Detection of Anomalous Behavior of IoT/CPS Devices Using Their Power Signals

by

Abdurhman Albasir

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2020

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Srinivas Sampalli Professor, Faculty of Computer Science, Dalhousie University
Supervisor(s):	Kshirsagar Naik Professor, ECE Dept., University of Waterloo
Internal Member:	Fakhri Karray Professor, ECE Dept., University of Waterloo
Internal Member:	Pin-Han Ho Professor, ECE Dept., University of Waterloo
Internal-External Member:	Ali Elkamel Professor, Dept. of Chemistry, University of Waterloo

Author's Declaration

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

Embedded computing devices, in the Internet of Things (IoT) or Cyber-Physical Systems (CPS), are becoming pervasive in many domains around the world. Their wide deployment in simple applications (e.g., smart buildings, fleet management, and smart agriculture) or in more critical operations (e.g., industrial control, smart power grids, and self-driving cars) creates significant market potential ($ 4-11 trillion in annual revenue is expected by 2025). A main requirement for the success of such systems and applications is the capacity to ensure the performance of these devices. This task includes equipping them to be resilient against security threats and failures. Globally, several critical infrastructure applications have been the target of cyber attacks. These recent incidents, as well as the rich applicable literature, confirm that more research is needed to overcome such challenges. Consequently, the need for robust approaches that detect anomalous behaving devices in security and safety-critical applications has become paramount. Solving such a problem minimizes different kinds of losses (e.g., confidential data theft, financial loss, service access restriction, or even casualties).

In light of the aforementioned motivation and discussion, this thesis focuses on the problem of detecting the anomalous behavior of IoT/CPS devices by considering their side-channel information. Solving such a problem is extremely important in maintaining the security and dependability of critical systems and applications. Although several side-channel based approaches are found in the literature, there are still important research gaps that need to be addressed. First, the intrusive nature of the monitoring in some of the proposed techniques results in resources overhead and requires instrumentation of the internal components of a device, which makes them impractical. It also raises a data integrity flag. Second, the lack of realistic experimental power consumption datasets that reflect the normal and anomalous behaviors of IoT and CPS devices has prevented fair and coherent comparisons with the state of the art in this domain. Finally, most of the research to date has concentrated on the accuracy of detection and not the novelty of detecting new anomalies. Such a direction relies on: (i) the availability of labeled datasets; (ii) the complexity of the extracted features; and (iii) the available compute resources. These assumptions and requirements are usually unrealistic and unrepresentative.

This research aims to bridge these gaps as follows. First, this study extends the state of the art that adopts the idea of leveraging the power consumption of devices as a signal and the concept of decoupling the monitoring system and the devices to be monitored to detect and classify the "operational health" of the devices. Second, this thesis provides and builds power consumption-based datasets that can be utilized by AI as well as security research communities to validate newly developed detection techniques. The collected datasets

cover a wide range of anomalous device behavior due to the main aspects of device security (i.e., confidentiality, integrity, and availability) and partial system failures. The extensive experiments include: a wide spectrum of various emulated malware scenarios; five real malware applications taken from the well-known Drebin dataset; distributed denial of service attack (DDOS) where an IoT device is treated as: (1) a victim of a DDOS attack, and (2) the source of a DDOS attack; cryptomining malware where the resources of an IoT device are being hijacked to be used to advantage of the attacker's wish and desire; and faulty CPU cores. This level of extensive validation has not yet been reported in any study in the literature.

Third, this research presents a novel *supervised* technique to detect anomalous device behavior based on transforming the problem into an image classification problem. The main aim of this methodology is to improve the detection performance. In order to achieve the goals of this study, the methodology combines two powerful computer vision tools, namely Histograms of Oriented Gradients (HOG) and a Convolutional Neural Network (CNN). Such a detection technique is not only useful in this present case but can contribute to most time-series classification (TSC) problems. Finally, this thesis proposes a novel *unsupervised* detection technique that requires only the normal behavior of a device in the training phase. Therefore, this methodology aims at detecting new/unseen anomalous behavior. The methodology leverages the power consumption of a device and Restricted Boltzmann Machine (RBM) AutoEncoders (AE) to build a model that makes them more robust to the presence of security threats. The methodology makes use of stacked RBM AE and Principal Component Analysis (PCA) to extract feature vector based on AE's reconstruction errors. A One-Class Support Vector Machine (OC-SVM) classifier is then trained to perform the detection task. Across 18 different datasets, both of our proposed detection techniques demonstrated high detection performance with at least $\sim$ 88% accuracy and 85% F-Score on average. The empirical results indicate the effectiveness of the proposed techniques and demonstrated improved detection performance gain of 9% - 17% over results reported in other methods.

Acknowledgements

All praise is due to Allah, the most merciful, the most compassionate, the creator, sustainer, and nourisher.

The amazing thing about research is that it is collaborative by nature. Completing my PhD was a huge undertaking that I could not have achieved it alone. Therefore, I would like to take this special moment to acknowledge some of the people who were part of my successful PhD journey.

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Kshirasagar Naik, who provided me endless support, encouragement, and friendly discussions. Throughout the five years of my PhD, Prof. Naik offered me several opportunities to collaborate with researchers on different research topics and funded me to attend and present my work in several international conferences. I have learned a great deal from him and from these activities, and as a result, I am more confident and well-rounded researcher.

My appreciation is extended to my committee: Prof. Fakhri Karray, Prof. Pin-Han Ho and Prof. Ali Elkamel for offering many insightful comments, which improved my work; and to the external examiner Prof. Srinivas (Srini) Sampalli who kindly agreed to assess my work.

Many thanks also go to the generous financial support that I received from the Ministry of Higher Education and Scientific Research - Libya and the Natural Sciences and Engineering Research Council (NSERC) - Canada.

I am grateful to my mentor Prof. Otman Basir for the continuous encouragement and wise advices throughout my years of education. Also, my warm thanks go to Prof. Ali Ghodsi for the very beneficial Statistical Learning and Classification course. I am also very thankful to my friends and colleagues Abdulbaset Ali and Tarek Abdunabi for their encouragement and introduction to the field of machine learning in the early days of my PhD. A big thanks to my labmates, Imtiaz Khan, Rubayatur Bhuyian, Alexander Kitaev, and Ricardo Alejandro Manzano, for the coffee breaks and the fruitful conversations!

I would like also to thank my friends in Waterloo for their joyful and supportive company. My sincere thanks go to my soccer friends for all of the fun games that helped me get enough exercises and cope with school stress.

Last and by far not least, I am indebted to my father, *Prof. Ali Elbasir*, and mother, *Mrs. Souad Elbasir* for their continuous and unlimited support, patience, and prayers. My thanks and appreciation go to my brothers and sisters as well. I definitely owe a great deal of gratitude to my wife, Khawla, my daughters, Jori and Hala, and my son Kais for their continuous understanding, unlimited encouragement, and unending love during the years of my study. I could not have done it without all of you being part of my life!!!

Dedication

to my beloved father, *Prof. Ali Elbasir*...

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of Things (IoT) has already begun to have a direct and decisive impact on hundreds of millions of people worldwide (e.g., through the the Google Nest® Thermostat, Amazon Echo®, and iRobot Roomba®). People are witnessing the abrupt adoption of IoT services and applications in different domains (as shown in Fig. 1.1). While it is exciting for both industry and academia, it is expected that this development is just the beginning of a transformational journey. In the near future, every device a person owns, together with any object with which he/she interacts, will be connected to the Internet. Whether it is through a smartphone, wearable tech, or everyday household objects, the IoT will connect people in ways they cannot yet even imagine. In homes, smart thermostats, alarm systems, smoke detectors, and intelligent refrigerators will help reduce energy consumption, secure and protect assets, and increase user comfort. In industry, autonomous controllers operate and monitor complex critical production pipelines to improve productivity and reduce service operation costs. In agriculture, large-scale networks of small sensors collect information about the environment to help increase yields and better allocate resources. In smart health care, remote health monitoring makes a significant difference in people's lives (for example, those with chronic illnesses); while decreasing the cost of health care for these patients, it improves their quality of life.

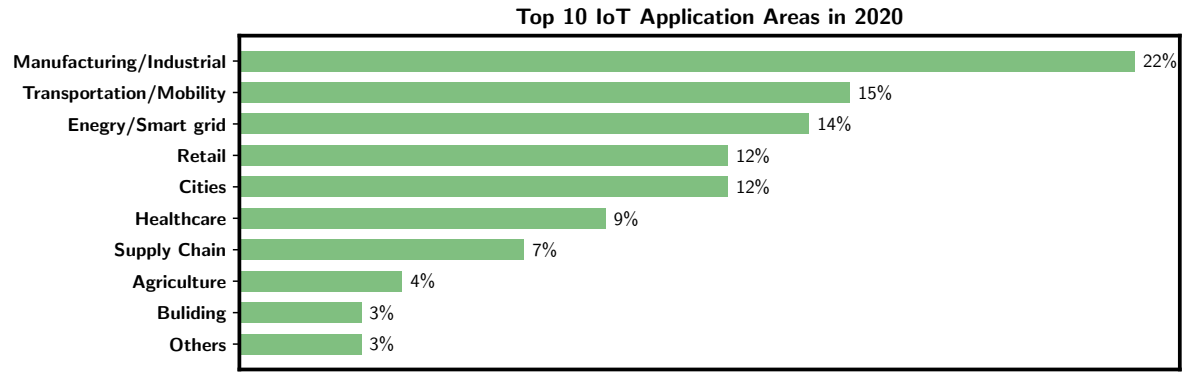


Figure 1.1: Adoption percentages of IoT technology per domain - Source: adopted from IoT Analytics [194]

According to Heer et. al. [104], the generic definition of the IoT is: "*The Internet of Things denotes the interconnection of highly heterogeneous networked entities and networks, following a number of communication patterns, including human-to-thing, thing-to-thing, and thing-to-things*". This definition includes any system that is based on computing devices that are uniquely identified and have the ability to autonomously process and transfer data over a network. By definition, IoT technology covers a wide range of services and applications (e.g., self-driving cars, smart health care, Industrial IoT, or any cyber-physical system (CPS)). In this context, regardless how critical the offered IoT applications are, the core of these systems is embedded devices -"things" - that are connected to the Internet and are designed to monitor and control certain tasks about the surrounding environment/system. In other words, these devices may be working within a complex system or as stand alone applications [203].

Despite the overall positive feeling about IoT and CPS development, as well as the considerable benefits for the end user, there are two perspectives to be considered. From one perspective, IoT/CPS applications and devices create extensive market potential. Some sources report that $ 4-11 trillion in annual revenues is expected in 2025 [154]. From the other perspective, unfortunately, IoT/CPS applications and devices create the next target for security attacks. In fact, IoT devices are forecast to be involved in more than 25% of the identified attacks in enterprises in 2020 [116]. Due to the fact that device manufacturers are

racing to be the first in the market, little time and effort is usually assigned for testing and securing the developed devices and solutions. Also, given the fact that the functionality of these devices depends on a direct data exchange (via the Internet or otherwise), these devices have become inherently not secure. This also arises from the fact that these devices are constrained in memory and processing (a typical IoT device has a CPU of $\sim$ 64 MHz to 1 GHz and a RAM of $\sim$ 4 MB to 512 MB [179]). Therefore, having complex security algorithms (e.g. anti-virus) is computationally expensive, hence is not viable.

The success of IoT and CPS based systems and applications is faced with the challenge of properly and efficiently managing the devices [56, 129, 41, 218]. This is generally a multidisciplinary problem. Management of the device challenge includes ensuring device connectivity, fault identification, performance, reliability, quality of service, and security [80, 95, 190]. Under the umbrella of device management, many important research problems need to be tackled. However, the challenge to be addressed in this thesis concerns the accurate and efficient non-intrusive detection of a device's anomalous behavior. In this context, *accurate and efficient* detection means that the methodology produces low false alarms and causes no overhead to the operation of the devices.

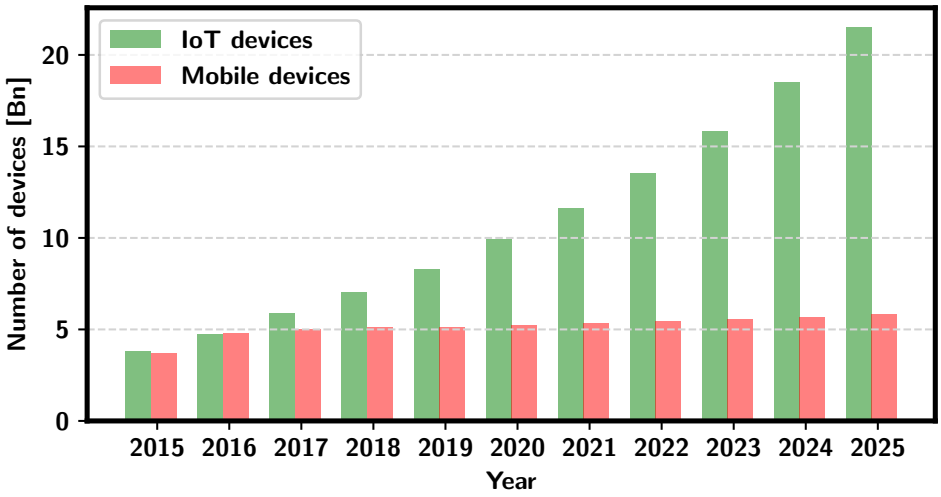


Figure 1.2: Evolution of IoT and non-IoT devices from 2015 to 2025: Source: adopted from GSMA [28]

According to a Gartner report, 5.5 million new devices have been added every day since 2016. The total number of devices is forecast to surpass 20 billion by 2025 [82], as the trend in Fig. 1.2 illustrates. The scale at which these devices are deployed, and the level of criticality of certain applications, shows that overcoming the aforementioned challenges has become imperative.

Several cyber-security incidents that involve IoT/CPS devices (as an attacker or a victim) have been reported in the last few years. In the industry sector, 66% of manufacturers have experienced a security incident related to IoT devices in the span of the last two years [2]. The infamous Stuxnet worm attack against Iranian nuclear uranium enrichment facilities is one example [88]. The Ukraine smart power grid, which was the target of a cyber-attack that resulted in a three hour power outage nationwide, is a second example [132]. More recently, the infamous DDOS attack on the Dyn DNS (domain name system), which involved $\sim$ 100,000 malicious IoT devices [105] is a third example. Other sectors have suffered from several ransomware attacks [74]. Self-driving vehicles [16], smart home devices [21], and medical devices [137], to name a few, have all been the subject of different kinds of attacks and vulnerabilities. These are just some examples of the type of critical infrastructure that can be targeted globally. These recent incidents and studies confirm that more research needs to be conducted and also show that existing solutions are far from satisfactory in the attempt to stop the exponential growth in the number and complexity of attacks [162]. Thus, there is paramount need for more lines of defense.

1.1 Problem Overview

Motivated by the aforementioned arguments, this thesis focuses on the problem of detecting the anomalous behavior of IoT/CPS devices by considering their side channel information. Solving such a problem is extremely important in maintaining the security and dependability of critical systems and applications.

Statement:

This research has been inspired by the fact that cyber-security attacks are increasingly occurring worldwide, targeting the IoT/CPS devices used in critical systems. Effective and robust approaches for detecting anomalous behavior of devices can minimize a significant amount of financial and life losses. Therefore, the objective of this present work is to propose new approaches that leverage the power consumption of devices and deep learning techniques to make these devices more resilient against different kinds of attacks and failures.

A typical approach to detecting a device's anomalous behavior is through formulating the problem as an anomaly detection problem. An anomaly, as per the IEEE standard, is defined as any condition that deviates from *expectations* [246]. Anomaly detection refers to the process of using models to identify behavior that is different from the normal behavior of a system [55].

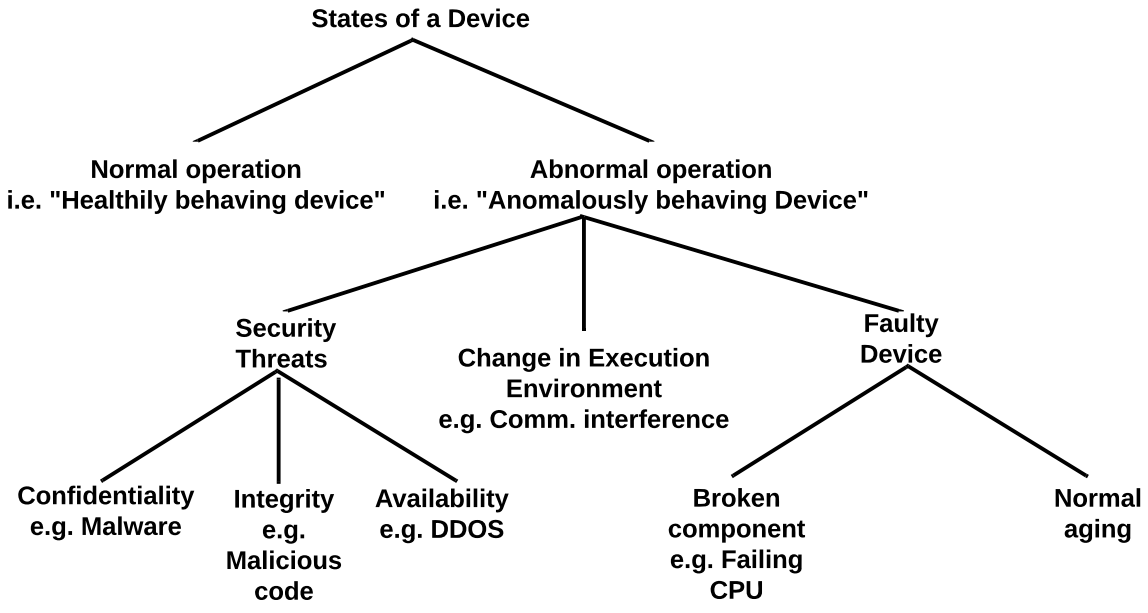


Figure 1.3: Breakdown of an IoT device's possible status

In the context of this thesis, under normal operational conditions, a device is *expected* to perform repetitive, periodic tasks (e.g., sensing the surrounding environment, actuating physical actions, performing local computations, and offloading data to the cloud). Consequently, its normal - *expected* - behavior will have a pattern that reflects those tasks. Therefore, when something unexpected goes off on board the device, its behavior will accordingly deviate from the *expectation*. Based on this concept, and as shown in Fig. 1.3, the cause of a device's anomalous behavior can be due to an anomaly of one of the following forms: (i) security threats (e.g., malware, DDOS, or ransomware); (ii) a change in the execution environment (e.g., interference in the communication channel or firmware updates); or (iii) a faulty device (e.g., hardware/software aging or failing components) [56, 115].

Given the complexity and heterogeneity of the problem, many research efforts have been made to address the problem of detecting the anomalous behavior of devices at different levels: device level [29, 241, 51]; network level [223, 183]; and system level [224]. Previous studies and results have demonstrated some success [131, 226, 241, 123, 156, 134, 98, 190]; however, many significant research gaps are still there and has to be addressed. These gaps are discussed in a broad sense in the following section. It is also worthy to note that several recent incidents confirm that more has to be achieved [105, 74, 21].

1.1.1 Challenges

To achieve the goals of this thesis, an anomaly detection-based solution is developed. As depicted in Fig. 1.4, the general process of detecting anomalous behavior in devices starts with the data acquisition stage (also called *monitoring*), followed by the data *analysis* stage, and the final decision-making (also called *detection*) stage. The key challenges faced in designing this detection framework are discussed in the following subsections.

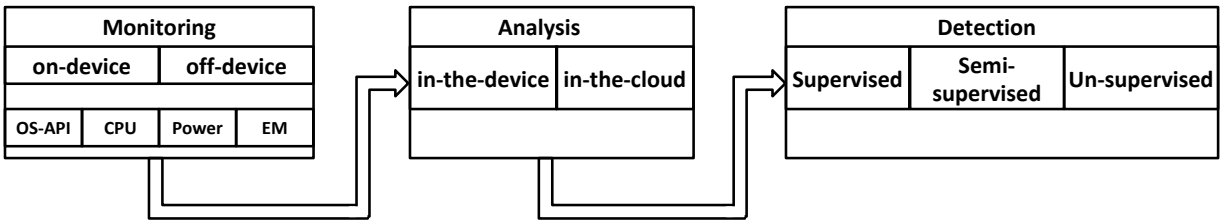


Figure 1.4: Steps for anomaly detection techniques

Monitoring

Monitoring is the process of collecting data - observations - from a system, an IoT or a CPS device. The collected data points (signals) describe certain measures or phenomena about the monitored system (e.g., API calls, CPU utilization, power consumption, or electromagnetic emissions). Monitoring can be generally classified into two categories based on the invasive nature of the process, namely intrusive or non-intrusive. Intrusive methods refer to methods that run on-device using applications/software that are installed in the device to monitor its behavior for the purpose of detecting any abnormality. The main problem with intrusive methods is the *resources overhead*. The operation of the intrusive monitoring tool interferes with the normal operation of the device, causing unaffordable computational cost (e.g., computing power, memory, storage space, and bandwidth). In the reviewed literature, most of the proposed approaches are based on coupling the monitoring tool with the device being monitored (on-device monitoring) [241, 231, 151, 54, 197, 50, 84, 22]. Such invasive approaches become questionable when it comes to usability and practicality.

Since in practice, *monitoring comes at a cost*, in the present case, the side channel information of a device, represented by its power consumption signals, is monitored. Such a non-intrusive (off-device) monitoring is lightweight, thus more suitable for resource constraint devices. The necessity for exploiting the side channel information is rooted also to the idea of preserving *data integrity*. In the case of security threats, namely malware invasion, if the malware is sophisticated, it can detect the existence of intrusive monitor-

ing methods [124, 98]. Consequently, the malware can block the attacker or manipulate the collected data to avoid being detected. Therefore, intrusive monitoring raises a data integrity flag. In other words, if a device is compromised, then so could the monitoring means as well as the collected information [111, 172].

Although some studies found in the literature are based on off-device monitoring, it may be noted that those require a degree of invasiveness. To elaborate, a physical access to the internal components of the device (e.g., CPU input power) is required in some of the power based anomalous behaviour detection techniques [151, 48]. In another example, electromagnetic emissions (EM) based approaches require physical proximity to the device's CPU, along with proper alignment and setup [131, 130, 196]. The reported experimental setup in these studies shows that EM receiver has to be placed close to the CPU in order to collect the EM signals. Now, if the targeted device is packaged in a package that suppresses the EM radiation or even weakens the radiated signal, then the applicability of these approaches becomes problematic. Moreover, the EM receiver itself might be the subject of an external attack (e.g., using signal jammers), which could jeopardize the whole idea of EM based detection.

Analysis and Detection

Thus far, non-intrusively collecting the power consumption of devices seems to satisfy the requirements for a suitable and practical detection approach. It is important to mention that the collected signals are the overall power consumption of the device. Such a signal combines the instantaneous power consumed by all hardware components of the device and the software that runs it. Moreover, the power consumed by devices is a 1-D stochastic time-series signal, which is generally complex to be described with analytical equations with parameters to solve. Consequently, the nature of the collected data poses other kinds of challenges. These challenges arise when deciding on how and where to analyze the collected power signals. "How" refers to the analysis type (e.g., static, dynamic, or hybrid) and technique (e.g., supervised, unsupervised, or semi-supervised), and "where" refers to

the location of analysis (i.e., on-device or off-device analysis).

The first challenge in the analysis phase is that, in the field of this thesis's topic, there is a lack of realistic experimental power consumption datasets that reflect the normal and anomalous behaviors of IoT and CPS devices. Such power consumption-based datasets are vital to reproduce, evaluate, and compare scientific results. These datasets can be utilized by AI and security research communities to validate and compare newly developed detection techniques and build effective and robust models.

The second challenge is *analysis type*. When the anomalous behavior of a device is the result of malicious actions, malicious adversaries often adapt or mutate themselves to make their anomalous trace in the collected data appear normal, thereby making the task of defining normal behavior more difficult. A more challenging case is when the malicious code is an encrypted malware. For instance, when static analysis is adopted in the detection approach, it has been found that hackers/malware can easily modify individual features to appear as a benign feature, hence avoid detection [18]. Therefore, addressing the choice of analysis type in the detection approach is a critical task.

The third challenge in the analysis phase is that *the hand-crafted features are exorbitantly expensive*. Analysis and detection involve the process of digesting the collected data to identify anomalously behaving devices from normally operating ones. In the present case, the challenge is how to extract knowledge and informative insights out of the raw power consumption signals. This is a challenging and yet interesting task. As mentioned above, the power consumed by devices is a stochastic time-series signal. In other words, these signals are highly dynamic and exhibit nonlinear behavior [217]. They have a 1-D structure and are non-stationary, which means that signals' characteristics (namely mean, variance, and frequency) change over time. Moreover, these signals contain much noise and do not clearly show the information (i.e., events, their frequencies, and duration) that can be used as discriminative features. Therefore, dealing with them in their original structure is problematic and it leads to the fact that achieving reasonably accurate detection and classification performance can not be easily achieved [32] when employing traditional detection methods. The general trend is to develop domain-specific features for each task,

which is expensive, time-consuming, and requires expertise in the signals and applications. Therefore, in this thesis, we explore other spaces to come up with alternatives that overcome the explained challenges.

Finally, the detection's aim challenge: *accuracy of detection vs novelty of detecting new anomalies*. In the detection process, the main issue to consider is the type of detection approach that is employed. The answer to such a question depends on the availability of labeled datasets. This also impacts the aim to be achieved, whether it is a high detection performance (accuracy, precision, recall, and F-score) or the novelty of detecting anomalies that cause a device to misbehave. Consequently, if a labeled dataset is obtained, supervised detection approaches can be implemented. Such approaches aim at improving the detection performance. However, if an unlabeled dataset (or at least one label is known, i.e. a normal behavior label) is available, then unsupervised or sim-supervised detection approaches can be used. These approaches aim at detecting new anomalous behaviors (e.g., zero-day attack or misuse detection).

1.1.2 Objectives

Power consumption of devices, as a side-channel information source, has been exploited by attackers to reveal secret keys [155]. That is due to the strong and direct correlation between the tasks a computer/device performs and its power consumption behavior. For the same reason, we have chosen to exploit the power consumed by IoT/CPS devices to make them better protected and secured. Therefore, the primary objective of this thesis is to propose a detection framework. The proposed detection framework can be used to enhance the security and dependability of computing devices used in building IoT and CPS applications. The aforementioned challenges are addressed through examining ways to detect interesting insights about the operational health of a device by seamlessly monitoring and analyzing its *power consumption signals*. The proposed detection approaches are to adopt the idea of non-intrusive monitoring of a device's power consumption signals and leveraging supervised and semi-supervised techniques to perform the detection task.

The secondary objective is to build several datasets that can be used to validate the proposed detection techniques. The aim is to cover a wide range of anomalous device behaviors obtained from smartphones and generic embedded devices (IoT). The anomalous device behavior is expected to include a large class security threats, which covers the three main aspects of device security (i.e., confidentiality, integrity, and availability), as well as partial system failures.

Since the nature of the collected data (i.e., the power consumption od devices) is discrete and sequential (i.e., time-series signals), such data can be found across different applications and domains. In fact, time-series signals can be captured by monitoring physical systems (e.g., earth seismic dynamics [161], vibration of a mechanical system [35], or a robot arm in an automobile plant [13]), human bodies (e.g., motion, electrocardiogram (ECG), or electroencephalographic(EEG)), and acoustic sources (e.g., sound and audio). This argument gives the proposed detection techniques a broader scope of applicability that is beyond the main objective of this thesis.

