. Ring, J. Dürrwang, F. Sommer, and R. Kriesten. Survey on vehicular attacks - building a vulnerability database. In *2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 208–212, Nov 2015.

[88] Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[89] Steven Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[90] Robin Joe Prabhahar Soundar Raja James, Abdurhman Ali Albasir, Kshirasagar Naik, Marzia Zaman, and Nishith Goel. A power signal based dynamic approach to

detecting anomalous behavior in wireless devices. In *Proceedings of the 16th ACM International Symposium on Mobility Management and Wireless Access*, pages 9–18. ACM, 2018.

[91] Statcounter. Market share mobile devices worldwide, 2019.

[92] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul V Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in neural information processing systems*, pages 1433–1440, 2008.

[93] L. Vokorokos, P. Drienik, O. Fortotira, and J. Hurtuk. Abusing mobile devices for denial of service attacks. In *2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 21–24, Jan 2015.

[94] T. Wei, C. Mao, A. B. Jeng, H. Lee, H. Wang, and D. Wu. Android malware detection via a latent network behavior analysis. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1251–1258, June 2012.

[95] Tzu-Tsung Wong. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition*, 48(9):2839–2846, 2015.

[96] T. Wüchner, A. Cisłak, M. Ochoa, and A. Pretschner. Leveraging compression-based graph mining for behavior-based malware detection. *IEEE Transactions on Dependable and Secure Computing*, 16(1):99–112, Jan 2019.

[97] L. Yang, V. Ganapathy, and L. Iftode. Enhancing mobile malware detection with social collaboration. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 572–576, Oct 2011.

[98] Ibrar Yaqoob, Ejaz Ahmed, Muhammad Habib ur Rehman, Abdelmuttlib Ibrahim Abdalla Ahmed, Mohammed Ali Al-garadi, Muhammad Imran, and Mohsen Guizani. The rise of ransomware and emerging security challenges in the internet of things. *Computer Networks*, 129:444–458, 2017.

[99] Yao-Saint Yen and Hung-Min Sun. An android mutation malware detection based on deep learning using visualization of importance from codes. *Microelectronics Reliability*, 93:109–114, 2019.

[100] Suleiman Y. Yerima, Mohammed K. Alzaylaee, and Sakir Sezer. Machine learning-based dynamic analysis of android apps with improved code coverage. *EURASIP Journal on Information Security*, 2019(1):4, Apr 2019.

[101] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 387–400, Boston, MA, 2012. USENIX.

[102] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 23–26, May 2017.

[103] M. Zaman, T. Siddiqui, M. R. Amin, and M. S. Hossain. Malware detection in android by network traffic analysis. In *2015 International Conference on Networking Systems and Security (NSysS)*, pages 1–5, Jan 2015.

[104] Thomas Zefferer, Peter Teufl, David Derler, Klaus Potzmader, Alexander Oprisnik, Hubert Gasparitz, and Andrea Höller. Towards secure mobile computing: Employing power-consumption information to detect malware on mobile devices. *International journal on advances in software*, 7(1&2), 2014.

[105] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105–114. ACM, 2010.

[106] Yueyan Zhi, Zhangjie Fu, Xingming Sun, and Jingnan Yu. Security and privacy issues of uav: A survey. *Mobile Networks and Applications*, pages 1–7, 2019.

APPENDICES

Appendix A

Mean and variance obtained with malicious and reference signals

A.1 Results mean and variance for malicious and reference signals

	Reference signal															
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	Total Mean
Mean	1481.8	1477.9	1492.3	1478.1	1464.2	1472.2	1480.9	1489.4	1518.1	1460.6	1466.7	1461.5	1471.9	1475.6	1483.9	1478.4
Variance	82282.61	87903.32	100931.15	89824.87	102802.27	106219.66	93888.23	87199.00	110834.19	83647.71	90008.37	88436.21	97108.54	85733.76	92318.64	93275.90

	Emulated malware 1%															
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	Total Mean
Mean	1471.59	1502.04	1470.58	1475.85	1480.10	1503.13	1467.95	1475.27	1498.50	1469.97	1461.99	1460.91	1479.43	1471.26	1499.93	1479.24
Variance	103113.82	108451.04	110175.98	100398.06	96743.83	115037.17	101587.40	99744.50	107200.17	111318.54	98593.98	101690.86	105012.30	109766.90	106642.57	105031.81

	Emulated malware 2%															
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	Total Mean
Mean	1477.09	1476.12	1487.31	1496.81	1502.01	1511.22	1493.10	1494.75	1474.13	1500.19	1503.09	1503.70	1491.36	1498.58	1498.54	1493.87
Variance	114061.78	107552.06	108195.77	107076.46	113269.54	125558.14	98232.30	105856.44	103222.57	108533.75	112682.90	109987.41	106597.28	108599.28	111752.49	109411.88

	Emulated malware 3%															
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	Total Mean
Mean	1508.84	1500.66	1496.57	1478.08	1496.69	1501.46	1497.84	1511.11	1494.88	1510.54	1505.19	1511.93	1503.14	1505.42	1501.67	1501.60
Variance	116832.19	108218.78	112063.76	114058.61	110909.83	117856.57	108371.80	120427.96	111520.54	124192.43	115749.80	130459.53	112810.24	112460.94	110493.49	115095.10

	Emulated malware 4%															
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	Total Mean
Mean	1501.98	1501.16	1502.51	1495.32	1484.69	1503.79	1496.77	1512.19	1496.88	1500.36	1507.41	1509.91	1517.41	1499.08	1503.90	1502.22
Variance	120173.20	123482.19	114427.33	115082.36	122375.59	119706.59	125887.09	118376.88	123906.61	121670.59	138995.81	128933.70	117363.45	121803.24	123016.69	

	Emulated malware 8%															
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	Total Mean
Mean	1522.07	1528.47	1515.77	1523.99	1496.99	1508.51	1517.19	1487.28	1510.25	1518.89	1516.19	1528.14	1523.49	1527.18	1516.46	1516.06
Variance	122596.99	129192.93	131570.34	130076.41	122444.90	121880.44	118684.91	112589.14	118780.44	126550.71	126972.98	124110.52	120981.32	122836.04	116801.88	123071.33

	Emulated malware 12%															
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	Total Mean
Mean	1518.03	1530.52	1543.73	1513.40	1535.60	1540.79	1538.35	1533.07	1533.93	1548.80	1538.82	1519.69	1526.66	1537.29	1535.23	1532.93
Variance	118700.77	120164.89	119442.58	125237.38	124559.94	131253.01	134293.97	124499.41	125016.98	126312.26	126772.00	118295.35	119669.95	172092.32	121057.37	127157.88