1.1.3 Contributions

- C1: This thesis extensively reviews the associated literature and identifies the tools as well as the proposed techniques that mainly consider side-channel information and also conduct dynamic analysis in order to detect the anomalous behavior of IoT/CPS devices. Such a review has never before been conducted, particularly in such way. This survey provides information such as the nature of information used for analysis, the type of anomalies causing such behavior, the tools used for analysis, and the reported detection performance.

- C2: This research models the causes of a device's anomalous behavior as an anomaly of one of the following forms: (i) security threats (e.g., DDOS attack, malware, ransomware, or resources hijacking); (ii) change in the execution environment (e.g., communication interference); or (iii) a faulty device (e.g., faulty component or hardware aging). It also investigates the effectiveness and usefulness of using the power

consumption signals of devices to infer some effective insights into the "operational health" of these devices, specifically, detecting security threats and faulty devices.

- C3: This research provides and builds power consumption-based datasets that can be utilized by AI and security research communities to validate newly developed detection techniques. The collected datasets cover a wide range of anomalous device behavior, namely three main aspects of device security (i.e., confidentiality, integrity, and availability) and partial system failures. The extensive experiments include: a wide spectrum of various emulated malware scenarios; five real malware applications taken from the well-known Drebin dataset; distributed denial of service attack (DDOS) where an IoT device is treated as: (1) a victim of DDOS attack, and (2) a source of DDOS attack; cryptomining malware where the resources of an IoT device are hijacked to be used to the advantage of the attack; and faulty CPU cores. The wide range of the collected datasets allows extensive validation to the proposed detection approaches to be performed. These datasets, together with the details of the experiment, will be made publicly available in the future. This level of extensive validation has not been previously reported in any study in the literature.

- C4: This research presents a novel *supervised technique* to classify and detect anomalous device behavior based on transforming the problem into an image classification problem. The main aim in this methodology is to improve the detection performance. For this reason, the employed technique is a supervised model (i.e., requires labeled data). The methodology combines two powerful computer vision tools, namely Histograms of Oriented Gradients (HOG) and a Convolutional Neural Network (CNN) to achieve the goals of this study. In this methodology, the 1-D instantaneous power consumption signals of devices are transformed to time-frequency representation (TFR), using Constant Q Transformation (CQT), to form 2-D images. The CQT images capture valuable information about the tasks performed on board a device, e.g., events' location in time and frequency domains, frequency of events, and the shape and duration of events. Applying HOG to the CQT images allows the extraction of more robust features that preserve the edges of time-frequency structures (i.e.,

description of local information) and also the directionality of the edge information (i.e., how these structures/events evolve with time). Finally, a CNN model is trained on images containing HOG features to classify these signals and detect anomalously behaving devices. Such a detection technique is not only useful for our particular case but can contribute to most time-series classification (TSC) problems.

- C5: This research proposes a novel *unsupervised detection technique* that requires only the normal behavior of a device in the training phase. Therefore, this methodology aims at detecting new/unseen anomalous behaviors. The methodology leverages the power consumption of a device and Restricted Boltzmann Machine (RBM) AutoEncoders (AE) to build a model that makes them more robust to the presence of security threats. The methodology makes use of stacked RBM AE and Principal Component Analysis (PCA) to extract feature vector based on AE's reconstruction errors. A One-Class Support Vector Machine (OC-SVM) classifier is then trained to perform the detection task. Such a technique is important since obtaining sufficient labeled data for training/validation of models used by anomaly detection techniques is acknowledged as a major issue in this field of study.

1.2 Thesis Structure

This thesis is structured as follows: **Chapter 2** provides the necessary background information to understand the context and findings of this work. It also provides a discussion of the state of the art in the field of this study. **Chapter 3** discusses a high-level overview of the system model used throughout the thesis. The discussion covers the architecture model and the anomaly model that covers the causes that could lead a device to behave anomalously, i.e., security threats and faults. **Chapter 4** presents the experimental setup and the experiments that were conducted to collect the datasets used to validate the proposed detection techniques. This chapter provides a summary of the collected datasets that covers the three main aspects of a device's security (namely, confidentiality, integrity,

and availability), and partial system failures. It also presents an in-detail technique for generating a wide spectrum of different families of emulated malware. **Chapter 5** presents the supervised detection technique, where the methodology details and obtained results are discussed. **Chapter 6** presents the unsupervised detection technique along with the validation results. **Chapter 7** outlines the lessons learnt in this work in the conclusion section while the future work section presents the wider applicability of the proposed approaches presented here, and identifies some limitations. A summary of potential ways to extend the work are also included.

Chapter 2

Background and Literature Review

Before we begin formulating the problem and discussing the proposed solutions, it is important to give a broad overview of the topics that are necessary to understand the proposed framework and justify the need for such a solution. Therefore, the goal of this chapter is two-fold: first, we provide an overview of the Internet of Things and introduce the concept of anomalous behavior detection; second, we discuss the research related to the thesis's main objective. The literature review includes the topic specifically related to the sources of anomalous behavior we consider in this thesis; namely detecting security threats and failing components in IoT/CPS devices.

2.1 Internet of Things

Internet of Things (IoT) has gained much of attention in academia and industry in the last decade. IoT in a nutshell refers to almost any device or "thing" that may exchange data (either by transmitting information collected by its sensors or/and receiving information by its actuators) to enable centralized analysis and derivation of insights in a designated cloud infrastructure. Depending on the application, "things" in this scope refers to any computing device that is connected to the Internet; e.g., appliances, infrastructures, build-

ings, cars, robots, animals, and even people. To elaborate, in smart health care domain, medical sensors are attached to patients to measure and monitor various medical parameters to determine the patients' health condition. In such a case, people are transformed, in some sense, to "things" where they start transmitting information about their health to a central processing entity (hospital). Another example is the transformation of traditional power grids into smart grids by incorporating computing devices designed specifically to ensure that the power is delivered efficiently, reliably, and affordably. The common objective across all of the offered IoT services and applications is to improve the sustainability and safety of industries and societies. Moreover, the IoT is meant to enable efficient interaction between the physical world and its cyber counterpart (also known as the digital transformation or cyber-physical systems (CPS)).

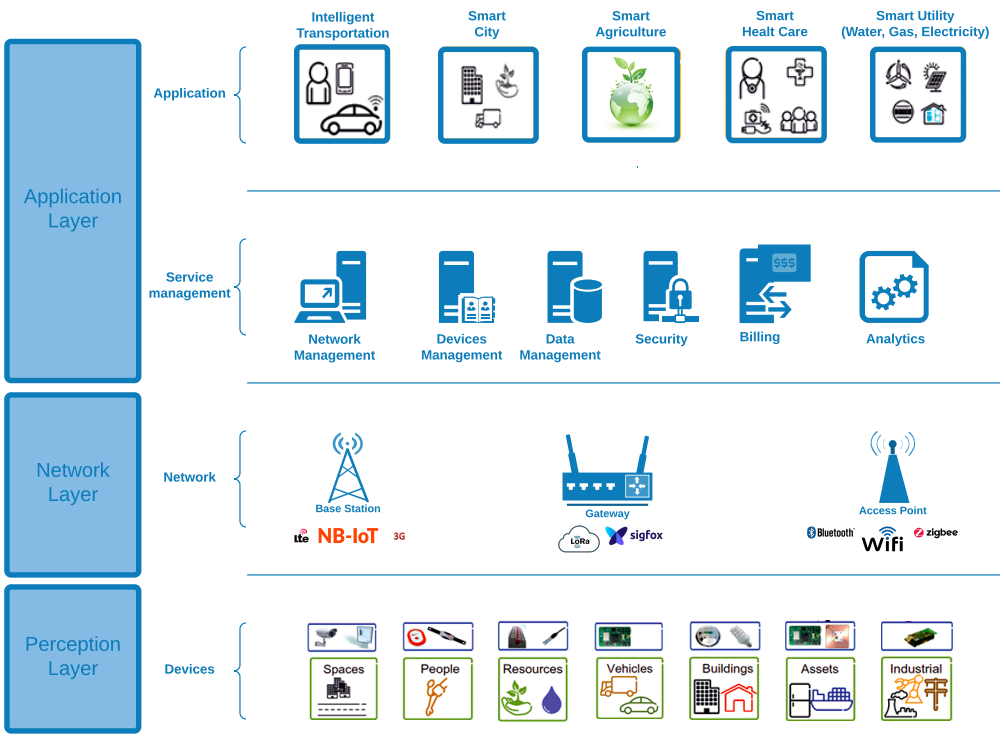


Figure 2.1: IoT Reference Architecture

2.1.1 Reference Architecture

The IoT is envisioned to be capable of interconnecting huge number of heterogeneous devices via the Internet; therefore, it is very important to have a flexible layered architecture. In literature, we can find that there are several architectures, each is described slightly differently depending on the ecosystem and domain of study. We have chosen the most commonly reported models that describe the IoT architectures, as shown in Fig. 2.1. The model on the left hand side shows a 3-layer abstract architecture. This model consists of application, network and perception layers. However, the model in the right hand side of Fig. 2.1 depicts more details and breaks down the IoT reference model into four main layers/levels: (i) IoT devices (things), (ii) IoT network (infrastructure transporting the data), (iii) IoT services platform (software connecting the things with applications and providing overall management), and (v) IoT applications (specialized business-based applications such as customer relation management (CRM), accounting and billing, and business intelligence (BI) applications).

IoT Device layer

The first level represents the physical sensors or actuators used to build IoT applications. In general, these devices have three main requirements in IoT; sensing, actuating, and addressing. Sensing is essential to collect key information to monitor and diagnose the things; addressing is necessary to uniquely identify things over the Internet; and actuating is important to control things and take actions when needed.

IoT devices may be very simple with a core function to collect and transmit data; however, they can also be complex - smart - by providing additional functionality. This requires some sort of computing capabilities to be present on the IoT device itself. In this case, an IoT device requires at least four elements: sensor(s)/actuator(s), microcontroller(s), power source, and connectivity to send data to IoT gateway or other systems. Figure 2.2 shows the components of a typical IoT device.

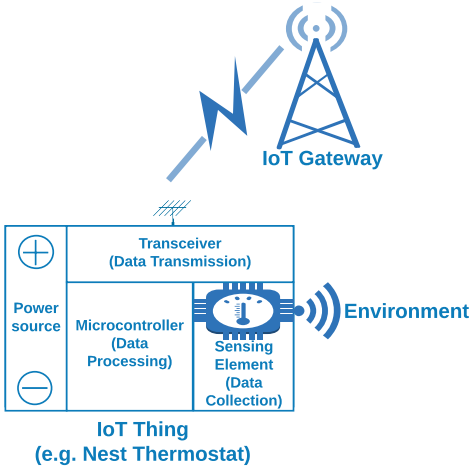


Figure 2.2: Main Components of an IoT Thing

We highlighted the main components of an IoT device since it serves the motivation and the solution strategy of our study. The operation of IoT devices is driven by the component (hardware) shown in Fig. 2.2 and/or software. These hardware components can be leveraged as sources or metrics to detect anomalous behavior. Most of the parameters described (power, cpu, network, or sensor data) can be measured using an external device (off-device) or internal logging (on-device).

IoT Network Layer

The second level represents the interconnection between gateways and the IoT devices, as shown in Figure. 2.1. There is no specific preferred networking protocol for IoT so far, as a result, IoT is envisioned to support hybrid network architecture. Most of the information reported in this section is extracted from [191] [184].

The main functionality of this layer is to transfer the collected data from the device's layer (i.e., monitoring task) to the application layer. In case of a controlling task, the network layer delivers the data to actuators (i.e., telemetry commands) in the perception layer.

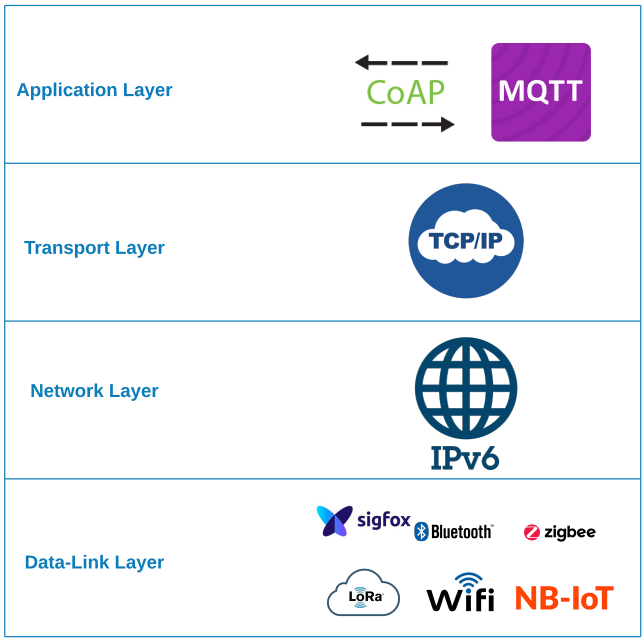


Figure 2.3: Main Components of the IoT Network Stack

In practice, the gateways are envisioned to support more than one wireless technology. Figure. 2.3 describes a typical communications network protocol stack. Common features of such a stack are the idea of encapsulation and modularity. These two features allows different layers to work independently, and as a result, they reduce the system's complexity and allow for scalability.

IoT Service Platform Layer

The players in this level are responsible for many of the IoT management tasks. Services Platform are there to automate the ability to deploy, configure, troubleshoot, secure, manage, and monitor IoT solution entities. These entities range from sensors to applications in terms of firmware installation, patching, debugging, and monitoring, to name a few. In its simplest form, the layer ensures/controls that a specific services is delivered to the entity that requested it based on addresses and names.

IoT Application Layer

This layer provides the diverse kinds of services requested by the customer. The type of service requested by the customer depends on the specific use case that is adopted by the customer. For example, if smart home is the use case under consideration, then the customer may request for specific parameters such as heating, ventilation, and air conditioning (HVAC) measurements or temperature and humidity values. This layer provides the various types of smart services, which are offered by various IoT verticals. Some of the prominent IoT verticals are as follows:

- Intelligent Transportation
- Smart City, Industry, Agriculture, and Health Care
- Smart Utilities (Water, Gas, Electricity)
- Smart Buildings or Homes

2.1.2 Opportunity and Challenges

The realization of the IoT applications requires tackling several problem, in all of the layers we described above, by different research fields. However, in this section we focus mainly on the challenges that overlap with the objectives of this thesis.

Looking at the scale that these devices are deployed at and the level of criticality of certain IoT and CPS applications, these devices and applications create huge market potential. Some sources report that $ 4-11 trillion in annual revenues is expected in 2025 [154]. The astronomical figures are creating a competition between the manufacturers, where each of them wants to be the first in the market. This race is leaving a little time and efforts for testing and securing the developed devices and solutions; consequently, such devices and applications are highly susceptible to different security threats and performance failures. Based on this argument, IoT/CPS applications are expected to create the next

target for security attacks. According to a recent report by Gartner, it is expected that IoT devices will be involved in more than 25% of the identified attacks in enterprises in 2020 [116].

Building upon these challenges and statistics, we found, in almost every study that aims at discussing the challenges within the IoT domain, a section that is devoted to the security and privacy matters [244, 129, 16, 23, 104, 127]. As of now, several cyber-security incidents that involve IoT/CPS devices (as an attacker or a victim) have been reported in the last couple of years. These security breaches range from Distributed Denial of Service (DDoS) attacks - e.g., the infamous DDOS attack on Dyn's DNS (domain name system) which involved $\sim$ 100K malicious IoT devices, according to reference [105]; several ransomware attacks [74]; attacks/vulnerabilities on/of self-driving vehicles [16]; attacks on smart home devices [21] and medical devices [137].

The takeaway is that IoT and CPS have introduced new security risks for both consumers and businesses. Mitigation of these risks is challenging due to limited resources on-board the used devices to build IoT and CPS applications. Moreover, The need for continuous Internet connection for the devices makes them become inherently not secure. These recent incidents and studies confirm that more research needs to be done. They also show, unfortunately, that existing solutions are far from being satisfactory to stop the exponential growth in number and complexity of attacks [162]. Thus, the need for more lines-of-defense to make IoT and CPS systems more resilient and proof to such attacks is paramount. The next section introduces the concepts related to anomalous behavior detection in IoT devices, followed by previously reported research.

2.2 Anomalous Behavior Detection

The detection of anomalous behavior of a device involves three main phases, as shown in Fig. 2.4, the wide vertical arrow to the left. It starts with a monitoring phase, followed by an analysis phase and a decision phase.

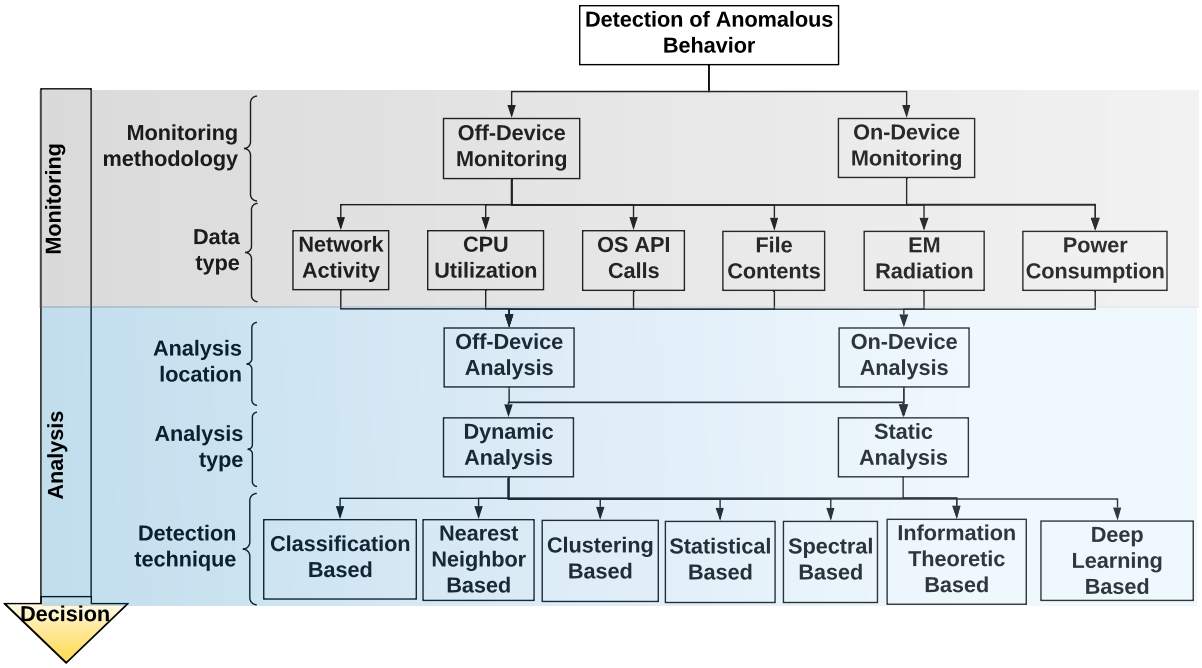


Figure 2.4: Schematic classification of malware detection techniques in wireless devices

2.2.1 Phase 1: Monitoring

Monitoring is the process of collecting the data points (also called observations) that describe certain measures or phenomenons about the monitored system. In the domain of this thesis, there are two main approaches to collecting the raw data: (i) *Off-Device* monitoring [130][123], which refers to the process of collecting the data using **external tools** (software/hardware), and (ii) *On-Device* monitoring, which refers to using applications/-software that are installed in the device to collect the data [89] [29] [231][241]. The type of the collected data for the analysis can be: time series signals (e.g., power consumption or electromagnetic radiation of a device as in [110][131]), events (e.g., CPU, memory utilization, and network traffic as in [164][212]), system calls, and files/entities [26], as classified in Figure 2.4. Each of these data points can be used as a *raw data* or *processed data*. *Raw Data* means that the collected data points are used as is for analysis. On the other hand, *processed data* means some features are extracted from the collected data points for anal-

ysis. This process is very crucial for detecting anomalies, where bad/noisy observations could jeopardize the overall detection performance. The type of data can also play a role in determine the detection technique to be used for analysis in the next step.

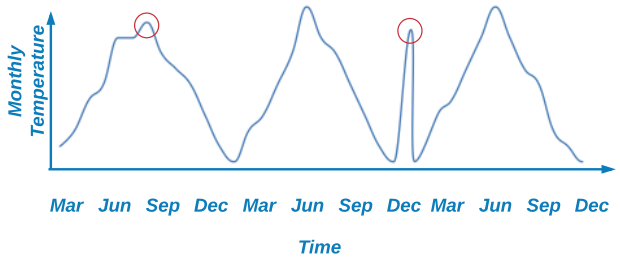
2.2.2 Phase 2: Analysis

Analysis is the step where the detection technique/algorithm is applied. In general, the first thing to determine in this step is the type of problem (i.e., *anomaly*). An anomaly can take the form of: (i) contextual anomaly, (ii) collective anomaly, and (iii) point anomaly, according to [55].

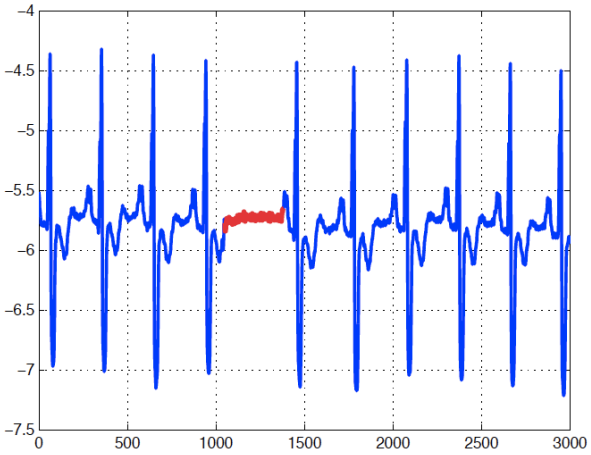
Contextual Anomaly, as the name suggests, depends mainly on the context understudy. An example might be: detecting irregular heart rhythms from ECG record [181]. In ECG record, the magnitude variation by itself does not tell exactly if there is anomaly or not, while the irregularity in the cardiac cycle gives the clue to the detection of anomalies. A simpler example is: having a high temperature is normal if it is observed in the Summer, however, it is considered to be up normal (anomaly) it it was observed in Winter, as shown in Figure. 2.5a.

Collective Anomaly, refers to the case where having multiple related (relationship can be in the form of sequential) anomalous data points with respect to the entire data set. In this anomalous category, having just one anomalous data point may not be considered as anomaly by itself, rather multiple occurrence of anomalous data points together is considered as a collective anomaly, as depicted in Figure. 2.5b. While point anomalies can occur in any data set, collective anomalies can occur only in data sets in which data instances are related [55].

Point Anomaly, also called out-lier, refers to an individual reading or observation that does not fall within the expected range. For example, in Figure. 2.6a, points marked in red lie outside of the boundary of the normal regions, and hence are point anomalies since they are different from normal data points (marked in blue).

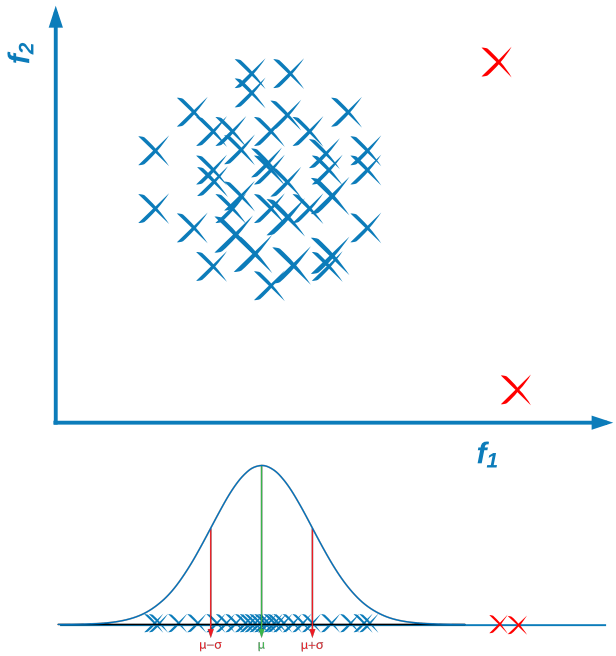


(a) Contextual anomaly



(b) Collective anomaly, taken from [55]

Figure 2.5: Types of Anomalies: 1/2



(a) Visualization of the collected data in 2D and 1D along with the fitted normal distribution (blue cross represents normal behavior and red represents anomalous behavior

Figure 2.6: Types of Anomalies: 2/2

As we depict in Fig. 2.4, in the domain of this study the analysis phase can be broken down into three sub-steps:

(i) *Analysis location*: In the analysis phase, the first thing to decide on, based on certain requirements (e.g., resources, privacy, etc.), is where the analysis is performed: *On-Device* or *Off-Device*. In *Off-Device* analysis, also called cloud based analysis, the collected data is processed and analyzed in the cloud and only the decision is sent back to the device or to the device operator as a notification [130][123]. Given the fact that some devices are resource-constrained, this option is preferred [55]. On the other hand, *On-Device* analysis refers to analyzing the collected information on the device itself. Although this option requires more computational resources, it is considered to be the desired option when the data privacy is required [201].

(ii) *Analysis type*: When the anomalous behavior is due to malware, the literature can be further classified based on the analysis type: *static analysis* [26], *dynamic analysis*, or hybrid [72]. *Static analysis* refers to analyzing a software's source code and manifest file without the need to execute it to check if it is malicious or not. Such an approach has the ability to detect zero-day (unknown) malwares; however, it has the limitation of the need to access the source code. Another drawback is, if malware developers learn about the hypotheses in static analysis, they can adapt their code to look benign. Since our proposed approach falls under the dynamic analysis category, the static based methods are out of the scope of this thesis. On the other hand, in *dynamic analysis*, a device is examined during execution time of certain tasks. The idea is to profile a device by observing its behavior (e.g., generated traffic, CPU utilization, or emitted electromagnetic radiation) while it is in operation; consequently, you learn some insights about it. This kind of analysis depends mostly on the data collected through side channels. Moreover, approaches based on *dynamic analysis* aim to detect and identify infected devices to minimize further damage.

(iii) *Detection technique*: In the final step of the analysis phase, the type of the employed detection technique is chosen, i.e., supervised, semi-supervised, or unsupervised anomaly detection [55]. The decision depends on the collected data. Supervised anomaly detection requires labeled data. Therefore, if the collected data is not labeled, then the following step is to label the collected data. This step falls under the data pre-processing phase in the traditional machine learning pipeline model.

If the technique is semi-supervised anomaly detection, a training set with observations sampled from only one of the two classes is used. For example, in the case of anomalous behavior detection problem, this kind of techniques is applied typically when there are no observations from the anomalous class. Based on the definition of anomaly, these techniques usually train a model to recognize normal behavior. A new observation will be classified as an anomaly if it falls in a low-probability region [55].

Finally, if the technique is unsupervised anomaly detection, then the used train-

ing data is unlabeled. This kind of anomaly detection techniques work well when the normal observations are noticed with very high frequency/density in contrast to anomalies which are rare to happen/notice [55].

2.2.3 Phase 3: Decision

After the analysis is performed and a model is trained, a decision about the model output type is needed to be taken. That is, when a test observation is fed to the model, what is the proper output that determines whether that observation is an anomaly or not. Typically there are two types of outputs: *Score* and *Label*. *Score* refers to the case where the model assigns an anomaly score to each of the test data points. This score represents a degree of believe that a particular observation came from an abnormal device. Such a technique produces a ranked list of anomalies, and therefore, the analyst, based on the chosen detection metric, needs to define a threshold to decide whether a point is anomalous or not. *Label* refers to the case where the model assigns a label - normal or anomalous - to every test data point. It is important to note that scoring based anomaly detection techniques gives the analyst the ability to use their domain-specific experience and knowledge to select a threshold that allows for most of the relevant anomalies to be detected [55].

2.2.4 Putting it together

To put the pieces together, and to better understand the basic concept of *anomaly detection*, we provide the following formulation of the problem in mathematical terms. Suppose we have a phenomenon that we want to *monitor*. The collected data points that describe the phenomenon is arranged as in the following set: $X = \{x_1, x_2, x_3, ..., x_i, ..., x_m\}$. This means that the set contains m observation points, and x_i represents the i^{th} data point of the data set. Further suppose that each data point is characterized by two features $x_i = [f_1, f_2]$. Under normal conditions, these data points represent a normal behavior of the monitored

phenomenon, and the data set X should follow a probability distribution P(x), a Gaussian or multivariate Gaussian for instance. If we visualize this in 2 dimensions space, we notice the shape/distribution shown in Figure 2.6a. If we project those (data transformation) on the f_1 axis, then we get the distribution shown the lower part of Figure. 2.6a.

For simplicity, if we apply machine *analysis* on the projected points, we can fit a normal distribution with mean μ and standard deviation σ to model the distribution of the data set.

$$\begin{aligned} \mu &= \frac{1}{m}\sum_{i=1}^{m} x^i \\ \sigma^2 &= \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu)^2 \end{aligned} \tag{2.1}$$

The density for a new observation x can then be computed by:

$$P(x) = \frac{1}{(2\pi)^{\frac{1}{2}} \times \sigma} \times \exp^{-} \frac{(x-\mu)^2}{2\sigma^2} \tag{2.2}$$

In this example, an observation is classified as anomalous if it lies in a low probability region, as shown in Figure. 2.6a. The *decision* of whether this point is anomaly or not is based on the threshold ϵ chosen by by using a validating set with both types of labels and picking one that gives the best performance.

Anomaly Detection Techniques and Metrics

In this section, we focus on the most widely used supervised anomaly detection algorithms. The objective is to explain all the techniques that are used throughout this thesis. Most of these techniques appear in the last sections of Chapter 4. We explain in some details how each of the mentioned algorithms work. Then we discuss the used metrics to evaluate the performance of machine learning algorithms.

Supervised Anomaly Detection

In supervised anomaly detection problems, there must be a feature matrix X with m columns (number of observations) and n_x rows (number of features). For each x_i, there is an assigned label y_i. In general, the terminology in any supervised machine learning problem is as: $D = (x_i, y_i), i = 1, ..., m$. This translates to: D is the data set which has m examples (usually is used for training and testing), and the ground truth y_i in the case of anomaly detection can be: $\{0, 1\}$. y_i refers to the label of the i_th observation (x_i). Moreover, the 0 label corresponds to the negative class (**not anomalous**), and the 1 label corresponds to the positive class (**anomalous**). In the context of anomaly detection, the label 1 is rare. The skewed classes problem (also called class imbalance) can be delt with in many different ways, depending on the classification algorithm that is used. In the next subsections, we will present the classification algorithms and methods that were used. Note that supervised learning algorithms have been rarely used for anomaly detection in the literature, because observations of the rare class are not available most of the time.

I. **Logistic Regression** Logistic regression is a classification technique that works, in its simplest form, as following: given an observation x, and a vector θ, it returns a probability that this observation is in class 1 [101]:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^{\mathrm{T}}.x}} \tag{2.3}$$

The weight vector θ is determined by solving an optimization problem. The objective is to minimize a cost function over all of the training data set. Common used cost function is negative log-likelihood, shown in Equ. 2.4.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{2.4}$$

II. **k-Nearest Neighbors (kNN)**

It is one of the simplest non parametric classification method. The basic assumption this method is built on is that, it assumes that the data points are in a feature space. This assumption allows the notion of distance to be applied. The parameter k refers to the number of how many neighbors (where neighbors is defined based on the distance metric) influence the classification decision. Note that k is usually odd to prevent tie situations

For example, given a new data point x, the algorithm finds the k closest points, and use their labels to assign the new point to a class. A more formal formulation is given by Equ. 2.5. The kNN classifier, based on the distance d between the k neighbors and the point x, outputs/predicts a probability as to which class the new data point x belongs to. The used distance d can be any distance metric, such as the Euclidean (given in Equ. 2.6), Manhattan, Chebyshev, and Hamming distance.

$$P(y = j|x, D) = \frac{1}{k} \sum_{i \in N_k(x,D)} I(y^{(i)} = j) \tag{2.5}$$

$$d(x, x') = \sqrt{(f_1 - f'_1)^2 + (f_2 - f'_2)^2 + ... + (f_{n_x} - f'_{n_x})^2} \tag{2.6}$$

where $N_k(x, D)$ is the set of the k closest neighbors in the training set D and $I(x)$ is the indicator function which output 1 when the input x is true and 0 otherwise.

The parameters (k, the cutoff threshold ϵ, and in some cases the chose of the distance metric) in this algorithm are tuned using cross-validation or validation. A common practice is to perform normalization to all the features of the data set since the distance is highly influenced by the scale of the variables [101].

III. **Support Vector Machines (SVM)**

It is a very popular regression and classification method, and is considered to be among the best "off-the-shelf" supervised learning algorithms. The objective of this algorithm is to separate data points (into two classes for example) using a *hyperpalne* and to maximize marginal distance (optimal distance) between the classes. Optimal

margin means that SVM search for the *hyperpalne* that not only separates the two classes but it also finds the one guarantees it lies exactly at the same distance from the two class, as shown in Figure. 2.7.

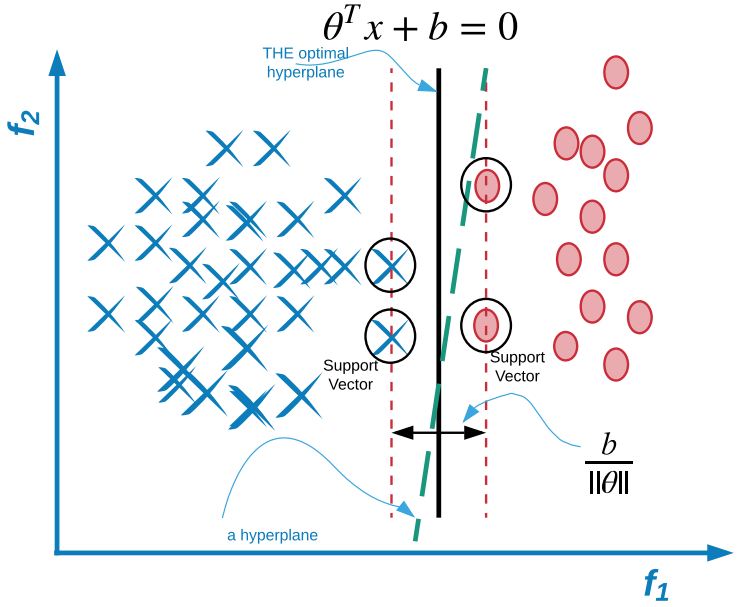


Figure 2.7: SVM: the optimal *hyperpalne* is marked in black, and the dashed green *hyperpalne* is any hyperpalne that can separate the two classes

Using what is known as the "*kernel trick*, in the case of non-linearly separable classes, the samples/data points are mapped to a new feature space (usually a very high dimensional space) using a *kernel function* where the classes can be separated using a *hyperpalne*. An important feature of SVM is that the determination of the model parameters corresponds to a convex optimization problem, thus any local solution is also a global optimum.

To introduce the classification problem SVM tires to solve, we use the following mathematical formulation [40]:

- Given a training data set $D = (x_i, y_i), i = 1, ..., m$
- Let labels set be $y \in \{-1, 1\}$

- Let the *hyperpalne* function that represents the decision boundary be: $y = h(x)_{(\theta,b)} = sign(\theta^T x + b)$

To determine the optimal *hyperpalne*, the parameters $\theta \& b$ have to be found. This can be achieved by solving the following:

$$\begin{gathered} \min_{\theta,b} \frac{1}{2}||\theta||^2, s.t. \\ y_i(\theta^T x_i + b) \geq 1, \forall i \end{gathered} \tag{2.7}$$

This is a quadratic programming problem in which the objective function to be minimized is subject to inequality constraints. To solve it, Lagrange multipliers are introduced for each of the constraints, as shown in Equ. 2.8.

$$\begin{gathered} L(\theta, \alpha) = \frac{1}{2}||\theta||^2 - \sum_i \alpha_i [y_i(\theta^T x_i + b) - 1] \\ \alpha_i \geq 0, \forall i \end{gathered} \tag{2.8}$$

Based on Slater's condition for convex optimization, the optimization problem can be rewritten as follows [40]:

$$(dual) \max_{\alpha \geq 0} \min_{\theta,b} \frac{1}{2}||\theta||^2 - \sum_i \alpha_i [y_i(\theta^T x_i + b) - 1] \tag{2.9}$$

By solving for optimal θ *and* b as function of α, and substituting their values back in Equ. 2.9, with some simplification we get:

$$(dual) \max_{\alpha \geq 0,\ \alpha_i y_i = 0} \sum_i \alpha_i - \sum_i \sum_j \frac{1}{2} \alpha_i \alpha_j y_i y_j (x_i^T . x_j) \tag{2.10}$$

In the case where the classes are non-separable, the constraint in this optimization problem is relaxed. This is maintained by adding a slack variable ζ_i that indicates the amount of the misclassified points. The amount of violation has to be controlled

and kept minimum by penalizing it with a cost parameter C, as formulated in the following equation:

$$\begin{aligned} &\min_{\theta,b,\zeta} \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{m}\zeta_i, s.t. \\ &y_i(\theta^T x_i + b) \geq 1 - \zeta_i, \forall i \\ &\zeta_i \geq 0, \forall i \end{aligned} \tag{2.11}$$

By introducing Lagrange multipliers, switching the max and the min and solving for θ, we get the dual optimization problem:

$$\begin{aligned} (dual)\ \underset{\alpha}{maximize}\ &\sum_i \alpha_i - \sum_i \sum_j \frac{1}{2}\alpha_i\alpha_j y_i y_j (x_i^T . x_j), \quad s.t. \\ &\sum_i \alpha_i y_i = 0 \\ &C \geq \alpha_i \geq 0 \end{aligned} \tag{2.12}$$

What changed for the case of non-separable data is that an upper bound (C) is added on α_i. An intuitive explanation is that, without the slack variable ζ, α_i could go to infinity when the points are misclassified (violation of the constraints). Moreover, the cost parameter C limits the variable α_i so the misclassifications are tolerated.

The structure of the data is not always simple and can be separated with linear *hyperpalne*, for instance. This means that structure of the data requires different boundaries to ensure a proper classification. In SVM, this is achieved using the kernel trick. The idea behind kernel trick is that every data instances is mapped into a feature space of infinite dimension using a kernel $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$. In this new space, an optimal *hyperpalne* can be found. The interesting part is that we do not need an explicit formula for the ϕ function. For more elaboration about the mathematical formulation and optimization of the non-separable case, the reader is referred to [40] and [206].

The tuning of SVM is performed by the proper chose of the cost parameter C and the type of kernel function along with its parameters through cross-validation testing.

IV. **Classification Decision Trees**

Decision Trees are an important type of algorithm for predictive modeling machine learning. How classification trees workis as follows: it starts by partitioning the data space X into subsets. The partitioning process starts with a binary split and continues until no further splits can be made. Various branches of variable length are formed. After a number of runs, a tree (classifier attributes) with certain features is built to fit the training set.

More formally, decision tree is a classifier in the form of a tree structure. Suppose a new person is randomly selected by a research. How can we tell whether this person is fit or not? To answer this question, we can pose a series of questions about the characteristics of people in general. We can start with how old the person is? If he/she older than 33 year old, then we can ask a follow-up question: Does the person go to gym daily? Those people who are older than 33 years and go to the gym daily are definitely fit. On the other hand, if a person is yunger than 33 years and eats a lot of pizzas are likely to be unfit!.

The main components in a decision tree are:

- Root Node: It has no incoming edges and zero or more outgoing edges. Usually it specifies a test question on a single attribute that has the most distinguishing information.
- Internal nodes: These nodes has exactly one incoming edge and two or more outgoing edges.
- Leaf or terminal nodes: Each of these nodes has exactly one incoming edge and no outgoing edges. They indicate the value (class or label) of the target attribute
- Path: a disjunction of test to make the final decision

A new data point x can be classified straightforward once a decision tree has been built. Based on its features vector and starting from the root node, an appropriate label is assigned according to the outcome of the test questions in each of the nodes[40][101]. Decision trees are successful and powerful machine learning techniques; however, they very prone to overfitting.

Before exploring the techniques that solves the problem of overfitting, the next question is how to choose a good attribute/feature that splits the data the best. In other words, a good attribute is the one that has good amount of information to classify the distribution of examples in each node. For that a measure to quantify the amount of information contained in each attribute is needed. The common used one is the *Entropy*.

Entropy in the information theory field refers to the amount of information that is contained in a random variable. Maximum entropy is when all the classes are equally likely, and zero entropy (no information) is when all the classes are deterministic (all examples of the same class, for example). Entropy is the only function that satisfies all of the following three properties:

- When node is pure, measure should be zero.
- When impurity is maximal (i.e. all classes equally likely), measure should be maximal.

To over come issue of overfitting, the concept of *Pruning* was introduced.

Pruning refers to the process of shortening the branches of the tree. It reduces the size of the tree by turning some branch nodes into leaf nodes, and removing the leaf nodes under the original branch. Lower branches may be strongly affected by outliers. Pruning enables finding the next largest tree and minimize the overfitting problem. A simpler tree often avoids overfitting. The most common staregaies are: (i) *Early Stopping*, which refers to the idea of stopping the growth process prematurely. (ii) *Post-Pruning Post-Pruning*, which refers to growing a full tree that captures all possible attribute interactions, later remove those leafs/branches that are due to

chance. For more details on Classification Trees and Regression (CART) and the commonaly used algorithms (ID3 and C4. 5), the reader is refered to[46][178][114].

V. **Random Forests**

Random Forests comes under the umbrella of ensemble learning methods. In a nutshell, Random Forests grows a collection of decision trees, and each tree (classifier) is built based on a selected subset of the attributes. To classify a new point, the input vector is pushed down each of the trees (classifiers) in the forest. Based on the attributes questions in each tree, each tree gives a classification decision (vote). Then using the majority "vote", the forest chooses the label that have the most votes (over all the trees in the forest).

Random Forests' main idea is to minimize the high variance problem that traditional classification trees suffer from. This is achieved through averaging the variance of many estimates.

The formal way of how Random Forest works is as following: The key idea behind random forest algorithm is the introduction of two randomizing aspects to the process of building a traditional decision tree based model/classifier. This idea results into reduction of the correlation between constructed trees.

- Random samples from the data set. In general, a bootstrap sample consists of number data points that are randomly selected from the data set with replacement. In random forests, a number of bootstrap samples each containing a subset of the training data are used to generate a sequence of decision tree classifiers.
- Random attributes selection. The second randomizing aspect the random forest algorithm applies is selecting a random subset of attributes/features for each decision tree it constructs.

The Random Forest algorithm is summarized in the following steps:

1 Prepare a sample $Z = (X_Z, Y_Z)$ from configuration set $D = (X, Y)$.

2 Draw a sequence of bootstrap data sets $Z^{(B)}$ from the sample $Z = (X_Z, Y_Z)$.

3 Learn/fit trees $h_B(x)$ on $Z^{(B)}$, in particular select p features randomly out of n_x features as candidates before splitting for each tree.

4 Output prediction results: (i) for Regression, average all individual predictions of the sequence of regression trees $h(x) = \frac{1}{B} \sum_{allZ^(B)} h_B(x)$. (ii) for classification, apply majority vote.

Describing all of these methods in detail is beyond the scope of this work. The interested reader is referred to, e.g. [40],[55],[101].

Evaluation Metrics

In any anomaly detection technique, there must be a proper metric to evaluate the performance of a detection method. Table 2.2.4 summaries the commonly used measures of detection performance.

The first measure to highlight in this section is Detection Rate (DR), also called Accuracy. It represents the ratio of the correctly classified samples (True Positive - TP and True Negative - TN) over all of samples. Another very important measures are Sensitivity and Specificity. In the case of imbalanced classes, which usually the case in anomaly detection, Accuracy is not a useful or a meaningful measure any more. However, Sensitivity and Specificity are. Sensitivity, also called Recall, translates to the proportion of correctly identified positives (anomalous samples), and specificity represents the proportion of correctly identified negatives. Precision and recall are not adequate for showing the performance of detection and are even contradictory to each other, because they do not include all the results and samples in their formula. Consequently, the commonly used measure in anomaly detectoin is F-score (i.e., F-measure). It is calculated based on precision and recall in order to compensate for this disadvantage.

Receiver Operating Characteristic (ROC) is a statistical plot that depicts a binary detection performance while its discrimination threshold setting is changeable. The ROC space is supposed by FPR and TPR as x- and y-axes, respectively. It helps us determine trade-offs between TP and FP, in other words, the benefits and costs. Since TPR and FPR are equivalent to sensitivity and (1-specificity) respectively, each prediction result represents one point in the ROC space. The point in the upper left corner or coordinate (0, 1) of the ROC curve stands for the best detection result, representing 100% sensitivity and 100% specificity. This point is also called the perfect detection. An Area Under the Curve (AUC) is usually between 0.5–1.0; the bigger it is, the better the detection. Different measurements could be contradictory with each other. It is hard to meet high precision and recall at the same time. We need to make a trade-off to balance them. Thus, the F-measure, i.e., F-score, is often used to indicate detection performance.

Measures	Description
True Positive (TP)	Anomaly behavior is correctly identified as anomaly, i.e., True Positive = correctly identified.
False Positive (FP)	Benign behavior is incorrectly identified as anomaly, i.e., False Positive = incorrectly identified.
True Negative (TN)	Benign behavior is correctly identified as benign, i.e., True Negative = correctly rejected.
False Negative (FN)	Anomaly behavior is incorrectly identified as benign, i.e., False Negative = incorrectly rejected.
Recall: True Positive Rate (TPR)	$TPR = \frac{TP}{TP+FN}$, i.e., sensitivity or recall means the benefits we gain.
True Negative Rate (TNR)	$TNR = \frac{TN}{TN+FP} = 1 - FPR$, i.e., specificity means the costs we spend.
False Positive Rate (FPR)	$FPR = \frac{FP}{FP+TN}$, false alarm rate.
Precision: Positive Prediction Value	$P = \frac{TP}{TP+FP} = 1 - FPR$, i.e., specificity means the costs we spend.
F-score (F-measure)	$F - score = F - measure = (1 + \alpha^2)\frac{P \times TPR}{\alpha^2(P+TPR)}$, α is a predefined parameter.
Accuracy = Detection Rate	$Acc = \frac{TP+TN}{(TP+FP+TN+FN}$.
Receiver Operating Characteristic (ROC)	Defined by FPR and TPR as x- and y–axes, respectively, it helps us determine trade-offs between True Positive and False Positive, in other words, the benefits and costs.
Area Under the Curve (AUC)	$AUC = \int_{-\infty}^{+\infty} TPR(x)FPR(x)dx$

2.3 Literature Review of Existing Research

2.3.1 Security threats

In this subsection, we discuss the work about detecting device's anomalous behavior due to security threats. In this context, security threats detection refers to the process of monitoring events/behavior on a host or a network to detect (and in some cases act upon) a malicious program/software that aims to cause some sort of harms [93]. It can be noticed that there is a big body of literature that focuses on malware detection; however, we limit our review in this thesis to studies that mainly consider the *side-channel information* and adopt *dynamic analysis* to detect malware. More specifically, we review only the studies that investigated various techniques to detect malware in *embedded and wireless devices*, namely, smart grid systems, software-defined radio (SDR), smartphones, general-purpose computers, IoT, CPS, and medical devices. We organize the research in this area based on what information has been analyzed, as follows:

CPU utilization, API calls, and Network traffic based approaches

Shabtai et al. [197] proposed a host-based malware detection technique where they collect on the device several features and events while executing an app on a smartphone. Their framework, Andromaly, processes the collected observations using a supervised machine learning technique to identify whether an app is normal or malicious. Their feature vector includes different aspects of the device, such as CPU utilization, number of running processes, and API calls. The reported performance shows that their approach is capable of classifying an app as to whether it is a game or a tool with an accuracy of 99.7%. However, they do not show or mention whether the classified apps are actually malicious or not. In a similar study, Burguera et al. [50] developed a device level malware detection technique. They designed a tool - named Crowdroid - that collects the calls of the system while running apps on an Android smartphone. The tool sends the monitored behavior, represented by Linux system calls, to a central server (off-device analysis) to detect malware. The

central server, in turn, analyzes (dynamic analysis) the data and creates a system call vector. In the final stage, the server uses the collected dataset to train a k-means clustering algorithm. Specifically, they have employed 2-means clustering algorithm to distinguish between normal apps and their malicious replicas. On the other hand, some authors have used on-device measurement to measure and collect datasets. In a recent work, Cui et al. [66] designed a cloud solution to identify malware by analyzing the network packets sent by the device. Their proposed system, namely, service-oriented mobile malware detection (SMMDS), analyzes and compares the packet information with previous information extracted from malicious and benign applications using clustering.

Electromagnetic Information (EM) based approaches

The usage of EM radiation for intrusion detection was investigated in several studies. For example, Sehatbakhsh et al. [196] have leveraged the idea of analyzing EM signals to detect anomalies in embedded devices. Their framework externally monitors a device in order to collect its emitted EM signals while it is performing its regular tasks. In the learning phase, the collected signals are converted into a sequence of sample spectra (the power spectral density using STFT) which are then used to extract features. These features are basically the number of spikes in the spectrum that satisfies a certain threshold. In the detection phase, the features are extracted from a measured signal and then checked to see if it came from the same statistical distribution of the training dataset. If not, then, with a certain degree of confidence the device is flagged to be compromised based on the resultant anomalous behavior. Following the same concept, Khan et al. [130] have explored the idea of using deep learning techniques to model the behavior of a device using its EM emissions. In [130], the authors trained a multi-layer neural network on EM signals obtained from an uncompromised device to learn its normal behavior. Once the device encounters any deviation in its activity, the model flags it as an anomaly. Validations show that the framework has 100% accuracy in most of the test cases that they applied.

In 2019, Riley et al. [186] developed an approach that considers an IoT's processors

EM radiations to detect when changes in a device's firmware are encountered. Their methodology is based on analyzing the correlation between the generated EM signals. The idea is that when bits from a program counter and the instruction register are altered, such an event can be detected. Finally, in a recent study Khan et al. [131] presented IDEA, an intrusion detection framework that is able to detect malware in embedded CPS devices. In the training phase, they build a dictionary of EM signatures obtained from a device executing legitimate activities. In the detection phase, an EM trace is compared, using 1-Nearest Neighbour algorithm, to the dictionary of words to tell whether the device has been compromised. They tested the methodology on different devices and against different malware; the reported performance shows 100% detection accuracy in most of the cases with less than 1% false positive rate.

Power Information based approaches

Most of the malware detection studies that we came across are about detecting malware in smartphones. For example, Kim et al. [134] were among the first to highlight that the power consumed by devices can be used to detect anomalies which were otherwise difficult to detect by only analyzing the static characteristics of an application. Their prototype, built on Windows phone, works by leveraging power signatures of the phone. These signatures are based on the power consumed by the device while running a program. In their work they consider the amount of the power consumed when executing a program - and not the behavior of how power changes as the code is executed. They show that the methodology is able to detect numerous malware with at least 80% detection accuracy.

In 2014, Zefferer et al. [241] proposed a different methodology in which they extracted the power consumption of an emulated malware running on a smartphone using a tool called PowerTutor [243] (i.e., using on-device monitoring). The concepts of Mel Frecuency Cepstral Coefficients (MFCC) was utilized to extract features from each power trace, and then they fit a GMM (Gaussian Mixture Model) to classify the power traces. This methodology performs well to recognize emulated malware with an accuracy of 92%, but one drawback

is that the power consumption was taken with a low sampling rate. Consequently, such a technique may fail to detect malware that are rarely active. This disadvantage has been tackled in this study using an external tool with a much higher measurements sampling rate (5000 samples/sec), and changing the way the features are extracted and the way the classification is performed.

Following a very similar approach, Yang et al. [231] monitored the power consumption of Samsung GALAXY S5 and LG G2 smartphones using an on-device tool (PowerTutor [243]). They collected the CPU power consumption signals and used them to extract a feature vector; the extracted features are MFCC coefficients. Then they perform a waveform feature matching using a GMM model. They claim that the learned model can detect malware with 79% accuracy; moreover, the model can classify the category - whether the app is a game, browser, or music - of an app with accuracy of more than 65%.

Robin et al. [122] proposed a methodology for malware detection that applies Independent Component Analysis (ICA) on a smartphone's power signals. The idea is based on the hypothesis that the power consumed by a device, when it is running a benign app, is the sum of some noise plus the needed power to execute the actual app. However, when a device is running a benign app while also running malware in the background, the power consumed by the phone is the sum of both apps: power consumed by a benign app running in the foreground and the power consumed by an emulated malware running in the background. Therefore, by using ICA, the methodology can separate the two signals and hence can detect malware. The obtained detection accuracy on an emulated malware based dataset was 91%.

Dixon et al. [77] used battery information of a smartphone and the user's location to detect abnormalities. Hoffman et al. [77] conducted experiments to investigate whether the power information of a smartphone can be used as a good source of information for malware detection. The conclusion in their study indicates that the extra power consumption caused by a malware is insufficient to profile a device or an application. They concluded that battery life monitoring is mostly not satisfactory for malware detection on modern smartphones. While their claims might be correct given the fact that their claim is based

on on-device PowerTutor application [243], our measurement approach is much different and our measurement sampling frequency is much higher (5000 samples/sec) which, in turn, felicitates higher chances of detecting events that PowerTutor fails to detect.

Caviglione et al. [54] studied attacks related to covert channel using the power consumption of the processes running. A covert channel happens when two or more malicious applications exchange information exploiting different permissions assigned to them. To illustrate, let one app have the permission to read the phone numbers, and the other application have the permission to use the network. Both applications can cooperate to transmit a user's phone numbers, and traditional static or dynamic approaches can fail to detect 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 an emulated covert channel attacks using PowerTutor with high-level and middle-level information. To discriminate malicious and non-malicious behavior, they used neural network and decision tree approaches obtaining accuracies of above 80% in general.

Azmoodeh et al. [29] presented a machine learning based approach to detect ransom attacks using power consumption of Android services. Similar to most of the smartphone malware detection papers, this study uses PowerTutor to collect the power traces. They use k-Nearest Neighbors, Neural Networks, SVM, and RF as detection techniques. Liu et al. [151] in 2016 developed a power side-channel strategy to detect anomalous behavior in control flow execution applied to IoT microcontrollers. They measured the variations in the power consumption in the VCC pin of the microcontroller through a resistor connected to the power supply. In this work, it was tested insertion, deletion and replacement on AES PROGRAM to prove firmware modification attack.

Nash et al. [171] developed a methodology to estimate the power amount of a desktop computer by using machine learning algorithm. By taking into consideration a number of features, such as CPU activity, storage access, and network traffic, they employed a multiple linear regression model to model a desktop's consumed power. They validated their technique to detect malware in desktops. The results proved the concept can be effective, however, it was necessary to combine the obtained features with the OS data

counters. While the technique showed potential, it is strongly intrusive based on the information needed to perform the detection. Moreover, it is the complexity to compute proper numerical values for the needed thresholds for detection decisions.

Similar to Nash's work, Jacoby and Davis [120] showed how to analyze the power consumption and CPU utilization to detect network attacks by using a battery-based Intrusion Detection System (IDS). Bridges et al. [48] analyzed CPU power consumption of a general-purpose computer to detect malicious software. They monitor an uninfected device performing a fixed task for a certain period of time in order to learn its normal behavior. Then the collected signals are processed to extract three features, namely, statistical moments, L^2-Norm Error, and Permutation entropy. Finally, a classifier is trained on the extracted features. The reported results show high detection performance ($\sim$ 100% true positive rate - TPR) when using an anomaly detection ensemble.

2.3.2 Fault Detection

In the context of a device's fault detection, we find that the literature is rich with techniques and frameworks that present the topic under the umbrella of system's resilience or fault analysis [36]. Diving into the definition of a system's resilience [211], one can find that a main milestone to achieve resilient systems is through controlling (detecting, identifying, and correcting) faults of and threats to devices, while the system is still correctly operating [148]. Therefore, the main step to improve the availability of the overall system is through finding out when a device starts to misbehave or behave anomalously (due to a faulty component, for example). We group and discuss the conducted research on a device's fault detection and identification as follows:

Contextual Information based approaches

Chio et al. [60] proposed DICE, a framework to detect faulty devices in smart home systems based on a device's context (e.g., a sensor's collected data). In the learning phase,

called pre-computation phase, the method extracts the context (correlation and transition probability) from sensors that make a smart home system. Assuming that the devices are faults free and that the collected data integrity is preserved, DICE learns the normal behavior of the sensors that make a smart home system. The reported results show a high detection performance (95% precision and 92.5% recall). Such solutions demand sensor domain knowledge along with the contextual information and historical data from similar near-by sensors.

Lin et al. [147] developed SensorTalk, a methodology to detect potential sensor failures in the field of smart farming. By performing multiple mutual tests between measurements collected by sensors, a failing device can be identified. The study uses analytic and simulation models to appropriately select the time of when a potential failure is about to occur. Their results show a low false detection probability $< 0.7\%$.

Alippi et al. [14] introduced a framework to detect changes at the sensor level in the domain of CPSs. The methodology builds a predicative model of the expected sensor's generated signal over time. Then, it keeps inspecting the actual signals (datastreams) coming from different sensors and the model's forcasted signals to detect any possible changes. The way it checks for changes is based on change-point detection technique called Intersection-of-Confidence-Intervals proposed in [15].

Antunes and Neves [22] have presented a methodology to detect software flaws in network servers based on their behavioral profiles. The methodology starts with a data (e.g., system call, memory usage, and OS APIs) collection step in the learning phase. The collected data is used to build a profile of the normal behavior of the server using finite-state-machines. In the operation phase, while monitoring the server, any flaw that causes the behavior to change is detected.

2.3.3 Discussion

Table 2.1 summaries the work done in the area of anomalous behavior detection in embedded and wireless devices. The takeaway from Table 2.1 and this section is two fold:

(i) Regardless of the type of the system, namely, IoT, autonomous cars, CPS, and UAV, the core element that serves as the first point of target of a compromised/faulty system is embedded/wireless devices. While most of the reported work touches upon malware detection in smartphones, there is a considerable body of work that focuses on malware and fault detection in IoT and CPS devices. A main attribute that differentiates these studies is how the device is monitored and what aspects of its behavior is being observed, as Table 2.1 illustrates. (ii) From the reviewed papers, the most common shortcomings are as follows:

- The monitoring tool is coupled with the device being monitored (on-device monitoring), which raises several flags, such as data integrity, resources limitation, and computational overhead [54, 29, 241, 231].

- Although some studies are based on off-device monitoring, it may be noted that those require a degree of invasiveness. To elaborate, a physical access to the device's CPU input power is required in some of the power based anomalous behaviour detection techniques [151, 48]. In another example, EM based approaches require physical proximity to the device's CPU, along with proper alignment and setup [131, 130, 196]. Such invasive approaches become questionable when it comes to usability and practicality. According to their experimental setup, EM receiver has to be placed close to the CPU in order to collect the EM signals. Now, if the targeted device is packaged in a package that suppresses the EM radiation or even weakens the radiated signal, then the applicability of these approaches becomes problematic. Moreover, the EM receiver itself might be the subject of an external attack (e.g., using signal jammers), which could jeopardize the whole idea of EM based detection.

- An important factor that is not given enough attention in some of the power based approaches is the data sampling rate. Regardless of how effective and creative the proposed analysis approach is, ignoring the role of sampling can lead to a bad performance in the detection process. This applies to the studies that make use of

Table 2.1: Summary of Device's Anomalous Behavior Detection Literature

Anomalous Behaviour Detection	Source of Anomaly	Analyzed Info.	Ref.	Type of Anomaly	Application Domain/ (Targeted Device)	Discussed Consequences	Side-Channel Info.? (Monitoring)	Approach - in 3 words & (Performance)
✓	Security Threat	EM	[131] [130]	Ransomware, Control-flow hijack	CPS / (embedded devices: OLinuXio boards)	financial & physical damage	Yes (off-device)	EM emissions + Neural Networks [131] EM emissions + K-NN [130] (∼100% detection accuracy in both studies)
			[186]	Change in framework	IoT / (Atmega328P)	not discussed	Yes (off-device)	EM signals + Hamming distance + SVM (99.0% detection accuracy)
			[100]	Control-flow integrity	ICS / (Allen Bradley CompactLogix PLC)	could lead to catastrophic failures	Yes (off-device)	EM waves + LSTM (98.9% classification accuracy)
			[195]	Attacks on attestation integrity	CPS, SCADA & ICS / (Arduino UNO)	full control hardware/ software	Yes (off-device)	EM + STFT + Thresholding [195] EM + FFT + FSM [196] (100% detection accuracy in both studies)
		Power	[241]	Emulated Malware	Smartphones / (Galaxy S2)	financial losses	Yes (on-device)	CPU power signals + MFCC + GMM (92% malware detection accuracy)
			[231]	Malware	Smartphones / (Galaxy S5 & LG G2)	not discussed	Yes (on-device)	CPU power signals + MFCC + GMM (79% malware detection accuracy)
			[122]	Emulated Malware	Smartphones / (Galaxy S5)	privacy breach	Yes (off-device)	device's power signals + ICA + (SVM, NB, & RF) (∼88% detection accuracy)
			[151]	Control-flow integrity	IoT / (microcontroller: STC89CS2)	key extraction	Yes (off-device)	device's power signals + FFT + PCA + HMM (∼98% detection accuracy)
			[54]	Convert channel attack	Smartphones / (Galaxy S3 & LG X4)	sensitive-data leakage	Yes (on-device)	amount of consumed power + (NN & RF) ∼(from 65% to 96%) detection accuracy)
			[29]	Ransomware	IoT / (Android device)	financial losses	Yes (on-device)	power signals + DTW + KNN (∼92% detection accuracy)
		CPU, API, and Network	[197]	Emulated Malware	Android mobile devices / (HTC G1)	billing & data theft	No (on-device)	(CPU, of packets, API, battery level) + DT-J48 (99% malware detection accuracy)
			[50]	Convert channel attack	Smartphones / (Galaxy S3 & LG X4)	not discussed	No (on-device)	OS calls + k-means (∼100% detection accuracy)
			[84]	Malware	Smartphones / (Android device)	financial & data theft	No (on-device)	(CPU, of packets, memory, & of connx.) + GMM (∼100% detection accuracy)

PowerTutor as a monitoring means (data collection), since the provided sampling rate is < 5 samples per second.

Table 2.2: Summary of Device's Anomalous Behavior Detection Literature: continuation of Table 2.1

Source of Anomaly	Analyzed Info.	Ref.	Type of Anomaly	Application Domain/ (Targeted Device)	Discussed Consequences	Side-Channel Info.? (Monitoring)	Approach - in 3 words & (Performance)
Performance Issue	Context	[60]	Faulty sensor	Smart Home System / (Raspberri Pi)	User privacy exposure	No (off-device)	Sensor data and context + CC + Thresholding (95% precision rate)
		[147]	Sensor failure	IoT / (Weather station & moisture sensors)	inaccurate readings	Yes (off-device)	Sensors data + Mutual Test ("low false positive rate")
		[22]	Software flaws	Data centers / (network server)	not discussed	no (on-device)	(CPU & memory usage, OS calls) + FSM ("few false positives")
		[14]	faults ageing effects	CPS / (ST STM32 Nucleo)	not discussed	Yes (off-device)	Sensors data + ICI (AR predictive model) ("reduced false positive rate")
		[173]	software, hardware, & communication failures	WSN / (TelosB)	environment hazardous	Yes (on-device)	Sensors readings + set of ML classifiers (∼55% to 100% detection accuracy, depending on fault type)

Note: ICS = Industrial Control Systems, UAV = Unmanned Aerial Vehicle, CAN = Controller Area Network, PDA = Personal Digital Assistant, EM = Electromagnetic, GPC = General-Purpose Computer.

2.4 Summary

In summary, this chapter builds the context for this study and can be divided into three sections: Section 2.1 the IoT architecture has been explained, along with most commonly used protocols. This serves to frame the context of this thesis and serve as a preamble for detecting anomalous behavior in IoT Devices. Section 2.2 provides an overview of supervised anomaly detection techniques and the common measures to evaluate them. And finally, Section 2.3 presents the related work and discuss how each method works.

Chapter 3

Models and Assumptions

In this Chapter we discuss the models and assumptions used throughout the proposed detection framework. The discussion covers a generic system model, anomaly model (threats and faults), problem definition, and the solution strategy.

3.1 System Model

Since this work aims at detecting anomalously behaving devices in the general sense, it is important to show a generic model that includes an abstract representation of CPS/IoT application - we call it Cyber-Physical space. Therefore, we use the following models to explain the high-level view of the components that can be found in our solution strategy as well as any cyber-physical system. The cyber-physical space term covers how devices, the main focus of this thesis, are integrated to/in build/ing complex systems? We show also how the monitoring approach, that is used to collect the behaviors of devices, is instrumented to devices in such a way that satisfy the main challenges addressed in this thesis. Finally, the system model shows how the detection engine, which performs the data analytic, is positioned in real case scenario. The purpose of the components in this section is to set up the intuition and build the necessary context for next chapters.

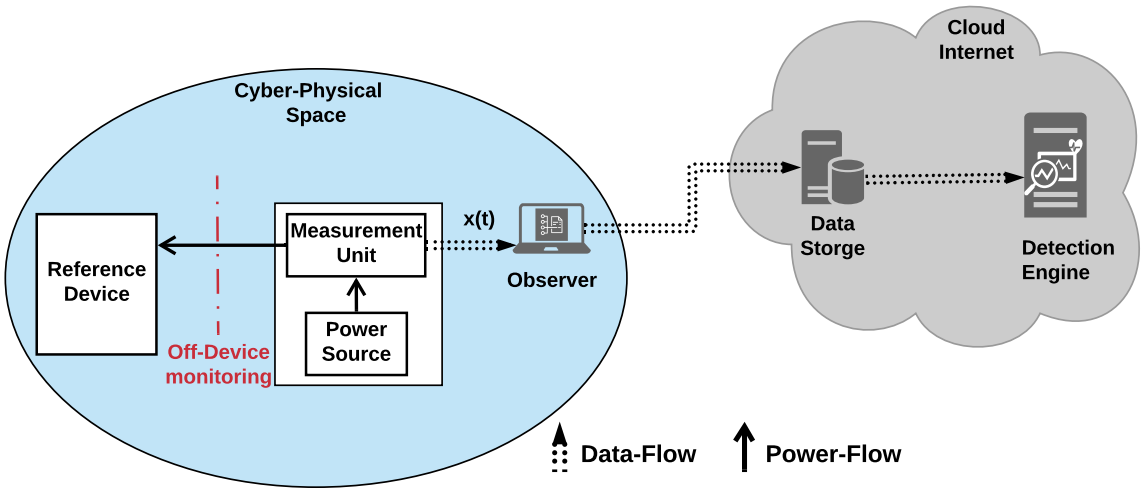


Figure 3.1: System model

3.1.1 Architecture Model

Figure 3.1 shows a high-level view of the architecture model of the proposed solution strategy. We start off by defining some terminology.

1. *Cyber-Physical Space*. This term refers to any cyber system or cyber application that is built on top of or based on a technology to allow autonomous/semi-autonomous operation of the physical system. In this context, a physical system may refer to a wide range of applications (e.g., Fig. 3.2 shows a robotic arm as an example of an IIoT application). Cyber-physical space primarily revolves around the pervasive instrumentation of physical objects (robotic arms in our example in Fig. 3.2) with devices (mostly embedded) that are equipped with sensors, actuators, and the connection of those devices to the Internet. We refer to these devices in Fig. 3.1 and Fig. 3.2 as "reference devices". It is important here to stress the fact that the core layer of establishing a cyber space are the what we call reference devices.

2. *Reference Device*. It is basically a hardware device (e.g., embedded device [113] or PLCs) instrumented to a physical object (robotic arm in our example shown in Fig.

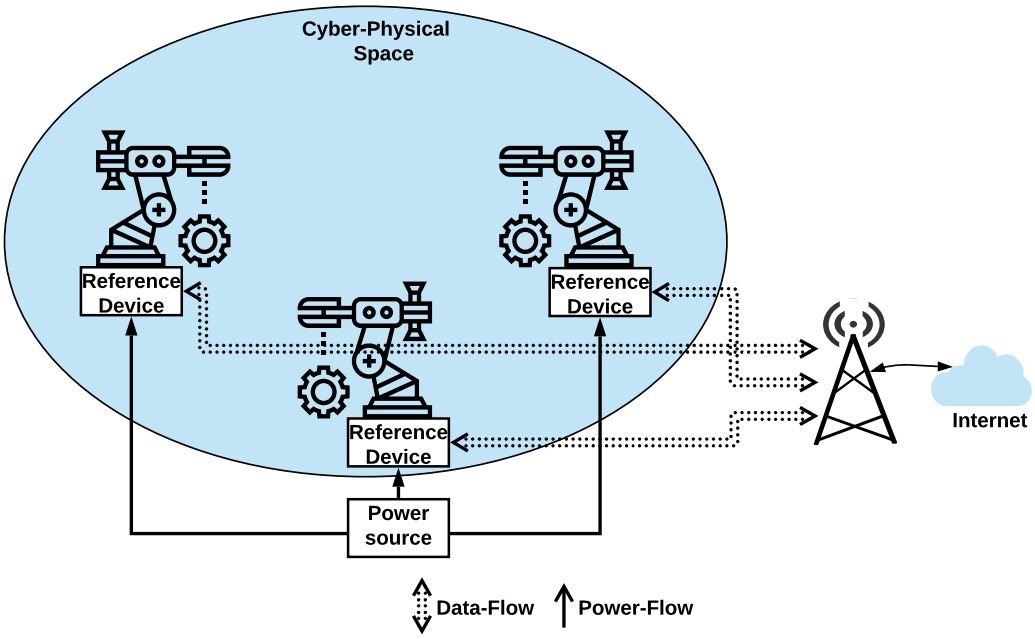


Figure 3.2: An example of an IIoT cyber-physical space

3.2) and is designed for one or very few specific task(s). As shown in Fig. 3.3, such a device contains a special-purpose computing element that runs software. It is also equipped with network module that can support one or more (wired/wireless) communication technologies (e.g. ZigBee, Cellular, WiFi). Such a device works as a main block in a larger system for monitoring and controlling purposes. To perform its tasks, a reference device interacts with external environment physically or logically through sensors and actuators. Finally, the reference device requires a power source (battery or power system) to operate and function. It is important to mention here that the examples that can be covered by the given reference device's model can be found in a large range of applications; however, in the context of this thesis, we target IIoT, CPS, and MCPS (Medical-CPS) applications/systems. In other words, the focus is on devices that have limited direct human interactions.

3. *Device's Software*. It is a software component (operating system (OS)) needed to operate the different components within the device. In the normal operation mode,

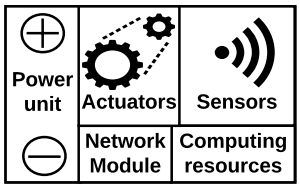


Figure 3.3: An illustration of a reference device

the software is responsible on performing the following: (i) it controls the operation of sensors or actuators; and (ii) it processes the collected data or the received commands. For example, in the case of sensing, it reports the collected data to the server via the network interface. And in the case of actuating, on the other hand, it processes the received commands from the server to actuate the physical object to perform the desired actions/tasks.

4. *Power Monitoring Unit.* This is a device that can be connected to the power source of the monitored device. The assumption is that the power monitor unit causes no overhead to the actual operation of the device itself (i.e., off-device monitoring approach). Moreover, it has a reasonably high sampling rate to capture the short lasting activities (i.e., activities that last for small durations). Its mode of operation is completely decoupled from the monitored/target device in order to ensure that the integrity of measurements is preserved. Lastly, the power monitoring unit is integrated to a reporting system, called observer, that sends the collected power signals to the cloud for analysis (i.e., to the Detection Engine).

5. *Observer.* In our experimental system model, this is a lap-top computer used to store the collected power measurements in a database in CSV format and then upload these data to the cloud for analysis.

6. *Detection Engine.* It is the main block of the proposed framework. The engine resides in the cloud to perform the data analytics and model the device's behavior. The employed data analytics approaches are discussed in detail in Chapter 5 and Chapter 6. Once the engine receives a new power signal, the function of this component is to

give informative insights to the device's operator or owner as to whether the device is behaving normally or anomalously. Therefore, the assumption we make at this point is that there is some sort of a trusted path from detection engine to the device's operators to alert them of any potential anomaly.

7. *Misbehaving Device*. It is a device that exhibits anomalous behavior. The possible anomalies that may lead to a device misbehaving are explained in subsection 3.1.2.

The main focus of this thesis is on devices that perform basic functions as outlined above. Although the reference device's model seems basic and simple, such a device and the performed tasks exist in a wide range of applications. Whether it is self-driving cars, smart parking systems, Industrial IoT, or any cyber-physical system, to name a few, the explained reference model still applies. That is simply because, in all of these systems, we find that tasks like sensing, actuating, and controlling are there.

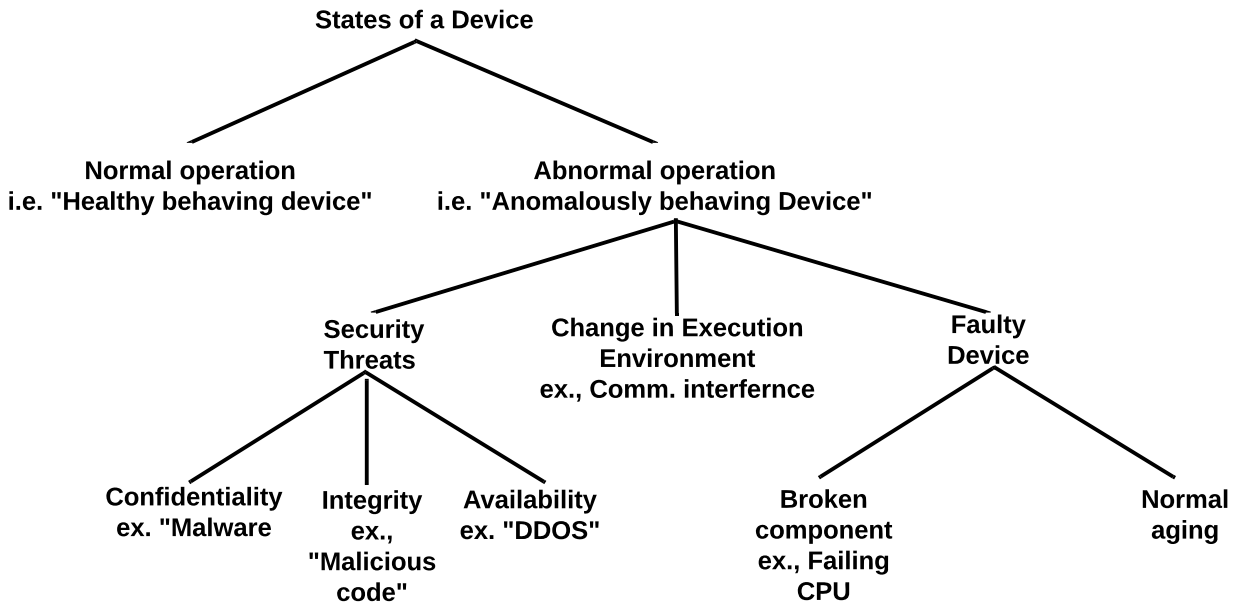


Figure 3.4: Break-down of an IoT device's possible status

3.1.2 Anomaly Models: Threats and Faults

In this subsection we describe the possible causes that may lead a device to behave anomalously. As Fig. 3.4 shows, there are three main causes: security threats, change of the execution environment, and the device's faults. Since our framework is based on anomaly detection, it is important to specify the threat and fault models [233]. We define the sources of the three classes of anomalies as follows:

Security threats

We define the possible threats as a security issue or a harmful event that aims at harming the device. A common characteristic of all security threats (malwares, worms, Trojans, viruses, to name few) is the must for performing actions over the network [85]. In the given example shown in Fig. 3.2, an attack on robotic arms in a production line could lead to delays in manufacturing, potentially causing hundreds of millions of dollars in losses.

The possible security threats can harm a system in one of the following ways:

1. Compromise the *confidentiality* and the *integrity* of the device/user/system through (a) transferring of data to unintended remote servers (e.g., malware and spyware); (b) hajacking of the device's resources to perform tasks for the attacker's desire (e.g., bots: mining cryptocurrency and ransomware).

2. Restrict the *availability* of the device such that the provided services become partially/fully inaccessible (e.g., DDOS and jamming).

In comparison to previous methods, the only assumption we make in our work is that an attacker has the ability to install a threat on the targeted device regardless of the employed protection measures or the used communication protocol/technology. Moreover, the installed malicious code is able to gain full control over the device's resources (software and hardware). Since in our proposed anomalous behavior detection framework, we mainly target Internet-connected devices, such an assumption holds in real systems/scenarios.

Change of the execution environment

The anomalous behaviour due to this anomaly refers to the case where the surrounding of a device impacts its behavior. In such a case, the device is not under attack, nor it is faulty, yet it is behaving abnormally. The causes of such an anomalous behaviour includes firmware updates (external change pushed to the device) [182, 170] or communication interference (external issues that worsen the network availability).

Faulty device

This covers any changes of the device's behavior due to a failure which requires some sort of maintenance. Unlike security threats, the term fault covers the causes that lead a device's behavior to deviate from its norm without involvement of any external doer/intruder. This can be due to: (1) a *broken component* (ex., a failing CPU [160]); or (2) an *aged device* (ex., hardware aging [78]). The faults considered in this model are assumed to not cause a device to shut down (i.e, stop working entirely). Failures that lead a device to stop (e.g., fail-stop model) are out of the scope of this thesis. The reason is that if a device stops working, it can be detected right away once it becomes unresponsive.

.

3.2 Detection Model

This section presents a brief description of the problem and the solution strategy with the main faced challenges.

3.2.1 Problem Description

Traditionally, in industrial plants and manufacturing environments, for example, the control and telemetry components were typically applied within isolated networks. The in-

frastructure in this physical systems were built using specialized operating systems. What made resilient to attacks and failures is the fact that the deployed operating systems and software are relatively masked and unknown to attackers. However, given the exponential growth of the IoT and CPS devices and offered applications, traditional industrial plants and manufacturing environments have been modernized and became more intelligent. The new industrial control systems' architectures are continuously connected to the Internet (via wireless or wired links) for real-time insights and automation. Consequently, the new paradigm has brought new avenues of risks that can compromise control systems and production lines as a whole.

In the last couple of years, 66% of manufacturers have experienced a security incident related to IoT devices, according to [2]. The infamous Stuxnet worm attack against Iranian nuclear uranium enrichment facilities is one example [88]. The Ukraine smart power grid was the target of a cyber-attack which resulted in a three hours power outage [132]. These are just few examples on the type of critical infrastructure that can be targeted globally. These will not be the only attacks, given the fact that the functionality of the targeted devices depend on a direct data exchange. This requirement (i.e., the continuous need for Internet connection) makes these devices become inherently not secure. This also goes back to the fact that IoT/CPS devices are constrained in memory and processing capability; as a result, having complex security algorithms is computationally expensive, and hence, is not viable.

Given the aforementioned scope and motivation, this thesis concerns the problem of detecting anomalous behavior of IoT/CPS devices by considering side channel information.

Statement:
This research has been inspired by the fact that cyber-security attacks are increasingly occurring globally targeting IoT/CPS devices used in critical systems. Effective and robust approaches for detecting anomalous behavior of devices can minimize a tremendous amount of economical and life losses. Therefore, our objective is to propose new approaches that leverages the power consumption of devices and deep learning techniques to make these devices more resilient and proof to different kinds of attacks and failures.

3.2.2 Solution strategy

Power consumption of devices, as a side-channel information source, has been exploited by attackers to reveal secret keys. That is due to the strong and direct correlation between the tasks a computer/device performs and its power consumption behavior. For the same reason, we have chosen to exploit the power consumed by IoT/CPS devices to make them better protected and secured. Therefore, we formulate the problem of detecting a device's anomalous behavior as anomaly detection problem.

As highlighted in the problem description and the system model sections, IoT/CPS devices are constrained in memory and processing capability. Therefore, the first challenge that we tackle in our solution strategy is the way to collect the device's behavior (i.e., its power consumption signals). In order for the solution to not cause overhead to the operation of the device's resources, we monitor the device's power non-intrusively (i.e, off-device).

The second challenge that we tackle by adopting the idea of monitoring the power consumption of devices non-intrusively is the following: In the case of security threats, namely malware invasion, if the malware is a sophisticated one, then they can detect the existence of the intrusive monitoring methods [124, 98]. Consequently, they can block the monitoring process or manipulate the collected data to avoid being detected. Therefore,

such a monitoring approach raises a data integrity flag. In other words, if a device is compromised, then so could be the monitoring means and the collected information themselves [111, 172]. Therefore, by decoupling the monitoring system and the devices to be monitored, we aim to tackle such a challenge.

The main objective of this thesis is to come up with a detection framework that can detect the anomalous behavior of devices in an accurate and efficient way. In this context, *accurate and efficient* detection means that the methodology produces low false alarms and causes no over-head to the operation of the devices. Throughout this thesis, our solution strategy for detection framework relies on the following:

- The fact that IoT, Industrial IoT, and CPS devices are likely to be designed to execute repetitive, and in some cases periodic, tasks;
- The fact that every single action on-board (whether hardware or software driven actions) will be reflected as a change in the monitored "side channel information" - the power consumption of a device;
- The powerfulness of deep learning techniques to learn complex models from raw data.

It is important to mention that the collected signals are the overall power consumption of the device. Such a signal combines the instantaneous power consumed by all hardware components of the device and the software that runs it. As a result, the data contains much noise which make the detection of anomalous behavior quite difficult and challenging.

So next, the challenge is *hand crafted features are exorbitant.* The collected data points can be used as a *raw data* or *processed data*. *Raw Data* means that the collected data points are used as is for analysis. On the other hand, *processed data* means hand crafted features are extracted from the collected data points for analysis. In our case, the raw power consumption signals contain much noise and do not clearly show what frequencies (events) are present. Therefore, dealing with them in their original structure is problematic and achieving reasonably accurate detection and classification performance can not be easily

done [32], when employing traditional detection and classification methods. Thus, the trade-off in such a case is that hand crafted features yields a better detection performance, and at the same time, the amount and the level of complexity involved in such a process should be kept minimal. The need to keep the number of features and their complexity minimal can help reduce the resources (e.g., personnel domain knowledge, time, memory, processing powers, and energy) needed to perform the task.

The following challenge is *accuracy of detection vs novelty of detecting new anomalies.* In the detection process, the main question to consider is the type of the employed detection approach. The answer of such a question depends on the availability of labeled datasets. It also impacts the aim to be achieved, whether it is a high detection performance (accuracy, precision, recall, and F-score) or the novelty of detecting anomalies that causes a device to misbehave. Therefore, in the field of this thesis's topic, there is a lack of realistic power consumption datasets that reflect the normal and anomalous behaviors of IoT and CPS devices. If a labeled dataset is obtained, then supervised detection approaches can be implemented. Such approaches aim at improving the detection performance. However, if unlabeled dataset (or at least one label is known - normal behavior label) is available, then unsupervised or sim-supervised detection approaches can be used. These approaches aim at detecting new anomalous behaviors (e.g., zero-day attack, or misuse detection).

An overview of the detection process is shown in Fig. 3.5. The proposed detection approaches employed in our detection engine are based transforming the raw power signals to another representations and then fit a model on the new representation as shown in Fig. 3.5. We overcome the lack of datasets availability by designing different sets of experiments that covers a wide range of device's anomalous behaviours. This allows to investigate the feasibility of using the power consumption of devices for detecting different kinds of anomalies. It also allows us to evaluate the detection approaches employed in our detection engine. The details of the experimental setup as well as the nature of the collected data are explained in Chapter 4.

The data modeling task, also called training phase or model building, employed in our detection engine leverages supervised and semi-supervised techniques. Our HOG+CNN

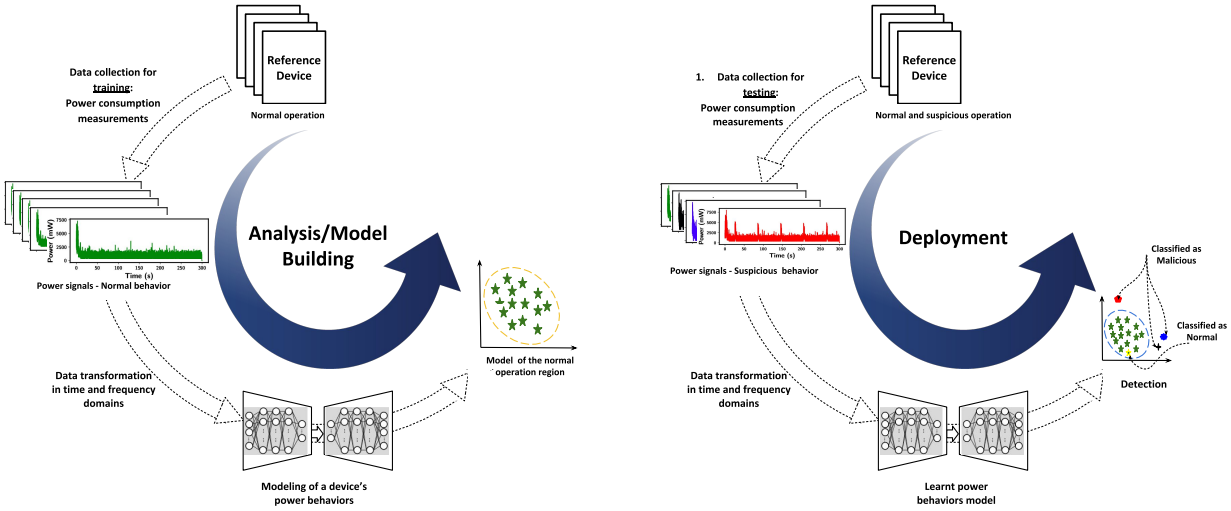


Figure 3.5: Overview of the detection model

approach, discussed in Chapter 5, is a supervised detection technique. It combines: (i) CQT and HOG features, sophisticated features that captures the time and frequency of the events occurring on-board a device, and (ii) a powerful supervised learning technique, namely CNN, to perform the detection. This approach aims improving the detection performance. On the other hand, our RBM+OCSVM approach, discussed in Chapter 6, is a semi-supervised and requires only the normal behavior of a device in the training phase. It is based on: (i) utilizing stacked RBM AutoEncoders to unsupervisedly extract features that are based on the reconstruction errors, and (ii) training an OC-SVM on the extracted features to detect possible security threats. The aim of this approach is to detect new anomalous behaviors that are novel and never been observed (e.g., zero-day attack, or misuse detection).

.

3.3 Summary

This chapter has presented our system model, assumptions, and the problem definition. Then, it explains the solution strategy and how the identified challenges are addressed in our solution strategy. The provided system model includes the architecture model and anomaly model. The architecture model present all the components that builds a cyber-space. It also shows how the proposed monitoring approach can be integrated to any system. A formal definition of the components is given as well as the assumptions made. The anomaly model, in turn, models the sources that can lead for a device to behave anomalously. The anomaly model breaks down the causes into two main causes: security threats and fault device.

Chapter 4

Experimental Setup and Dataset collection

The material of this chapter has been published in parts in our works published in [10, 8, 122]. The main objective in this chapter is to design classes of experiments to be used to build wide spectrum datasets that represent devices' anomalous behaviors. These datasets are to be used to answer the following questions: (i) Can the power consumption of devices be used to detect different kinds of anomalous behaviors?, (ii) how well do the proposed detection techniques perform in detecting security threats and system failures in the domain of IoT and CPS systems?, and (iii) how do the measurements sampling rate and the window size impact the detection performance? Answering these questions validates the effectiveness and robustness of the proposed techniques and the contributions of this thesis as a whole.

The collected data is used to build 18 datasets. These datasets cover three main aspects of a device's security (namely, confidentiality, integrity, and availability), and partial system failures. The security threats datasets are gathered using: (1) 5 real malware applications obtained from the well-known Drebin dataset [26], (2) from 7 families of emulated malware, (3) Distributed Denial of Service (DDoS), where we used the device as a victim

of DDOS and as an attacker (a source of an DDOS attack), (4) Crypto-miner, which is basically a malware that takes over a device's resources and use them for cryptocurrency mining without the user's explicit permission [213]. The partial system failures datasets (3 datasets) are gathered using a faulty CPU cores. Since the work in the domain of this study lacks the availability of such datasets, we aim to make the collected datasets and the developed software publicly accessible for the research community.

4.1 Experimental Setup

We conducted the experiments in our lab using the test bench shown in Fig. 4.1, refer back to Chapter. 3 for real implementation analogy. It consists of the following:

- *Monsoon Power Monitor:* This is a reliable tool used to measure the power consumed by a device in varying test configurations [207]. Essentially, the Monsoon Power Meter supplies power to the device and is connected to an external *laptop* that runs a software called Power Monitor Tool that comes with the Monsoon meter. The software has a GUI to acquire the power consumption data and control the functions of the power monitor. Further, the tool stores the measurements in a *database* in a specific format (csv files) in the cloud (Google Drive in our case). The Power Monitor provides a settable constant voltage and measures the current consumed by the device at 5000 samples per second.

- *Device Under Test (DUT)*: This is the device to be used to conduct the experiment and run one of the sources of anomalous behavior. We change the device under test (DUT) in our test bench depending on the application as we explain in the following sections. The DUT is connected to the *cloud/Internet* through a *router/gateway*.

- *Router/Gateway*: This device is used to establish a wireless connection between the DUT and a server.

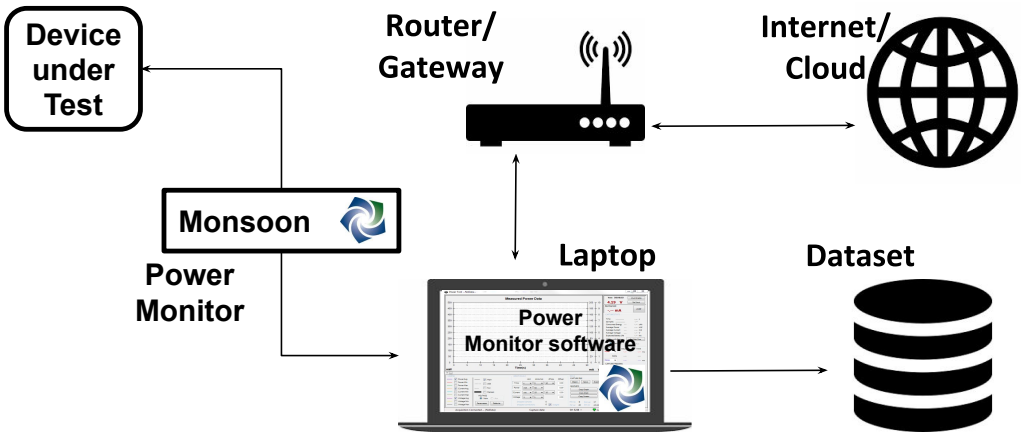


Figure 4.1: Experimental setup test bench

4.2 IoT Based Experiments

The IoT use-case built in our experiment is a Smart Parking system. The system includes an IoT device that monitors whether a parking spot is available. The IoT device is made of a *control unit*, which in our case is a Raspberry-Pi, attached to an *ultrasonic sensor*. The IoT device is connected to a server through a wireless network (a WiFi gateway). The operation of the IoT device is based on the Raspberry-Pi polling the ultrasonic sensor reading, processing the data, and sending the data to the server. The server can send instructions to the Raspberry-Pi (e.g., to start or stop gathering data), and can use or store the data provided to it by the control unit. Moreover, the server can accommodate a web or mobile application so that the end users can interact with the system and get online status of the parking slot. Our focus in this system is the IoT device itself and the activities that it performs. These activities can be summarised as follows:

- The control unit sends a 10 s pulse on the ultrasonic sensor's triggering pin.
- The control unit monitors the ultrasonic sensor's response through the ECHO pin.
- The ultrasonic sensor emits several short ultrasonic pulses, and pulls its ECHO pin high at the same time. The control unit records the time when this has happened.

- As soon as the sensor detects echoes from the pulses it emitted, it pulls its ECHO pin low. The control unit records the time when this has happened too.
- The control unit computes the difference between the two times to determine the round-trip time for the ultrasonic pulses to reach the object and return to the sensor, and divides this by 2 to get the one-way time for the ultrasonic pulses to reach the object.

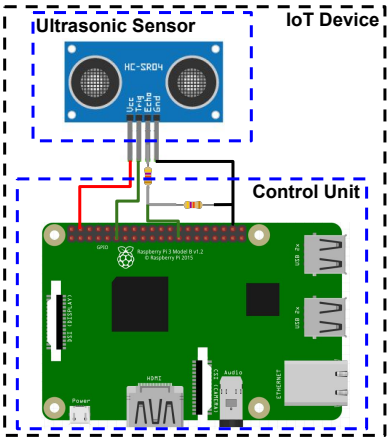


Figure 4.2: DUT: IoT parking sensor

- The control unit multiplies this time by the speed of sound (343 m/s) to determine the distance of the object from the ultrasonic sensor, and hence find out whether a parking spot is available or not.

To this end, the normal operation of our IoT device, which can be a component of a very large system, should not deviate much from the tasks explained above. Therefore, its normal behavior is expected to be reflected closely in the power consumed by the device. As we show in Fig. 4.1, we supply the IoT device with power using the Monsoon Power Monitor. Next, we explain the experiments that show what might lead to an anomalous behavior of such a device. The code used in the following testing scenarios is listed in Appendix A.1.

4.2.1 Distributed Denial of Service (DDOS) attack

In a nutshell, DDOS involves a node (e.g., IoT device) or a large number of them sending large traffic volumes to overwhelm the Internet infrastructure or servers in a harmful way (i.e., to restrict the availability of services). In our experiment, we consider two cases: (i) an IoT device as a source of the attack; and (ii) an IoT device as a victim of the attack. In the first scenario, the assumption is that an attacker takes control of the device and injects code to instruct the device to generate data/traffic to flood the network. In the second scenario, we consider the IoT device as a server providing data to a real time application. When the IoT device is attacked, it becomes unreachable and hence the system loses it. We use Low Orbit Ion Cannon (LOIC) as a tool for simulating the DDOS attack [180].

4.2.2 Cryptocurrency Mining Malware

This is basically malicious code, known as cryptojacking, that tries to hijack the device's resources so that it can perform cryptocurrency mining, such as Bitcoin, for the benefits of the hacker who installed it. In this kind of security breach, we assume that the attacker has made it past all of the security measures in place and was able to install the malware (Miner). IoT devices are easy target for this type of malware due to the increasing number of devices connected to the Internet and the increasing computational power of IoT devices. While an individual IoT device does not have enough computational power to obtain measurable profit from cryptocurrency mining by itself, cryptojacking malware can easily spread across the Internet and infect a large numbers of IoT devices. In fact, many cryptojacking programs have targeted IoT devices, specifically targeting Raspberry-Pi based IoT devices [158]. The program simulates cryptocurrency mining by computing the SHA-256 hashes of random byte sequences in order to find sequences whose hashes start with multiple zero bytes. Except for the difficulty of the problem, this is exactly the same protocol that Bitcoin uses for its proof-of-work, and is exactly how Bitcoin mining programs work. The miner is only active for part of the time, which is a strategy used by malware to avoid detection.

4.2.3 Faulty CPU

In this experiment, we simulate that an IoT device has a problems with its CPU. The Raspberry-Pi CPU has four cores, but, we deliberately deactivate some of the cores to simulate a faulty CPU. We run different experiments with different loads to stress the CPU.

4.2.4 Datasets

For each of these experiments, we collect power traces of the device to build our datasets. We use these datasets to train our anomalous behavior detection engine and evaluate its performance. Table 4.1 summaries our IoT based datasets. Dataset **D**, for example, refers to the case where an IoT device has a failing CPU (a faulty component). The number of observations in this dataset is 200 power traces, 50 of them are collected when the device is working normally (with 4 active cores) and the other 150 when the device has a faulty CPU: one core deactivated, two cores deactivated, and three cores deactivated.

Table 4.1: Description of the IoT device datasets

Dataset Name	Description	# of observations per class (Dataset size)
DDOS (**A**)	An attacking IoT device, total of 2 classes: A class of a normally operating device and a class of an attacking device.	50 (100)
DDOS (**B**)	A victim IoT device, total of 2 classes: A class of a normally operating device and a class of compromised device.	50 (100)
Crypto Miner (**C**)	Mininig Cryptocurrency using a hijacked IoT device's resources, total of 2 classes: A class of a normally operating device and a class of compromised device	50 (100)
Faulty CPU (**D**)	IoT device with a faulty CPU, total of 4 classes: A class of device with 4 working cores (normal class), and 3 classes of partially activated cores (3 cores, 2 cores, and 1 core), simulating the abnormal class.	50 (200)

4.3 Smartphone Based Experiments

The second device chosen to validate our methodology is a Smartphone. Although the resources on-board a smartphone are more powerful than most of IoT and CPS devices, some applications can be represented by such a powerful device. The examples range from intelligent surveillance cameras used in intelligent transport systems [188] to medical devices used in medical cyber physical systems [167]. The common characteristics between smartphones and these devices are the powerful compute capabilities and the high generated data/traffic.

4.3.1 Emulated malware

In this set of experiments, we have developed an emulated malware that we use to simulate different anomalous behaviors. The emulated malware in our case is a piece of code that runs in the background of the device and executes certain activities. The assumptions we make while designing these experiments are:

1. Real malwares perform network activity to either (a) only download (DL) information to the compromised device (100% DL), (b) only upload (UL) user's information to a remote server (100 % UL), or (c) upload and download information from/to the compromised device (ex. 50 % UL and 50 % DL as shown in Fig. 4.3).
2. Real malware perform some local computations such as read/write to memory/storage.
3. Real malware are not continuously active when they are installed, rather they work in bursts of activities.

By having these aspects included in the development of the emulated malware, we can cover a wide range of real malware that are out there. In spite of the malware type

(botnets, rootkits, and worms, refer to [141] for detailed description of these malwares), the activities that it performs will fall largely in the range considered above.

Figure 4.3 demonstrates the possible malware activity scenarios. We classify these scenarios into two main groups of malware, namely, non-adaptive and adaptive malware. Non-adaptive refers to a malware that is active periodically (in a cyclic fashion) as illustrated in Fig. 4.3. We introduce a tunable parameter that we call "activity Duty Cycle" (DC) which represents the degree of activeness of the malware, and it is calculated as: $DC = \frac{T_{ON}}{T_{ON}+T_{OFF}} \times 100\%$, where T_{ON} refers to the time the malware is performing some tasks and T_{OFF} refers to the time when the malware is inactive. DC is the percentage of the time a malware is active performing some tasks. Several datasets are generated; each one of them represents a family of malware that depends on their network activity (ex., 50DL50UL malware family). In contrast, adaptive malware try to eliminate any cyclic behavior from its activities by a random activation time and periods of activation, as shown in Fig. 4.3.

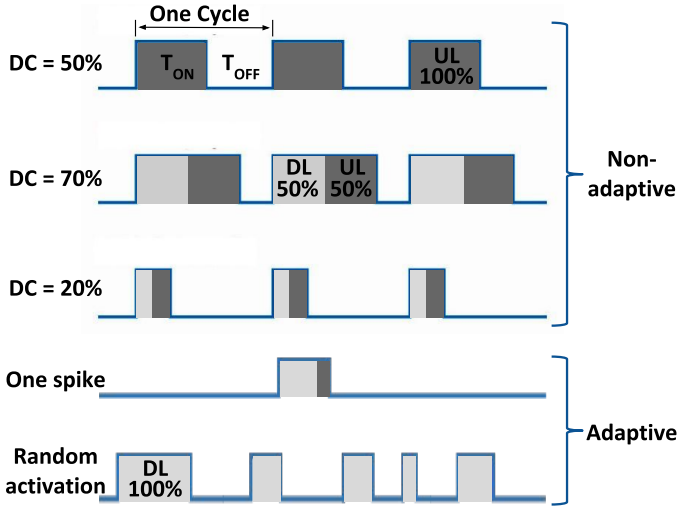


Figure 4.3: Emulated malware activity cycles/activation

4.3.2 Real Malware

In this set of experiments, we use 5 real applications (namely, Buscaminas, Tetris, Tilt, Wordsearch, and Yams) that are known as malware from the well-known malware dataset, Drebin, discussed in [26]. We downloaded their real-non-malicious versions from the Google Play store and conducted the experiments. In total, we run the benign version of the app and collect 15 signals. We repeat the same thing with the malicious version of the app and collect 15 signals as well. A summary of the collected dataset is descried in Table 4.2.

4.3.3 Datasets

In all of the *emulated malware* experiments explained above, the collected datasets comprise of two main class, namely *Normal* class and *Malicious* class. In the *Normal* class, we collect the power consumed by the smartphone while running YouTube, and no other applications are running in the background. In the *Malicious* class, the power measurements are obtained from the smartphone running YouTube and an *emulated malware* running in the background.

The choice of using the Youtube as the app for generating the normal behavior class is justified using the following scenario. A common question in the security field is: can the detection be evaded if attackers find out about the proposed detection approach? Generally, this is a difficult task since there is an inherent connection between the tasks a device performs and the power it consumes. In the side-channel attacks, the whole idea of revealing secret keys depends on the aforementioned fact. However, one way that an attacker might seek to evade detection is through the attempt to hide some of their malware's activities behind a legitimate power consumption bursts/spikes [226]. In our datasets scenarios, specifically in this emulated malware based datasets, we simulate such a strategy by having a high power activity application (YouTube) running as the normal/legitimate behavior, while the emulated malware runs in the background.

In the *non-adaptive malware* case ($E - DC$ in Table. 4.2), we use 5 different network

activity loads, namely [100% DL (download) and 0% UL (Upload)], [75% DL and 25% UL], [50% DL and 50% UL], [25% DL and 75% UL], and [0% DL and 100% UL], to represent 5 different malware families, as we show Fig. 4.3. Based on the tunable parameter -activity Duty Cycle (DC)- of emulated malware, in each malware family can we can generate different versions/instances of malware, each having a different behavior. In our experiments, we use six different DCs, namely 1%, 2%, 3%, 4%, 8%, and 12%. Each DC value represents a malware instance. To illustrate, 1% DC refers to a malware instance that is active for 0.6 seconds in every 60 seconds. And based on the malware family, e.g., [50% DL and 50% UL], 50% of the 0.6 seconds are to upload and in the other 50% to download. Referring to the examples in Fig. 4.3, malware with 20% DC means that the malware is active for 12 seconds in every minute, and in those 12 seconds the malware use half of the active time to upload and the other half to download. Thus the total number of malicious class (different malware instances) is $6 \times 5 = 30$.

Table 4.2: Description of the smartphone datasets

Name	Description	# of observations per class/ (Dataset size)
E-DC (**E**)	Emulated malware with varying duty cycles and varying network activity loads, total of 31 classes. One no-malware class and 30 different $E-DC$ malware classes.	15 ($465 = 15 \times 31$)
E-1S (**F**)	Emulated malware with one spike of activity, randomly activated, total of 2 classes: One no-malware class and one $E-1S$ malware class	15 ($30 = 15 \times 2$)
E-RA (**G**)	Emulated malware with random activation and random active periods/loads, total of 2 classes. One no-malware class and one $E-RS$ malware class	15 ($30 = 15 \times 2$)
Real (**H**)	Five malicious apps from [26] and their legitimate versions, total of 10 classes. One for malware-free version and one for malicious version	15 ($150 = 15 \times 10$)

We repeat each class's experiment 15 times, so we have 15 observations for each class, each observation (power measurement) lasting for 300 seconds (5 minutes). Given the

sampling frequency of our power monitor $F_s = 5000$ samples/second and the duration of the observation (T=300 second) each observation contains 1,500,000 samples that characterize a smartphone in either of the above mentioned classes. At the end, the total size of the *non-adaptive malware* dataset (Dataset **E**) is 465 $(15 + 30 \times 15)$ signals that make our 31 classes, as summarized in Table 4.2.

In the *adaptive malware* case, the idea is to introduce uncertainty to the behaviour of the emulated malware. We accomplish that using two setups: (i) Emulated malware with one spike (E-1S: Dataset **F**) activation refers to a malware that wakes up at a random instant to perform its activity and then go to sleep for the rest of the experiment. The only-once activated malware's activation graph is shown in Fig. 4.3. The 0.6s activity spike is generated at a random time in the 5 min interval. As explained in the previous case, we repeat this experiment 15 times while the device has the malware (E-1S) actived in the background. (ii) Randomly activated emulated malware (E-RA: Dataset **G**) is a malware that goes active at a random instant of time and performs some activity for a random period of time, then goes inactive for a random period of time. It keeps alternating between on and off based on a random timing, as demonstrated in the last graph on Fig. 4.3. Similarly, 15 power observations are collected for this kind of malware (E-RA). Table 4.2 summaries our smartphone based datasets.

Finally, in our real malware set of experiments, we run each of our 10 app (5 malicious and 5 legitimate) for 5 minutes each and collect the power consumed by the smartphone. Similar to all of our smartphone based experiments, 15 power observations are collected for each app resulting in a total of 150 power traces in this dataset (Dataset **H**).

In all of the smartphone based experiments, we automate the process of interacting with these apps using an Android tool called Droibot[1]. This minimizes the error probability and allows us to have consistent measurements.

[1]https://github.com/honeynet/droidbot

4.4 Preliminary Research: Investigating a Device's Power Signals Information for Detection Purposes

Before discussing the detection techniques proposed in Chapter 5 and Chapter 6, we conduct a preliminary analysis on the emulated malware datasets.

The following section present exploratory data analysis that helps us understand the nature of the distribution of classes across the different datasets. This analysis works as a base for our machine learning based analysis. We conducted several machine learning based analysis to justify the need to go to deep learning based detection approaches, as we propose in Chapter 5 and Chapter 6.

4.5 Exploratory data analysis

We start off by quoting John Tukey when he said "Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone" [222]. In this section, we analysis the obtained dataset statistically to better understand the nature of the data. Statistical analysis tools are used here to visualize how the classes of our datasets are distributed in 2-D space. We model each dataset using mean and standard deviation. These two characteristics are useful since in our emulated datasets case we have different scenarios and parameters. The resultant different behaviors of these scenarios are difficult to explore in 1-D format. In some cases, these behaviours are very similar. Therefore, it is a good idea to exploring the statistical properties of these datasets. Then, we show the histogram of a pair of anomalous and normal behavior of the same emulated malware.

In this analysis, we investigate whether a classic set of features, that are commonly extracted of time-series signals, convey good information about the device's health. The extracted features are:

$$mean:\ \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$Standard\ deviation\ (STD):\ s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

$$Kurtosis:\ k = N\frac{\sum_{i=1}^{N}(x_i - \bar{x})^4}{(\sum_{i=1}^{N}(x_i - \bar{x})^2)^2}$$

$$Median:\ md = \begin{cases} x_{\frac{N+1}{2}} & \text{for N odd} \\ \frac{1}{2}(x_{\frac{N}{2}} + x_{\frac{N}{2}+1} & \text{for N even} \end{cases}$$

$$Skewness:\ sk = \sqrt{N}\frac{\sum_{i=1}^{N}(x_i - \bar{x})^3}{(\sum_{i=1}^{N}(x_i - \bar{x})^2)^{3/2}}$$

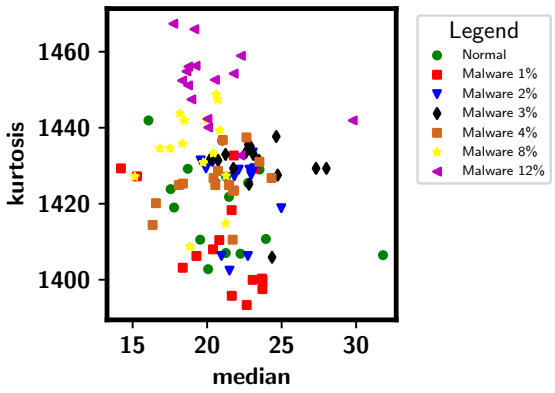
For each power signal of each malware instance we computed these features. Throughout the rest of this section, we will interchangeably call each measurement (power trace) an example or data point, x. The whole dataset is denoted with capital X. We form a Features vector that contiain the five features explain above. Therefore, each data point x is represented by five features $\{mean, sd, k, md, sk\}$.

Figure 4.4 shows the 100UL00DL malware family and how each class is distribution according to the extracted features. Each figure in Fig. 4.4 represents the six malware instances in this malware family. For example, Fig. 4.4a, which is denoted by Kurtosis vs Median, shows the distribution of the six malware instances using these two features. The 100UL00DL malware family, as explained in section 4.3.1, means that when the malware is in active state, it only uploads some data to a remote server. The scatter plot of Fig. 4.4d shows how the two features, namely, the mean and standard deviation, are used to characterize the power consumption signals of a device's behaviour while being compromised with one of the malware instances. In each of these figures, we show also the

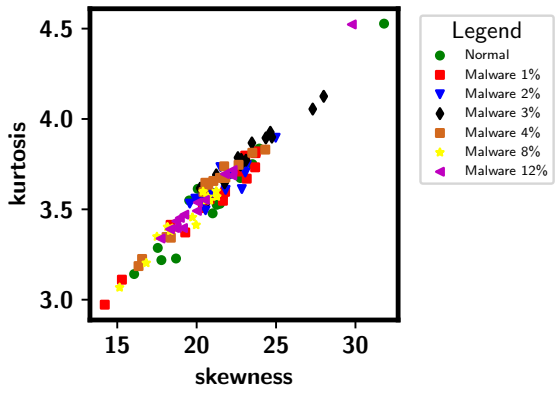
the normal behavior of the device (highlighted in green) to get a sense of how different is the normal behaviour from the malicious one.

The histogram of the raw data of the two classes (normal and malware) is shown in Fig. 4.5. In this case, the histogram illustrates the relative frequency of occurrence of power amplitudes of our raw signals. What each of the figures in Fig. 4.5 demonstrates is the histogram of the main two behaviours of our anomalous behaviour detection problem, namely, malicious and normal. As can be noticed in Fig. 4.5, the behavior of the device when the malware's activities are rare and done in short bursts of time appears very similar to the behavior of a device that is not compromised (normal behavior). The case of 1 % malware through the 4 % malware, the histogram fails to show distinct behavior (Fig. 4.5a, Fig. 4.5b, Fig. 4.5c, and Fig. 4.5d). However, in the cases where the malware is quite active (i.e., the 8 % malware and the 12 % malware), the histogram plots show that the power behavior of a normally operating device is quite different from a compromised one (Fig. 4.5e and Fig. 4.5f).

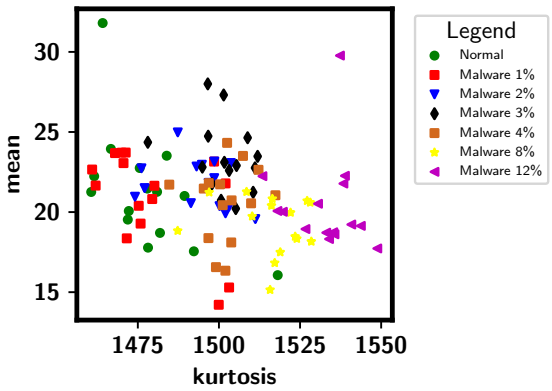
The takeaway from Fig. 4.4 and Fig. 4.5 is that the power consumption behavior of a device does convey some information about the operational state of devices. However, we also notice that the information are noisy and do not allow for an easy separation of the classes. This observation is clear in the cases where the malware instances are lightly active (Malware with duty cycle of 1% to 4%). Throughout this statistical analysis, whether it is the features based visualization or the histogram, the case made in the last sentence still holds. This is justifiable given the fact that device's power signals contain much noise and do not clearly show the information (i.e., events, their frequencies, and duration). Therefore, dealing with them in their original structure is problematic. This leads to achieving reasonably accurate detection performance can not be easily done [32], when employing traditional detection methods. To conclude this analysis, these signals have a 1-D structure and are generally complex to be described with analytical equations with parameters to solve [217]. The observations from Fig. 4.4 confirm this statement, and further, show the need for alternatives to capture better information and features to accomplish the objectives of this study.



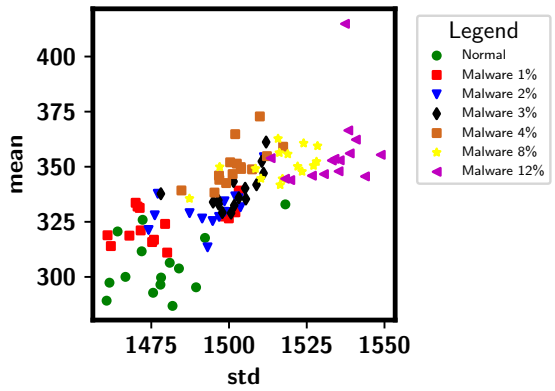
(a) Emulated Malware: Kurtosis vs Median



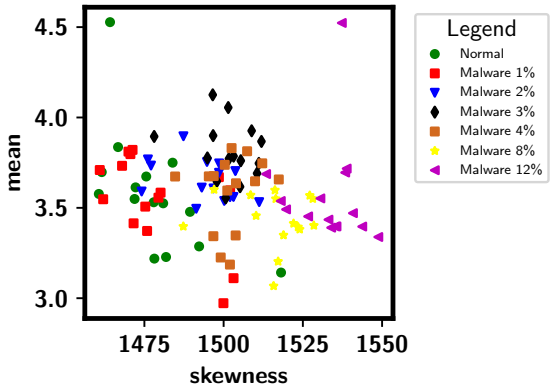
(b) Emulated Malware: Kurtosis vs Skewness



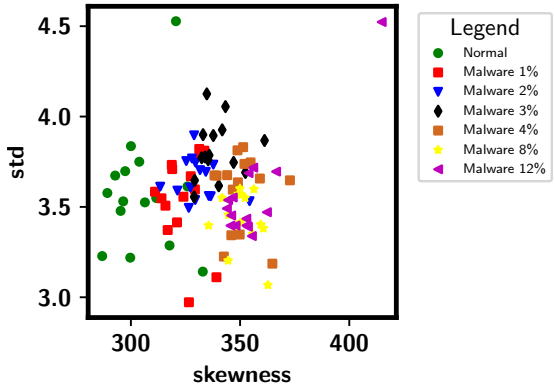
(c) Emulated Malware: Mean vs Kurtosis



(d) Emulated Malware: Mean vs STD

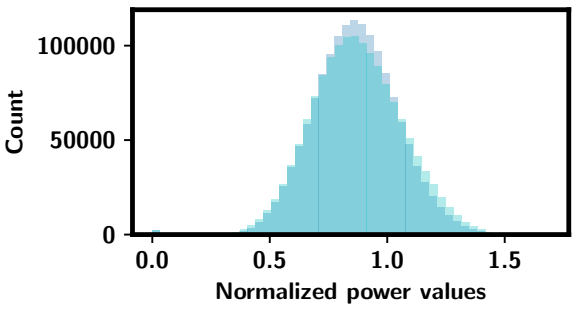


(e) Emulated Malware: Mean vs Skewness

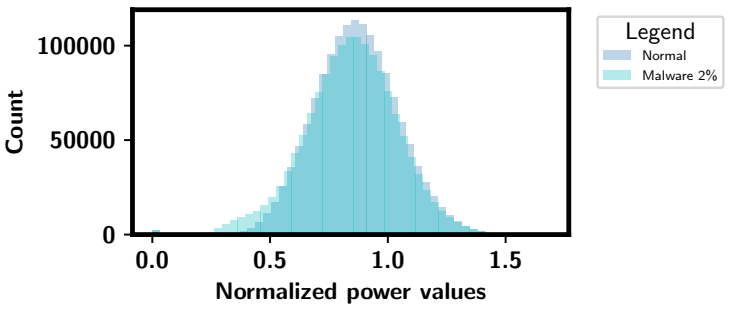


(f) Emulated Malware: STD vs Skewness

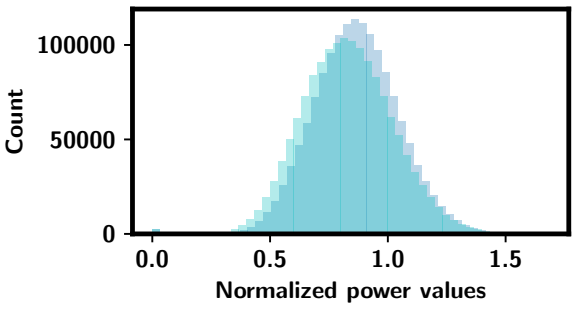
Figure 4.4: Scatter plot of the 100UL00DL emulated malware family for several set of features



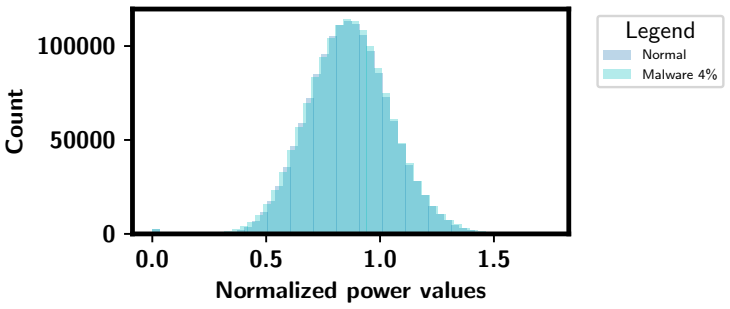
(a) Histogram of 1 % emulated malware



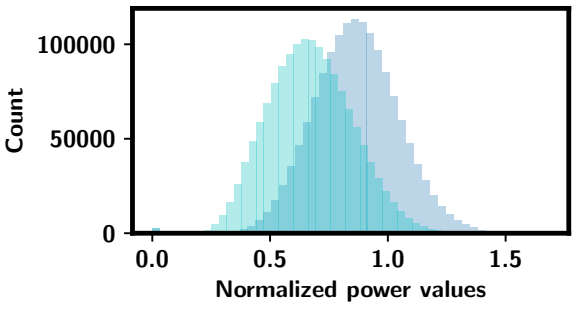
(b) Histogram of 2 % emulated malware



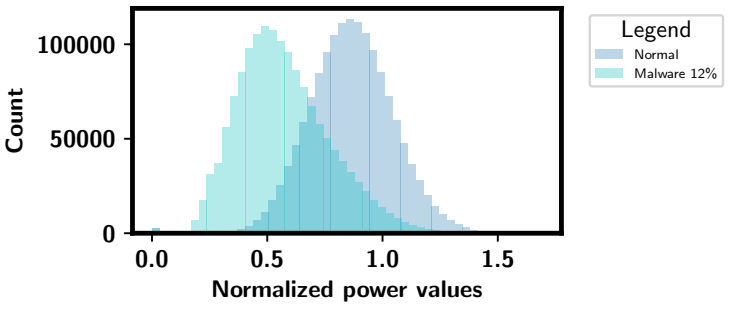
(c) Histogram of 3 % emulated malware



(d) Histogram of 4 % emulated malware



(e) Histogram of 8 % emulated malware



(f) Histogram of 12 % emulated malware

Figure 4.5: Histogram plot of power consumption of a devices in the two states: compromised and benign

4.6 Machine learning Analysis

In this section, we build on upon the discussion of the previous section. The distribution of the classes seem separable in some cases, hence, machine learning models can be explored to answer our first question: Can the power consumption of devices be used to detect different kinds of anomalous behaviors?

We start off by building a simple data pipeline model that we use for our machine learning analysis. In this model, we have two main steps, namely, (i) data pre-processing, where we use the five features explained in the previous section and then apply PCA; and (ii) data modeling, where we train an SVM model on the processed dataset.

Step 1: Data Pre-Processing

In this step, we use our dataset X. Each data point x, in X is represented by five features $\{mean, sd, k, md, sk\}$. We, then, apply PCA to reduce the dimensions and prepare the dataset for data modeling step.

Data Labeling, since we know a prior of time the label (l) of each power trace, we label each processed power trace x with the proper label, as we show in Fig. 4.6. Although we only use two principle components of each data point, the classes' distribution shows somehow distinguishable pattern, especially when the malicious code is highly active (i.e. has high Duty Cycle (DC) (Fig. 4.6).

Step 2: Data modeling

In this step, we train an SVM classifier as follows. It is important to explain how we formulate the problem. We are interested in detecting the anomalous behavior of a device due to the presence of emulated malware. Therefore it is safe to formulate the problem as a binary classification problem, $l \in \{1, 0\}$, where the first class's label is *No Malware* and the other one's label is *With Malware*. In chosen emulated malware family, we have

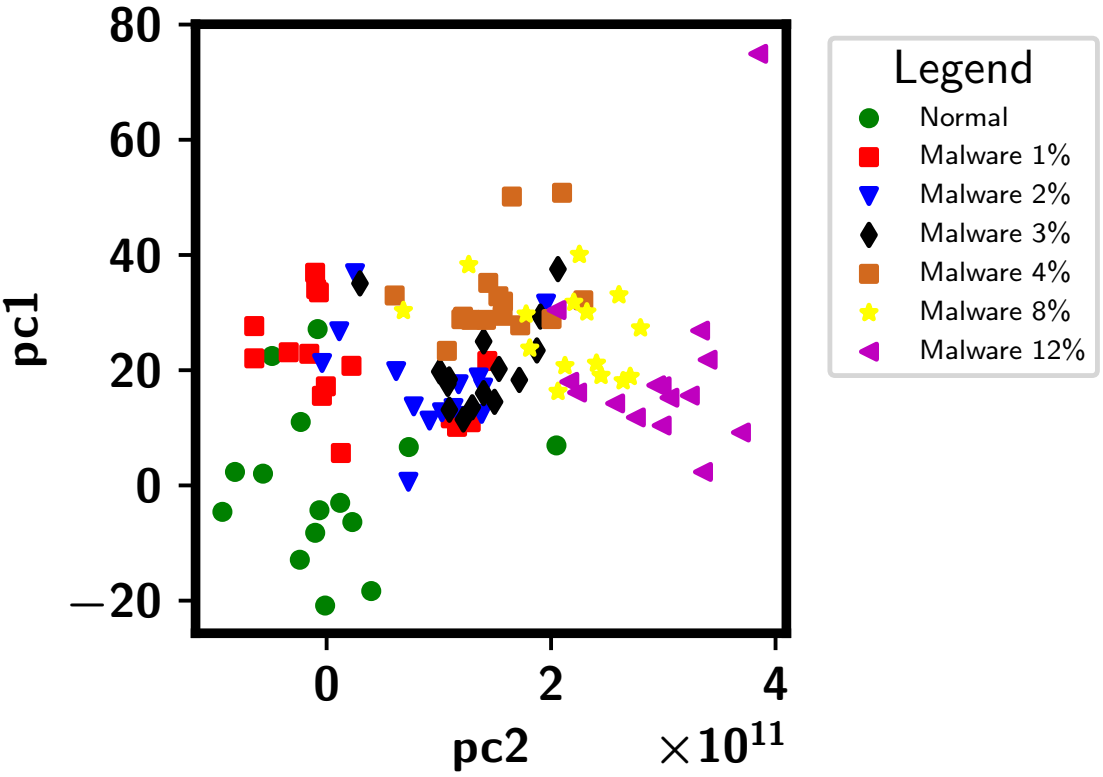


Figure 4.6: Distributions of classes using the the first two principle components

six different versions of the malware, each has different activity duty cycle. Since the first class represents a "healthy" operation (*No Malware*), we have it in all of the six classification scenarios. To elaborate, in the first scenario, we train on 10 data points that were randomly selected from *No Malware* and 10 data points from *Malware with DC = 1%* selected randomly. We fit a model for this scenario and evaluate it on the test dataset. In the second scenarios, we again train on 10 data points from *No Malware* and 10 data points from *Malware with DC = 2%*, and so on for the other duty cycles. At the end we have six models to be tested. This approach enables us to investigate the sensitivity of detecting malwares that are not active long period of time. Also, it gives us an initial sense on the feasibility of using power consumption of devices to detect their anomalous behaviour.

Given the knowledge we inferred from visualizing the classes in Figure. 4.6, we know

that detection of malware with DC less than 3% will be the most challenging scenarios. That is what we can confirm from F-score results shown in Figure. 4.7. It shows that, the more active the malware is, the higher the rate of detection is. After the DC becomes higher than 3%, we notice that the detection rate reaches 100%. This is expected from the fact that the two classes (*No Malware* and *Malware with DC > 3%*) are totally linearly separable, as can be noticed from Fig. 4.6.

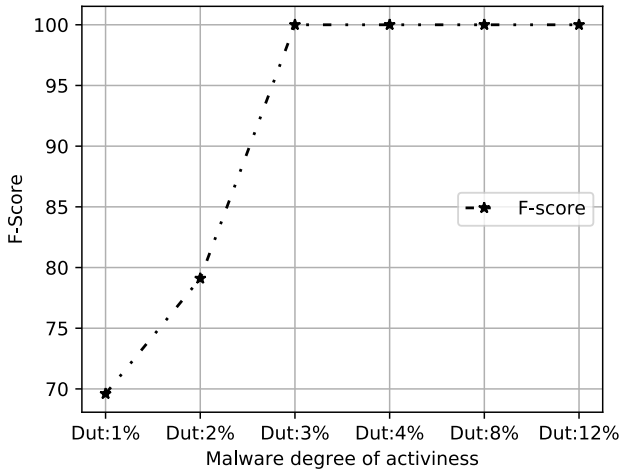


Figure 4.7: F-Measures results

A more realistic assumption is that real malware is lightly active. Therefore, unlike the emulated malware cases that cause high activity behavior, results of the malware with low activity behavior, in Fig. 4.7, give us the clue that such a model is not practical. Therefore, we went ahead and built data pipeline model that uses a class of machine learning model to investigate weather we can achieve a better performance on the low activity malware instances. As shown in Fig. 4.8, five classifiers are implemented, namely, 5-NN, 3-NN, SVM, Fandom Forest, and linear regression. To achieve a good model that generalizes well on the test dataset, we perform cross validation and search grid to fine tune the parameters and avoid the over-fitting problem.

The obtained results are presented in Table. 4.3. Unfortunately, none of the classifiers performed better than the previous case.

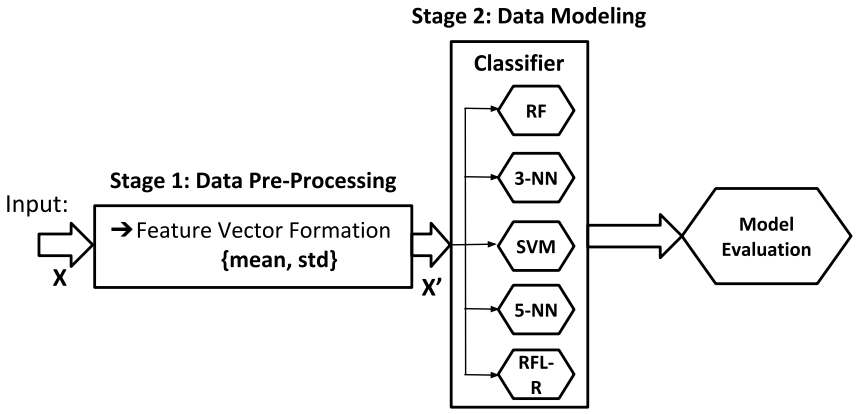


Figure 4.8: Implemented machine learning model architecture

Table 4.3: A summary of detection performance using a class of machine learning classifiers

ML algorithm	Training Accuracy	Testing Accuracy	F-Score
SVM	90%	69.9%	69.9%
3-NN	90%	69.9%	76.9%
5-NN	90%	59.9%	66.9%
RF	100%	69.9%	76.9%
L-R	90%	69.9%	72.7%

As a conclusion, the performance of the machine learning analysis, when using hand-crafted features, was not as good and suffered from high false negatives, regardless of the used machine learning models. We think that the reason behind the poor detection performance is that by using hand-crafted features, we tend to lose much discriminative information. Therefore, in the next chapters, we explore new avenues that leverage deep learning techniques to achieve better results and build more robust detection models.

4.7 Sampling Frequency impact Analysis

In this section, we utilize artificial neural network (ANN) to build a model. We utilize the same dataset described in the previous section; however, we treat the raw power consumption traces as signals that carry information about the operational state of the smartphone. The aim in this analysis is to answer the question: how do the measurements sampling rate and the window size impact the detection performance?

From the results show in Fig. 4.7, we chose the malware instance that gives use the highest detection performance. The reason is that, we guarantee that the ANN model will perform well on that classification scenario. Doing so allows us to investigate the impact of the measurements rate on the detection performance.

The implemented ANN model is illustrated in Fig. 4.9, where we start with samples preparation. Then we train ANN with 3 layers to classify the power signals.

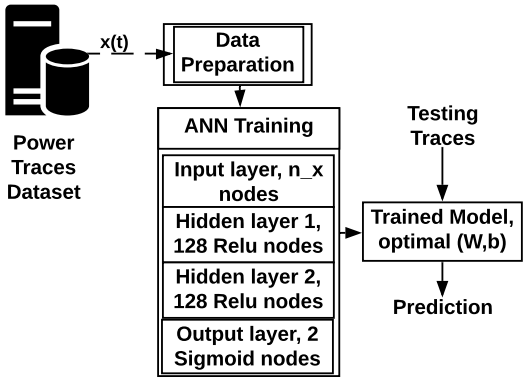


Figure 4.9: ANN model for malware detection

Data Preparation: A key aspect of this experiment is data preparation. In our dataset, each power trace $x(t)$, is a vector of size $T * f_s$ samples, as we explained in Section 4.1. We introduce the concept of the "labeling window", where the size of this window represents the number of selected samples n_x. For example, if $n_x = 5000$, this means the window has 5000 samples which is equivalent to 1 second of the experiment time. In

this case, the task involves labeling each 1 sec of each power trace with same label of the original signal. For example a power trace $x(t)$ with the label *No Malware* ($y = 0$), generates $x(t)/n_x = 1,500,000/5000 = 300$ training example $\bar{x}$ labeled with the same original label ($y = 0$). In total, in this example we get from 30 power traces in our dataset $30 * 300 = 9000$ training examples.

Model: Our ANN architecture model contains three layers. The input layer of size n_x followed by two hidden layers, each has n_h neurons. The output of the these two layers is fed to a classier layer, which is simply a fully connected (affine) softmax output layer of a size n_y (the number of classes), as illustrated in Figure. 4.9. The softmax layer produces a distribution over the 2 classes in our problem.

Figure 4.10 summaries our results and shows the accuracy for different sampling frequency rates. The accuracy is higher than 90% for all of the cases where the sampling rate is higher than 1000 samples per second. The takeaway from these results is that, the higher the sampling rate, the more is the discriminative information captured by the model. We also have tested our model on traces that was not included in the dataset, traces that the model has never been exposed to. The Accuracy gotten on those traces is 90% and the FN rate is 5%. The second observation from Fig. 4.10 is that the impact of the window size on the detection performance. We can see that the larger the window size, the higher is the accuracy. This makes sense in our case, given that the larger the widow size, the more is the captured information.

4.8 Summary

This chapter provided a detailed description of the experiments conducted to generate different anomalous behaviours of devices. The collected data is used to build a wide range of security threats that are found in the practice and academia. These datasets covered the three main aspects of a device's security (namely, confidentiality, integrity, and availability), and partial system failures. Such datasets should bridge the gap of

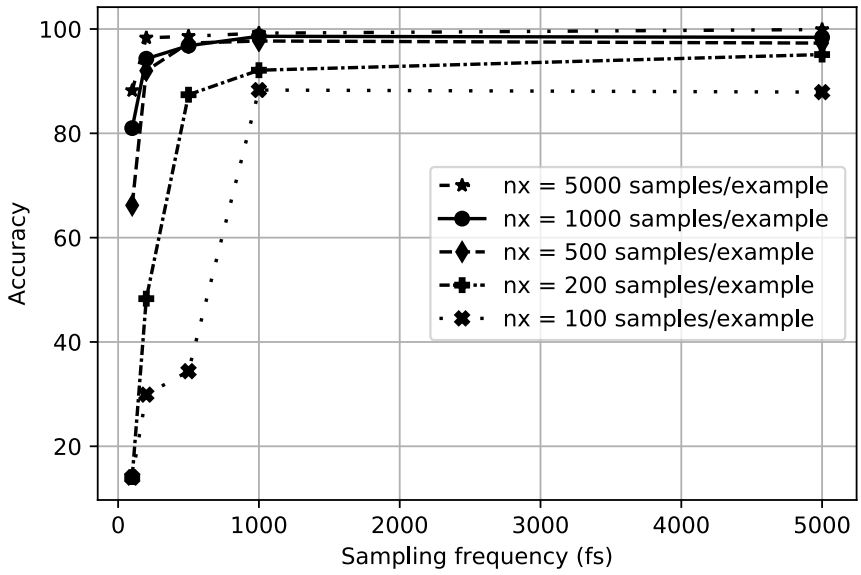


Figure 4.10: Detection accuracy vs sampling rate

having publicly accessible benchmark for the research community.

Chapter 5

Transforming the 1-D anomalous behaviour detection problem into an Image Classification Problem: A Supervised Approach

This chapter explains our supervised based detection methodology. The methodology comprises of two stages. In the first Stage- Feature Extraction -, we utilize signal processing techniques to transform 1-D power consumption signals into 2-D images. Then, in the second stage - Model Generation -, we train a convolutional neural network on the obtain images and generate a model to be used for the detection.

In light of the discussion presented in Chapter 2 and Chapter 3 and to address the issues aforementioned, in this chapter we extend the state of the art that adopts the idea of leveraging the power consumption of devices as a signal and the concept of decoupling the monitoring system and the devices to be monitored to detect and classify the "operational health" of the devices.

5.1 Problem Description

This work treats the power consumption of devices as signal that carrying information. The challenge is how to extract knowledge and informative insights out of these time-series signals. This is a challenging and yet interesting task, because time-series signals are generally complex to be described with analytical equations with parameters to solve. This is because power consumed by devices is a stochastic time-series signal. In other words, these signals are highly dynamic and exhibit nonlinear behavior [217]. Moreover, time-series signals have a 1-D structure and are non-stationary, which means that signals' characteristics (namely mean, variance, and frequency) change over time. Therefore, dealing with them in their original structure is problematic and it leads to the fact that achieving reasonably accurate detection and classification performance can not be easily done [32]. We have validated that in Chapter 4 that when employing traditional detection and classification methods. The general trend is to develop domain-specific features for each task, which is expensive, time-consuming, and requires expertise in the signals and applications. The alternative is to transform the 1-D time-series signals to another representation (e.g., 2-D image) that captures the temporal information of the time-series. Our approach in this chapter falls under the latter approach.

5.2 Solution Strategy

Unlike any of the similar reported studies [241, 123, 130, 77, 77, 231, 29], we transform the 1-D instantaneous power consumption signals of such devices to another representation (namely, 2-D images) that captures the temporal and frequency of the tasks performed on-board a device. In this chapter, the proposed solution relies on the following: (i) the fact that IoT, Industrial IoT, and CPS devices are likely to be designed to execute repetitive tasks, and the fact that every single action on-board (whether hardware or software driven actions) will be reflected as a change in the monitored "side channel information" - the power consumption of a device; and (ii) the hypothesis that time-frequency representation

of a signal is capable of preserving much of the information embedded in 1-D signals [90]. Then by applying Histograms of Oriented Gradients (HOG) on the CQT images, we extract more robust features that preserve the edges of time-frequency structures (i.e., description of local information and how these structures/events evolve with time) and also the directionality of the edge information (i.e., how these structures/events evolve with time)

The detection methodology comprises of two phases, namely, Features Extraction and Model Generation. In the first phase, we use Constant Q spectral Transformation (CQT) [49] to transform the power signals into 2-D TFR (Time Frequency Representation) images. CQT was chosen over short-time Fourier transform (STFT) or other TFR techniques (e.g. Mel Frecuency Cepstral Coefficients (MFCC)) because it adopts a logarithmic frequency scale, which allows for a better and clearer time and frequency resolutions. The CQT images capture valuable information - events' location in time and frequency domains, frequency of events, and shapes and duration of events - about the operational states of the device from its power consumption signals. Then, we apply HOG [71] on the CQT images to build a higher-level features and extract consistent information about time-frequency evolution of the shapes and structures (i.e., description of local information and how these structures/events evolve with time) of the CQT images. The natural expectation is that the HOG images capture discriminative information that is not captured by hand crafted features obtained from raw 1-D signals (e.g, mean and standard deviation). From this point, we transform the device's anomalous behavior detection problem into an image classification problem. Consequently, we train a convolutional neural network (CNN) model on these images for the purpose of classification. We argue that by treating the generated HOG features as 2-D images that contain information about the device's health, detecting when the device behaves anomalously due to a malware invasion or even a normal device performance degradation is something inevitable.

The above facilitates the following advantages: (i) the proposed solution is a lightweight one which makes it suitable for resource-constrained devices; (ii) the usage of a device's power signals has the potential to detect any anomalous behavior, even the ones caused by

malwares, for example, that can obfuscate and modify its code. Such malwares usually look as benign and go undetected when using approaches that are based on static analysis [226]. In our case; however, any change or modification will inevitably leave a trace (fingerprint) in the power consumption of the device and hence can be detected; (iii) In contrast to all on-device approaches [134, 77, 77, 231, 29], our solution ensures that the integrity of the collected data from a compromised device is preserved.

The proposed solution includes two main phases. The first phase is calibration (training) phase, where we perform two steps: (1) Data collection step, in which a reference device is monitored for certain amount of time to collect and build a power signal database (dataset). This step has been address in Chpater 4; (2) Data analytic step, where the data is being analyzed in our anomalous behavior detection engine. In this step, we transform the raw data to extract features and train a classifier on them. The second phase is detection (testing) phase. Once a model is learned, given a test 1-D power signal from a device, the anomalous behavior detection and classification engine gives an answer as to whether the device is behaving anomalously.

Validation Scenarios. In order to evaluate the effectiveness of the proposed solution, we make use of 18 dataset discussed in Chapter 4. These datasets cover a range of anomalous behaviors of devices (smartphones and generic embedded devices (IoT)).

- For security threats, we implemented a large class of experiments that covers the three main aspects of device's security, namely, confidentiality, integrity, and availability. We developed an emulated malware that can be tuned to represent different malware behaviors. By considering a tunable emulated malware, the idea is to cover a wide range of malware behaviours so that our methodology can perform well against real malwares as well. Moreover, we have tested this argument using 5 real malwares (taken from the well-known Drebin dataset [26]) and found out that our approach generalizes well on them too. We have even evaluated our system on a malware that performs Cryptomining which is basically a malware that takes over a device's resources and use them for cryptocurrency mining without the user's explicit permission [213]. We have also implemented security attacks that are widely encountered in

IoT and CPS devices, namely, Distributed Denial of Service (DDoS), where we used the device as a victim of DDOS and as an attacker (a source of an DDOS attack).

- For faulty device, we have designed an experiment to simulate an anomalous behavior of a device due to a faulty CPU.

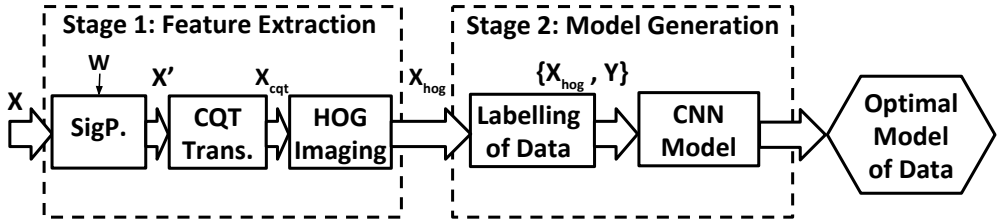
Across the 18 evaluation scenarios, we have achieved a high detection performance of $\sim$ 88% accuracy and 85% F-Score, where in some cases, the accuracy of our work outperformed previously reported detection approaches with $\sim$9% to 20% detection performance gain.

Contributions. In this chapter, we make the following contributions: we propose a novel unsupervised feature extraction technique which uses TFR images (namely CQT) and HOGs to build 2-D images out of 1-D power consumption signals. To the best of our knowledge, this is the first work that adopts such a concept to train a CNN classifier to detect and classify anomalous behavior of devices. We validate the accuracy, F1-measure, and other performance metrics of the proposed methodology using a large class of power datasets. This level of validation is quite extensive and have never been reported in any other study found in the literature.

5.3 Methodology

This section presents the details of the proposed methodology. Before explaining the model block by block, we explain the motivation behind this methodology.

This methodology is inspired by the computer vision domain [71]. The idea in the methodology is to transform the problem of device's anomalous behavior detection into an image classification problem. This is achieved by utilizing time-frequency representation (TFR) of signals to produce informative visual textures. The assumption is that TFRs (2-D images) construct textures that capture features about the events/tasks carried out from a 1-D power signals. Furthermore, Histogram of Oriented Gradient (HOG) is computed for



Legend:

SigP. - Signal Partitioning, **W** - window size, **CQT** - Constant Q Transformation

HOG - Histogram of Oriented Gradient, **CNN** - Convolutional Neural Network

X - (m*N) matrix of power signals **X'** - (M*W) matrix of partitioned power signals

$\mathbf{X}_{cqt}$ - (M*(s,s)) tensor of the CQT images, $\mathbf{X}_{hog}$ - (M*(s,s)) tensor of the extracted HOG images

Figure 5.1: Overview of the methodology: Signals Transformation and Model Training

the TFR images to form higher-level features. The HOG information is treated as images that contain information better than the features of CQT images. Finally, a CNN model is trained to classify these signals.

The basic concept of our anomalous behavior detection engine is illustrated in the methodology shown in Figure 5.1 and Figure 5.2. Figure 5.1 depicts the two main stages that make up the framework, namely *Feature Extraction* stage and *Model Generation* stage. We start with a brief explanation of these stages and the intuition behind them, and then in section 5.4 and section 5.5 we describe the methodology in more detail.

Stage 1: Feature Extraction

Feature extraction is the process of transforming the raw input data into another space that possesses better discriminative information. Since we are dealing with time-series signals that represent the behavior of wireless devices, having this step is necessary because such signals are generally complex to be described with analytical equations with parameters to solve due to their high stochastic nature and nonlinear behavior [217][43]. The resultant features from this stage are transformed versions of the input signals (e.g., TFR images).

Stage 2: Model Generation

In this stage, we train a model on the prepared dataset (X_{hog} in Fig. 5.1) to detect when

a device is behaving anomalously. To achieve this objective, our methodology makes use of deep learning techniques to generate an optimal model that can be used to accurately detect a device's anomalous behavior. We use stratified sampling to split the dataset into training, cross validation, and testing datasets. We train the model on the training dataset portion and then evaluate the model on the cross validation portion. We perform hyper-parameters tuning, and once we reach a certain performance accuracy on the validation dataset portion, we stop. The *output* of this stage is the learned optimal model of data to be used for the actual detection of anomalous behavior. More details of these stages are discussed in Subsection 5.4 and Subsection 5.5.

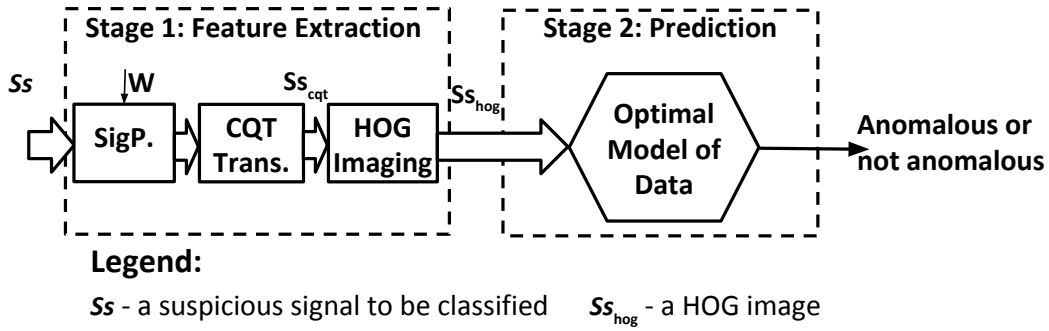


Figure 5.2: Detection of Anomalous Behavior: model testing

Detection of Anomalous behavior

The detection of anomalous behavior in real-life scenarios makes use of the model learned in the steps explained above. Once the model is learned, the operation of the methodology is depicted in Figure 5.2. Given an unknown power trace Ss from a wireless device, the methodology starts by extracting a feature vector (Ss_{hog}). Then, by feeding (Ss_{hog}) to the learnt model, the methodology gives an answer as to whether the device is behaving anomalously.

5.4 Feature Extraction: From 1-D signals to 2-D images

1-D signals are generally complex to be described with analytical equations and parameters to solve, since they are highly dynamic (nonlinear behavior) [217]. For instance, mean and variance are widely used to characterize signals [43]. However, given the non-stationary nature of power consumption signals, reliance on these kinds of features to classify them is not very promising, as demonstrated in [122]. An example of how power consumption signals of wireless devices look like is shown in Fig. 5.3-(a). It can be noticed that time-based measurements contain much noise and do not clearly show what frequencies (events) are present. Therefore, dealing with them in their original structure is problematic and achieving reasonably accurate detection and classification performance can not be easily done [32], when employing traditional detection and classification methods. Alternatively, our objective of using time-frequency representation (TFR) in the proposed feature extraction technique is to extract as much information as possible from a signal using both domains (time and frequency) at once.

Before we explain how the transformation of the time-series signals to 2-D images is done, we describe the necessary notations as follows: X is a matrix representing the raw dataset, which in our case is a set of power consumption traces measured from a device. The formal definition of X is as follows: $X = \{x[n]^1, x[n]^2, ..., x[n]^i, ..., x[n]^m\}$, $1 \leq n \leq W$. Hence X is a matrix of size $m \times N$, where m represents the number of observations in our dataset and N represents the length of each power trace, $x[n]^i$. Moreover, $x[n]^i$ is a vector (time-series: a discrete signal) of size $1 \times N$, and the elements in $x[n]^i$ represent the values of the power consumed by a device for T seconds and sampled at a rate of F_s samples/sec. Therefore, $N = T \times F_s$. Finally, in our dataset, $m = \rho + \alpha$, where ρ refers to the number of measurements from a normally behaving device and α refers to the number of measurements from an *anomalously* behaving device.

5.4.1 Signals Partitioning

We split each signal in our dataset, X, into k equal length sub-signals. The number of these sub-signals (k) is determined based on the window size W and N. After completing this step, the dimensions of our dataset X' will be $((M \times W))$, where $M = mk$. $X' = \{x'[n]^1, x'[n]^2, .., x'[n]^i, .., x'[n]^M\}$, $1 \leq n \leq W$. This means X' has (M) sub-signals, each of length W samples. If $W = N$, i.e., no signal partitioning was performed, then $k = 1$ and $M = m$.

5.4.2 Constant Q Transformation (CQT)

In this step, we start by applying CQT transformation to each sub-signal. CQT is basically a time-frequency analysis technique which was introduced in [49]. It is based on the idea of spacing the frequency components (bins) geometrically (i.e., exponentially or at a logarithmic scale), unlike any of the other TFR techniques (ex., Fast Fourier Transform (FFT)). The frequency components are spaced according to: $f_c = 2^{\frac{c}{A}} f_0$, where c is the frequency resolution and represents the number of frequency bins (components) in the CQT, $c \in \{1, 2, 3, ...c_{max}\}$. We treat c as a hyper-parameter that can be tuned to achieve better detection performance. A is a constant used to determine how to set the spaces between the different frequency bins, and f_0 is the minimum frequency of the CQT. In CQT, the window size (i.e., the length of each bin) of the c^{th} frequency bin denoted by $\wp_n[c]$ is variable and computed as: $\wp_n[c] = \frac{F_s}{f_c} \times Q$. $\wp_n[c]$ is inversely proportional to the frequency f_c so that the ratio ("quality (Q) factor") is kept constant, where F_s is the sampling rate of the signal. A Hamming window [99] function $\Gamma_w[c, n]$ is applied with the same lenght as $\wp_n[c]$, as prescribed by [49]. The bandwidth of each frequency component: $\delta_c = \frac{f_c}{Q}$, and $Q = \frac{f_c}{\delta_c} = \frac{1}{(2^{\frac{1}{B}} - 1)}$, which is a constant value. These are the main equations that differentiate CQT from other TFR approaches. The reader may refer to [49] for more details on computing CQT.

Now, given a power trace $x'[n]^i$, the CQT transformation coefficients ($x_{cqt}[n, c]$), as

explained in [49], are obtained as follows:

$$x_{cqt}[n,c] = \frac{1}{\wp_n[c]}. \sum_{n=0}^{\wp_n[c]-1} \Gamma_w[c,n].x'[n].e^{\frac{-j2\pi Qn}{\wp_n[c]}}$$

The reason behind using CQT transformation instead of other TFR methods in our methodology is that CQT provides a time-frequency analysis on a logarithmic scale (i.e., it provides higher frequency resolution for lower frequencies and higher time resolution for higher frequencies). Performing that provides the ability to capture much of the information from high/low frequency components. This is not possible when using other TFRs such as STFT - (Short Time Fourier Transform). Moreover, CQT preserves a better time-frequency structure, which is something desirable for better results in our methodology [49]. While reviewing the TFR options, we found some studies, from the signal and image processing domain, that showed that CQT consistently outperformed other traditional TFRs such as Mel-frequency cepstral coefficients (MFCCs) [117] and STFT [200].

In our methodology, shown in Figure 5.1, the input to the CQT box is X', which contains several 1-D sub-signals. The output of the CQT box is a 2-D tensor X_{cqt}, which contains M cqt images, x^i_{cqt}, $i \in 1, 2, 3, \ldots M$, each of size $(n \times c)$. Finally, we resize of each of the M CQT images to a fixed size image $(s \times s)$ pixels. We consider this as another hyper-parameter and investigate its impact on the detection performance. Our dataset after this step is X_{cqt} and has the dimensions of $(M * (s \times s))$. This means that we have M cqt images, each of size $(s \times s)$ pixels.

5.4.3 Histogram of Oriented Gradients (HOG)

In this step we apply the Histogram of Oriented Gradients (HOG) to each of the CQT images obtained from the previous step, as depicted in Fig. 5.3. The concept of HOG was first introduced in reference [71] to detect humans and objects in images. Its power lies in the fact that HOG representation of an image makes a classifier's generalization robust, when, for example, detecting an object in an image, even if that object is viewed

under different conditions. The idea of using HOG to build images of time-series signals emerged due to HOG's ability to capture local shape information; consequently, HOG representation is spatial invariance [71]

Algorithm 1: HOG computation

Input : Dataset - CQT images X_{cqt}:
x^i_{cqt}, i^{th} *cqt image*, $i \in \{1, ..., m\}$
/* x^i_{cqt} a CQT image of $(s \times s)$ pixels */

Output: Dataset - HOG images: X_{hog}

Parameter Initialization $\{C, O, B\}$;

/* C: CellSize, $(p \times p)$ pixels | O: # of orientations | B: BlockSize, $(q \times q)$ cells */

for *each input* x^i_{cqt} *in* X_{cqt} **do**

  /* Given x^i_{cqt}, use the following steps to obtain x^i_{hog} */

  - Perform global image normalization of x^i_{cqt}
  - Divide normalized image into non-overlapping cells
  - Compute $\nabla(x, y)$ in x and y directions for each image pixel
  - Compute gradients histograms based O
  - Normalize the histograms across the blocks B
  $X_{hog} \leftarrow append(x^i_{hog})$
  Return X_{hog}

end

The idea of HOG in a nutshell is to locally analyze the direction of a TFR's energy variation. The way we compute HOG is based on reference [71] implementation. Algorithm 1 describes the implementation of the HOG images transformation of our dataset. A visualization example is illustrated in Fig. 5.3 to show the process for one signal.

As shown in Fig. 5.3, given a 2-D array (CQT image: x^i_{cqt}), we; (i) perform global image normalization; (ii) divide the image into non-overlapping cells, each of size $s = p \times p$, pixels per cell. (iii) compute the first order block gradients ($\nabla(x, y)$) in x and y directions; this captures the contour and texture information of that cell, while providing more resistance to noise and spatial variations; (iv) compute gradients histograms based on a number of orientations (a parameter called O, in our methodology). This is basically putting the

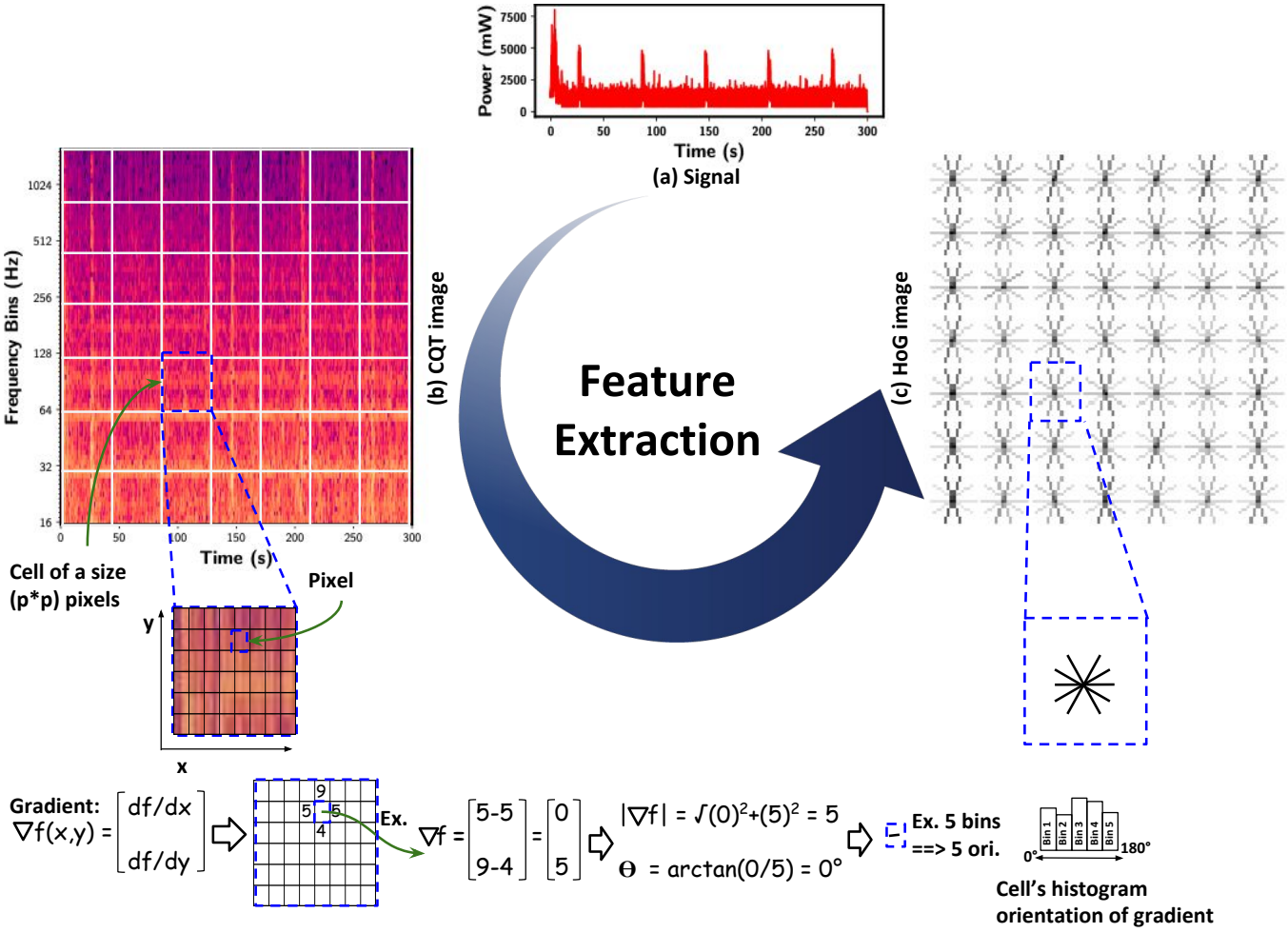


Figure 5.3: Overview of how HOG images are computed out of CQT images

gradient direction of each pixel in a bin. The number of bins (number of orientations: O) needed to calculate the histogram is considered as hyper-parameter that requires tuning. Finally, we normalize across the surrounding blocks. The block size B is also another parameter, $B = (q \times q)$ cells. This aims to produce an encoding that is sensitive to local image content while remaining resistant to small changes across images. The main goal of this step is to capture the shape of some time-frequency structures that are change resilient (in time or frequency) and are very relevant for characterizing time-series signals, in comparison to CQT time-frequency structures, for example.

In our methodology shown in Fig. 5.1, the inputs to the HOG box are CQT images (X_{cqt}) and the outputs are transformed versions of those images (e.g. shown in Fig. 5.3). Our dataset after this step is X_{hog} and has the dimensions of $(M * (s \times s))$. This means that we have M HOG images, each of size $(s \times s)$ pixels.

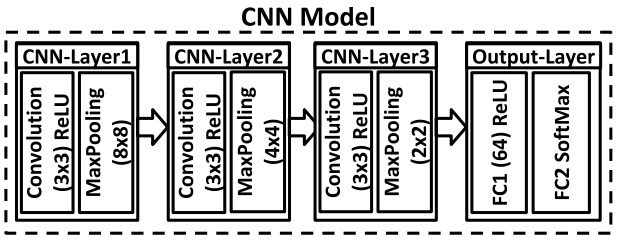


Figure 5.4: CNN model architecture: 3 layers: CNN model box in Fig. 5.1

5.5 Model Generation - Building Deep Learning Model

In this section we explain the deep learning model that we build as our classifier. Since Convolutional Neural Networks (CNN) are currently the state-of-the-art in image processing, and given that our prepared dataset comprises of images, we chose CNN to perform the classification and provide the decision. CNNs were originally presented in reference [144] and since then they have been widely used in image classification due to their unique characteristics. These characteristics include the fully connected layers, and the adoption

of max-pooling layers to down-sample an image to reduce their dimensionality. Consequently, a significant reduction in the computational cost can be achieved [144]. The idea is to train a CNN model that learns patterns from the HOG image dataset. The choice of CNN is due to the fact that it is good at learning spatially local correlations from input images. Therefore, we can leverage this to learn spatial and temporal information in HOG images in our case.

Generally, CNNs have three main layers: input layer, one or more hidden layer(s), and output layer. CNNs take matrices (tensors) of images as an input to process them, as shown in our model in Fig. 5.4. In our case, the input is the generated HOG arrays. These arrays can be thought of as images whose pixels contain HOG information. These images are arranged as a tensor (X_{hog}) of size ($M * (s \times s)$).

Next, the hidden layers perform three main processes: convolution, non-linearity, and max-pooling.

1. Convolution: The input to this layer is convolved with the layer's kernel matrix.

$$\chi^i = x^i_{hog} \circledast W_k + b, \qquad \forall\ i\ \in M$$

   where the $\circledast$ sign represents the convolution operation, x^i_{hog} is an input array (HOG image), W_k the used kernel filter, and b is a bias. These filters are represented as a sliding window of a certain size, call it W_k, which is a hyper-parameter to be tuned in the training phase. Each of these kernel functions has weights and bias that are learned during the training of the CNN.

2. Non-Linearity: The output of the convolution is fed to a non-liner function, called activation function. The non-linear function in our model is a ReLU (Rectified Linear Unit) $\phi(\chi^i) = max(0, \chi^i)$.

3. Max-Pooling: a window is slid over the output of the non-linear process and the max value in that window is taken at each step. This process is used to reduce the spatial

size of the feature maps obtained from that layer. It can be thought of as a down-sampling step. The aim of these convolution layers is to extract local information from the input image and learn weights with non-linear activation. As illustrated in Fig. 5.4, our CNN has three convolutional layers.

The output layer makes up our classifier and is made up of two fully connected layers. The first is followed by ReLU, and the second is followed by a Softmax layer made of D nodes (D is the number of classes). This layer produces a distribution over the D classes in our classification problem. To perform the training, a loss function is needed to learn the model's weights and parameters. We use the common loss function used in classification problems, which is the cross-entropy (X-E), and it is define as follows:

$$J_{X-E}(\Theta; X_{hog}, y)) = \frac{1}{M}\sum_{i=1}^{M}(y^{(i)} - log\{net[\Theta; x_{hog}^{(i)}]\})$$

where $net[\Theta; x_{hog}^{(i)}]$ represents the prediction of the network of that input instance, Θ represents all the network weights, and X_{hog}, y, M represent our dataset, the corresponding labels (classes), and the total number of observations in our dataset, respectively. The goal is to have a low distance for a correct class but a high distance (cost) for an incorrect one. In other words, the objective is to minimize the error between the predicted results by the model and the ground truth known ahead of time. The used optimizer in our case is ADAM, which is a stochastic optimization technique [136]. The ADAM optimizer works on mini-batch sizes of observations and updates the parameters (Θ) accordingly after each epoch.

5.6 Classification of Power Signals

To put the pieces of the methodology together, shown in Fig. 5.1, we use the following notations and description. Given a training dataset X that contains m signals, each signal x is a discrete signal represented by a 1-D vector of length N (an example is shown in Fig.

5.3-(a), we split each x into k sub-signals (x'), each of length W. We transform each x' into a CQT image (x_{cqt}), as explained in Section. 5.4 and shown in Fig. 5.3-(b). These CQT images are then transformed to HOG images. x^i_{hog} is an $(s \times s)$ array representing a HOG imag (an illustrative example is shown in Fig. 5.3 - (c). The dataset is finally rearranged as a tensor X_{hog}, and is ready to be used to train our CNN classifier.

Following that, we label each sub-signal/image with the corresponding label ($y_i \in Y = \{1, \ldots, D\}$). At this point, the entire dataset can be defined as $\{x^i_{hog}, y_i\}_{i=1,2,\ldots,M}$. For the training and testing, we split the dataset into 2/3 for training, 1/6 for validation, and 1/6 for testing. This will allow us to monitor the training performance so that we make sure that we are not falling into the problem of over-fitting or under-fitting the dataset.

The anomalous behavior detection procedure is illustrated in Fig. 5.2. The procedure is designed to give an answer for an unlabeled new measurement. Once a CNN model is learned, the operation of the methodology is as follows: given a suspicious power signal Ss, the methodology starts with transforming it to a HOG image Ss_{hog}. Then by feeding the HOG image Ss_{hog} to the learnt CNN model, the methodology gives an answer (a label/decision) as to which class this signal belongs to ($\{1, \ldots, D\}$), i.e., healthy device or compromised device ($D = 2$), for example.

5.7 Results

Machine Learning (ML) Experiments: The problem of detecting anomalous behavior in devices is a binary classification problem ($Y = \{0, 1\} \Rightarrow D = 2$). That is, the device is behaving normally or not. Consequently, given the datasets that we have, we perform several binary classifications to validate our method. In all of the collected datasets explained above, the *Normal* class (normally behaving device) is tested against one of the malicious classes.

Implementation: All of the experiments were performed on a powerful gaming machine (namely Acer Predator laptop) that is equipped with an Intel Core i7 processor, 32 GB of

RAM, and NVIDA GPU - GeForce GTX 960. The deep learning tasks were carried out in Python using Keras [63] and SciKit-learn [176] Python libraries with Google Tensorflow framework as a backend.

5.7.1 Classification results analysis

We break our results down into 4 main parts: (i) *Training/Testing performance*, where we demonstrate the loss/error rates of the CNN architectures over different training epochs and also show the impact of hyper parameters tuning; (ii) *Effectiveness of HOG Features*, where we analyze HOG image parameters and investigate how effective they are as compared to CQT images; (iii) *Detection Coverage*, here we report the methodology's detection rates (e.x., accuracy, F-measure, and recall) on the different datasets that we have collected; and finally (iv) *Comparison*, where we compare the performance of our model with some of the similar studies in the field.

Training Performance: A common problem when training a deep learning model with a relatively small dataset is over-fitting the dataset. For that, a couple of solutions are explored: (i) We use cross validation to monitor the training performance and evaluate our model [228]. In order to achieve that and to tune our hyper-parameters, we randomly split our dataset into 2/3 for training, 1/6 for validation, and 1/6 for testing. (ii) We apply dropout - a regularization technique - to randomly shut some of the units in each of our CNN layers [62]. In this ML experiment, we use dataset **E** - (refer to dataset E-DC in Table. 4.2). Figure 5.5 shows the training/validation loss and accuracy curves during the training phase. We can clearly notice that our CNN model is overfitting the dataset. Referring to Fig. 5.5, this can be understood from the loss behavior of the CNN model where the model learns its parameters according to the training dataset portion (a very low loss (red curve) and a high accuracy (green curve)). However, the learnt model fails to generalize on the validation dataset portion (a high loss (blue curve) and a low accuracy (yellow curve)). To mitigate this issue, we apply dropout after each layer with the following probabilities [0.3, 0.2, 0.1].

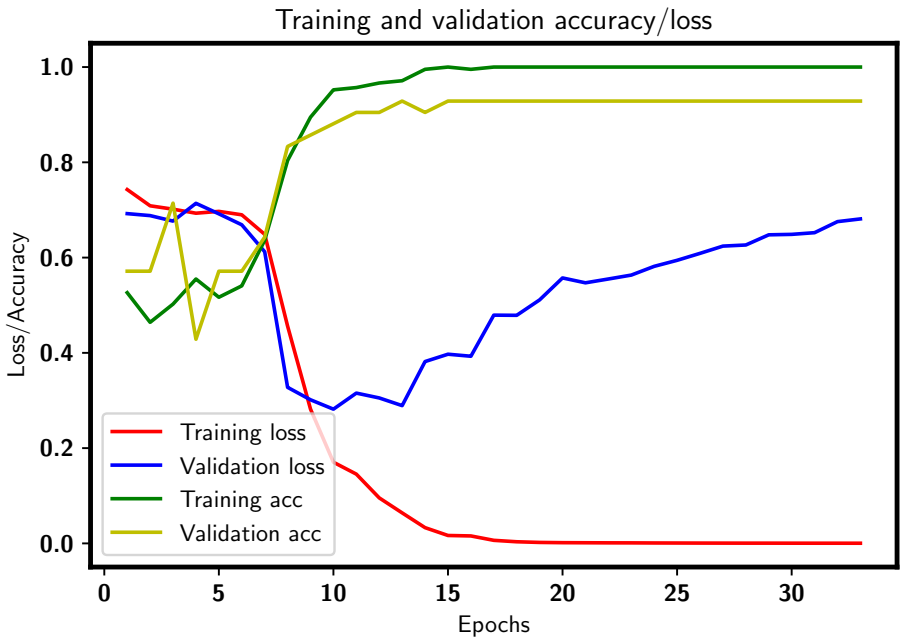


Figure 5.5: CNN over-fitting performance: without dropout

Figure 5.6 shows that the model stuck in a local minima till epoch 20 then started to converge. It also demonstrates that the signs of overfitting disappeared and our validation loss closely tracks the training loss. This confirms that our model is not over-fitting the dataset.

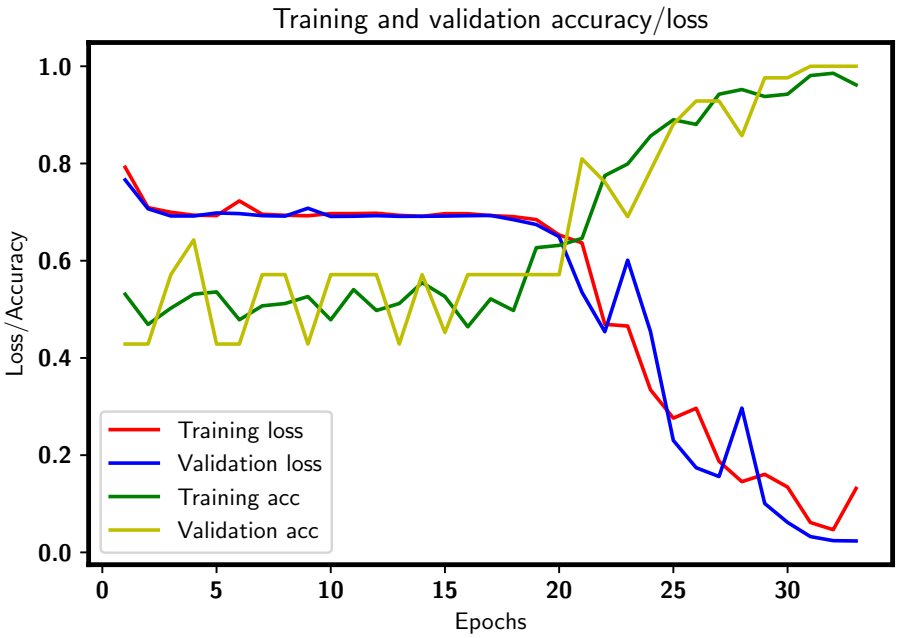


Figure 5.6: CNN generalization performance: with dropout

Figure 5.7 illustrates the impact of the window size (W) on the accuracy of the model as the malware intensity varies ($DC = i\%$). Accuracy is chosen in this case since we have balanced classes, as explained in Table 4.1 and 4.2. The window size (W) as mentioned in earlier sections, refers to the length of the discrete time signals $x[n]$. To verify that, we divide each power signal trace in our dataset $\mathbf{E}$ into non-overlapping sub-signals, as explained in subsection 5.4.1. The resultant number of sub-signals depends on the window size (W). As can be noticed in Fig. 5.7, the larger the window size (W) is, the higher the accuracy we get. This is because the larger the window size (W), the higher is the amount of information a signal contains.

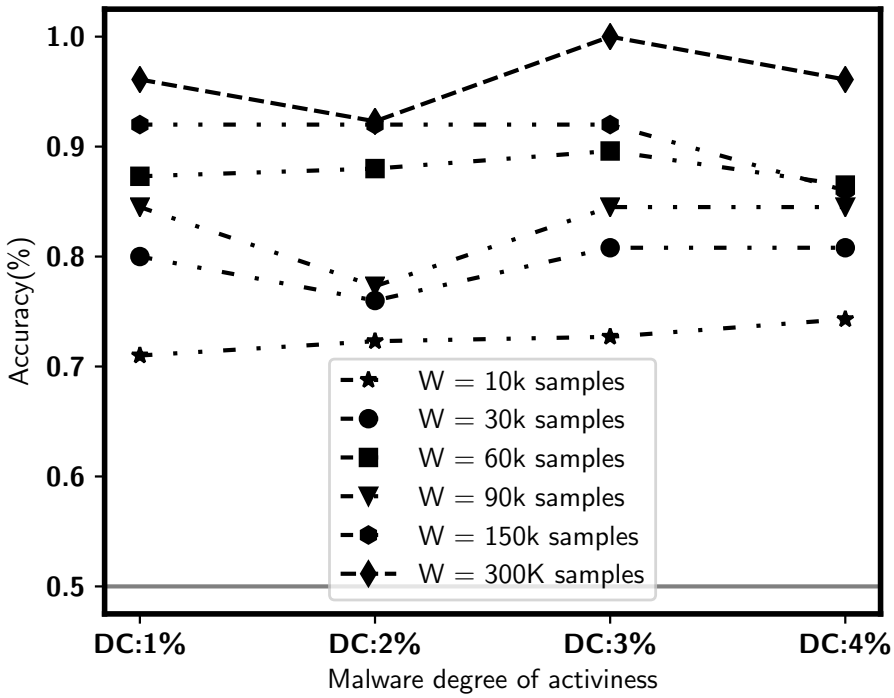


Figure 5.7: Window size impact

5.7.2 Results and analysis on the effectiveness of HOG features

In this part of the results, we start off by analyzing the impact of the HOG's parameters. Figures 5.8, 5.9, 5.10, 5.11 present the results obtained by varying some of the parameters, namely the number of bins used in obtaining the histogram of gradients (# of orientations O), the cell size: $C = (p \times p)$ pixels, and block size: $B = (q \times q)$, and keeping the image size $(s \times s)$ fixed. We fixed the image size at $(512, 512)$, and varied the other parameters to investigate their significance on the detection performance.

As the results in Fig. 5.8, 5.9, 5.10, 5.11 demonstrate, the parameters clearly impact the overall accuracy of the learnt CNN model. The most influencing parameter appears to be the orientation, as the results show. The higher the number of bins is, the better is the accuracy. Also the cell size has a strong impact on the robustness of the results, where we

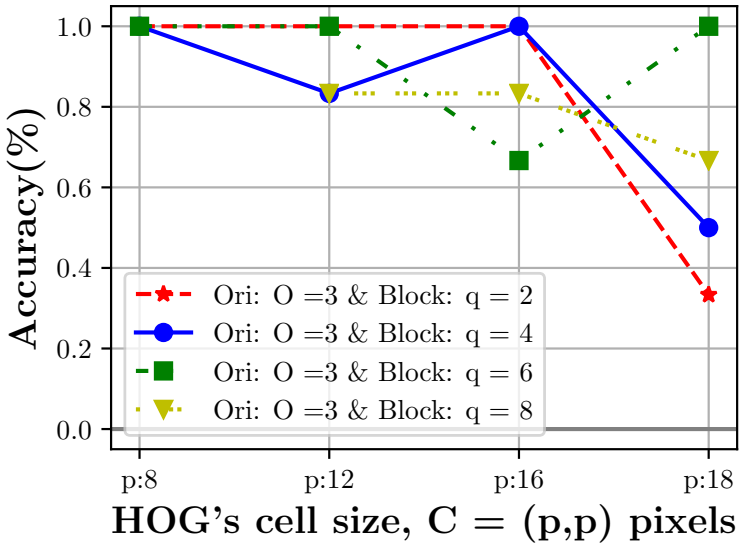


Figure 5.8: HOG's parameters analysis: Image size = (512,512), Orientation = 3, and for variable sizes of Blocks and Cells

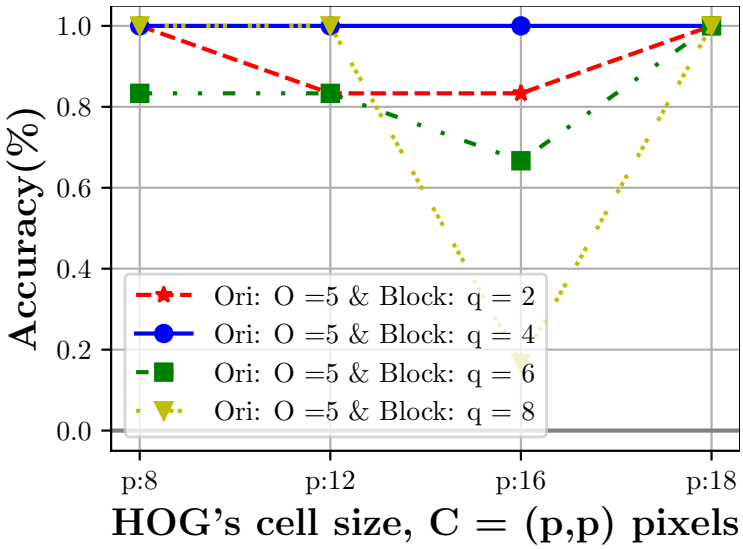


Figure 5.9: HOG's parameters analysis: Image size = (512,512), Orientation = 5, and for variable sizes of Blocks and Cells

notice that when we use a moderate value, we achieve good results despite the values of the other parameters. Given the fact that *HOG features* capture local shape information of the

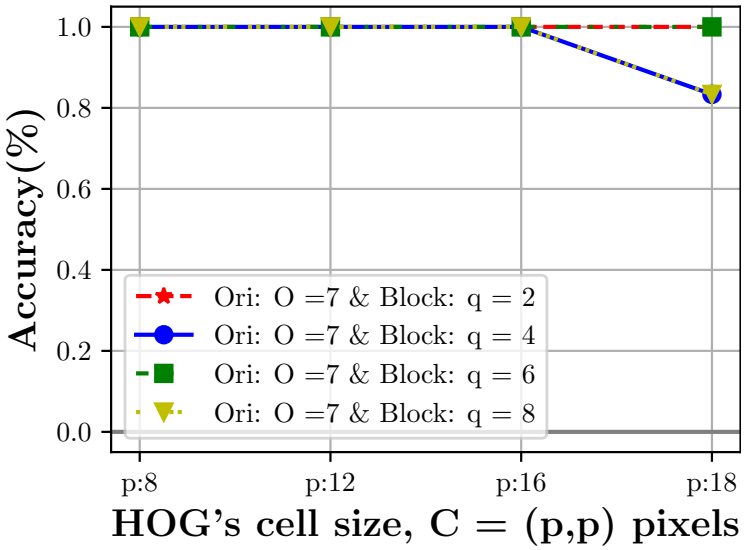


Figure 5.10: HOG's parameters analysis: Image size = (512,512), Orientation = 7, and for variable sizes of Blocks and Cells

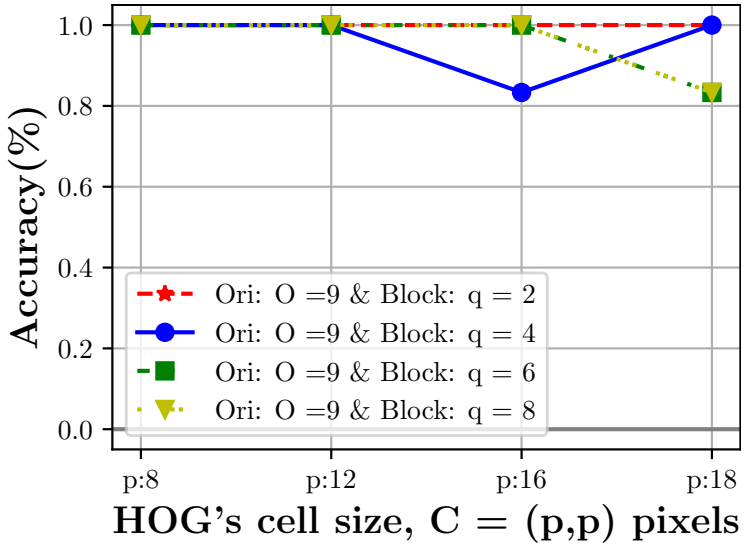


Figure 5.11: HOG's parameters analysis: Image size = (512,512), Orientation = 9, and for variable sizes of Blocks and Cells

TFR structures that are spatial invariance, those two observations can be justified. The block size $B = (q \times q)$ has an influence when the number of orientations is relatively small;

however, its impact almost vanishes when the number of orientations increase drastically. The take-away from this analysis is that, for each dataset we have, the hyper-parameter tuning is required to achieve the desired detection performance.

Table 5.1: Effectiveness of HOG features

Model	Detection Accuracy %
Our Model: {HOG + CNN}	97 (± 2%)
Baseline #1: {mean and STD} + SVM	65 (± 6%)
Baseline #2: {CQT+CNN}	85 (± 4%)

The effectiveness of the chosen HOG features in our methodology was verified by building two alternative baseline models. A brief description of the baseline model #1 (2 features and an SVM classifier) is as follows: we extract two features (namely the mean and standard deviation) out of each power trace. Then we train a Support Vector Machine (SVM) on the extracted features. Baseline model #2 (CQT and CNN) is as follows: we use the extracted CQT images (X_{cqt}) to train a CNN model. That is basically removing the HOG box in the methodology shown in Fig. 5.1 and feeding the CQT images directly to the model generation box. By doing so, we can investigate the impact of HOG transformation and show its effectiveness. Table. 5.1 shows that HOG based features outperform the implemented baseline models with at least 20% performance gain. It also confirms that HOG representations are more effective in terms of capturing better discriminative information than CQT features.

5.7.3 Detection coverage results

In this part of the results, we evaluate the proposed methodology in detecting different causes of anomalous behaviors in IoT and wireless (smartphone) devices. Table. 5.5 reports the the methodology's performance on our validation datasets. The table has been

structured to reflect the different causes of anomalous behaviors explained in Section. 3.4. Table. 5.5 demonstrates the accuracy and other performance metrics, namely, F-score, Recall, Precision, and AUC. It starts by showing the results obtained on the real malware dataset: Real (**H**), refer to Table. 4.2 in Section. 4.3. On four of the five apps the mean accuracy was higher than 86%, while the maximum accuracy on the 5 apps reached 100%. If we look into the recall, for instance, which refers to the percentage of time the model classifies a device's behaviour as anomalous and it is actually anomalous, the obtained recall performance is significantly high in the 5 cases.

Table 5.2: Summary of the results of detecting anomalous behaviours of devices caused by real malware

Source of Anomaly	Type of Anomaly	Scenario	Accuracy (%) (max, mean, std)	F-Score	Recall	Precisoin	AUC
Security Threat	Real Malware Apps (**H**)	Buscaminas	(1.00, 0.86, 0.12)	0.83	0.87	0.89	0.86
		Tetris	(1.00, 0.86, 0.19)	0.86	1.00	0.8	0.88
		Tilt	(1.00, 0.93, 0.13)	0.95	1.00	0.92	0.93
		Wordsearch	(1.00, 0.87, 0.13)	0.84	0.83	0.94	0.87
		Yams	(1.00, 0.75, 0.27)	0.66	0.62	0.75	0.81

Table 5.3: Summary of the results of detecting anomalous behaviours of IoT devices due to Cryptominer and DDOS attacks

Source of Anomaly	Type of Anomaly	Scenario	Accuracy (%) (max, mean, std)	F-Score	Recall	Precisoin	AUC
Security Threat	IoT attacks	DDOS (**A**)	(1.00, 0.95, 0.12)	0.96	1.00	0.95	0.93
		DDOS (**B**)	(1.00, 1, 0)	1.00	1.00	1.00	1.00
		CryptoMiner (**C**)	(1.00, 0.93, 0.09)	0.73	0.70	0.80	0.85

Table 5.4: Summary of the results of detecting anomalous behaviours of devices caused by emulated malware

Source of Anomaly	Type of Anomaly	Scenario	Accuracy (%) (max, mean, std)	F-Score	Recall	Precisoin	AUC
Security Threat	Emulated Malware (**E, F, G**)	100 UL 00 DL	(1.00, 0.91, 0.16)	0.90	0.93	0.90	0.92
		75 UL 25 DL	(1.00, 0.95, 0.04)	0.95	0.94	0.96	0.95
		50 UL 50 DL	(1.0, 0.99, 0.01)	0.99	1.00	0.99	0.99
		25 UL 75 DL	(1.00, 0.96, 0.045)	0.97	0.96	0.97	0.97
		00 UL 100 UL	(1.00, 0.91, 0.09)	0.92	0.88	0.96	0.92
		One Spike	(0.83, 0.67, 0.18)	0.59	0.63	0.53	0.50
		Random Activation	(1.00,0.85,0.13)	0.84	0.83	0.87	0.85

Table 5.5: Summary of the results of detecting anomalous behaviours of devices due to Faulty CPU

Source of Anomaly	Type of Anomaly	Scenario	Accuracy (%) (max, mean, std)	F-Score	Recall	Precisoin	AUC
Faulty Device	Faulty CPU (**D**)	3 cores are down	(1.00, 0.88, 0.29)	0.85	0.86	0.85	0.93
		2 cores are down	(1.00, 0.92, 0.17)	0.95	1.00	0.93	0.92
		1 core is down	(1.00, 0.60, 0.23)	0.46	0.57	0.4	0.60

Then, in the second part of Table. 5.5, we show the results obtained on IoT datasets, as outlined in Table. 4.1. We notice that the methodology is able to detect when an IoT device is under DDOS attack (datasets **A and B**) with a minimum accuracy of 95%. The interesting part here is that whether the IoT device is the source of an attack or a victim, our proposed approach is effective in detecting that. Moreover, as show in Table. 5.5, in CryptoMiner dataset (**C**), the mean detection accuracy is 93%.

In the third part in of Table. 5.5, we report different scenarios of our emulated malware. This part of the results confirms that the methodology generalizes well on a wide range of malware. Whether a malware has a cyclic behavior or random one, and whether its activities are mainly uploading data from the device (e.g., data theft), downloading data to the device, or a combination of both, the mean accuracy for most of the scenarios is higher than 90%. The only case that was difficult to detect was the single activation scenario, as shown in the table.

The last part of Table. 5.5 presents the performance of the methodology in detecting the anomalous behavior of devices due to a faulty CPU, described as dataset **D** in Table. 4.1. The more the number of defecting CPU cores, the easier it is to detect such an anomaly. This is expected, since the tasks carried out by the CPU will take longer once less cores are available. Therefore, the normal behavior of a device, where all the 4 cores are available in our experiment, will change if the number of cores decreases. The results of detection coverage across the wide range of validation datasets (reported in Table 5.5) shows very good performance, and hence, confirm the usefulness and applicability of the proposed approach.

5.7.4 Comparison results

In the final section of our results, we have implemented some of the previously reported power signal based anomaly detection techniques [122, 123][241] to compare their performance with ours on the same datasets. To adjust the available dataset to the methodology proposed in [241], a down-sampling of each of the signals was done in the beginning. After

Table 5.6: Comparison between our proposed technique and previously reported techniques

Model	Detection Accuracy %	F1 Score %
Our Model: {HOG + CNN}	96 (± 2%)	97 (± 2%)
[123]: {ICA+RF}	88.0 (± 2%)	87 (± 0%)
[122]: {ICA+ CC-CS*}	89 (± 5%)	80.1 (± 5%)
[241]: {FCC + GMM}	68.3 (± 5%)	72.2 (± 5%)

Note: ICA = independent component analysis, CC-CS* = Cross-Correlation comparison of Similarity, RF = random forest, MFCC=Mel-frequency cepstral coefficients,
GMM = gaussian mixture model.

that, the Mel Frequency Cepstral Coefficients (MFCC) features were extracted, followed by the Gaussian Mixture Models to discriminate between non-malicious and malicious signals. Table 5.6 shows a comparison of the average classification accuracy of the four techniques. The accuracy of the proposed methodology reaches $\sim$ 96.5%, while for the two references [123] and [122] it reaches $\sim$ 88-89%. On the other hand, the accuracy of reference [241] is $\sim$ 69%. As it is prominent, the proposed methodology surpasses all of the methodologies in terms of average detection accuracy with performance gain ranging from $\sim$ 9% to 17%.

5.8 Case study II: A Strongly Non-Intrusive Detection Methodology

To show that our HOG+CNN detection methodology has a wider scope of applicability, in this section we apply it to a different kind of data. However, the objective is the same as the objective throughout the chapters of this thesis. Instead of using the power consumption, we monitor the amount of traffic generated by a device as a signal to use it to detect when a device is misbehaving. We call such a monitoring approach, strongly non-intrusive detection.

The definition of the non-intrusive method only takes the computing overhead and malware's ability to spot the methods into account [124][214]. Anomalous behavior detection using power consumption data [156][9] is one example that utilizes non-intrusive observation method, and this data are collected by connecting the phone to a measurement tool. Thus, there is no computing overhead and the malware can not detect the presence of it. However, the observation method does interfere with and restricts the interactions between the user and the device as the measurement tool needs to be connected to the device. Besides, it requires changes to be made to the device to accommodate the measurement tool.

Therefore, we define a *strongly non-intrusive* method based on the following three criteria: i) The method does not run any extra code on the device so it does not incur any computing overhead. ii) The device is not explicitly connected with the monitoring tool. iii) The monitoring device is not explicitly communicating with the device under observation for monitoring purpose. A method is strongly non-intrusive if it satisfies all the three criteria.

Usage of network traffic signal as compared to power or thermal or any other signals can be justified as the data can be collected non-intrusively. The methodology only requires to have a look at the number of bytes/packets transmitted to and from the device over time. It does not depend on identifying IP address and type of data transmitted. Thus, it identifies anomaly without compromising user privacy. Also, it is inexpensive and easier to collect network traffic data as it does not requires any additional hardware such as power monitoring tool or thermal camera which are required to capture power signals and temperature signals respectively. Moreover, Due to these claims, the proposed monitoring methodology satisfies the definition of strongly non-intrusive.

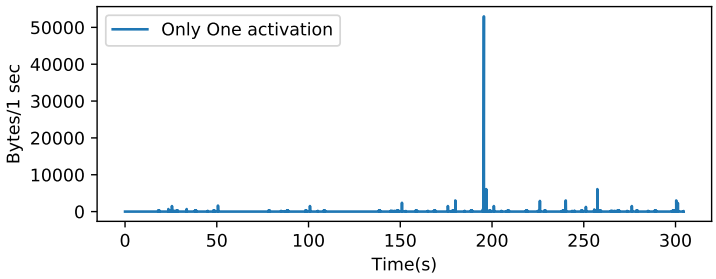
In this case study, we apply the methodology discussed in this chapter on the traffic data and compare it with the power based detection. We further compare our HOG+CNN approach with a similar study that is based on the changepoint analysis [156]. The comparison is done using the strongly non-intrusive approach (i.e., using network traffic data) and non-intrusive appraoch (i.e., using power signals).

5.8.1 Results

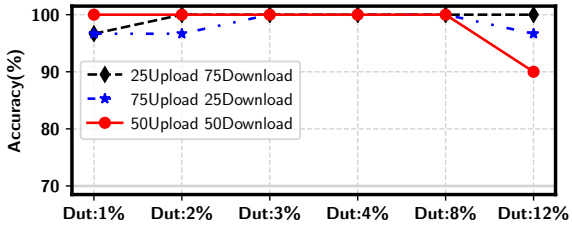
The used dataset is emulated malware based datasets. It is noteworthy that both techniques changepoint detection and HOG+CNN produce 100% of accuracy when the malware is only downloading or uploading (i.e., 100UL00DL and 00UL100DL, respectively) in all of the duty cycles. This is expected due to the fact that when the malware's activity involves only uploading or downloading tasks then it will have a clear and definite pattern reflected by the number of packets transmitted/received by a device. Comparing to the scenario where uploading and downloading are tested with different percentages for all the duty cycles, they will have more variability in their pattern and Thus, making more challenging to detect.

In addition, one can notice that the HOG+CNN methodology can detect malware with 100% of accuracy when uploading and downloading are tested with different percentages (i.e. 25UL75Dl, 50UL50DL, and 75UL25Dl) for all the duty cycles. On the other hand, changepoint detection can detect malware with less accuracy when the emulated malware is downloading or uploading with different percentages, as it is observed in Fig. 5.12 (b). However, the lowest accuracy in that case is 90%.

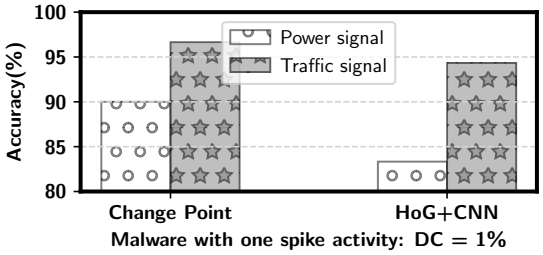
If we compare both the techniques when a malware is activated only once (the 1Spike dataset), it is observed that Changepoint detection can recognize it with greater accuracy as it can be seen in Fig. 5.12(c). Finally, the results obtained using the network traffic have been compared with the results cited in [156], which used power consumption data to detect malicious behavior. To have a fair comparison with the other papers [156][9], we consider the emulated malware scenario where it is only downloading a file with different duty cycles, as we can see in Fig. 5.12(d) & 5.12(e). The anomaly detection techniques perform better in terms of accuracy, reaching 100% when they are using network traffic data as opposed to the power consumption data. Hence, we can conclude that the network traffic data is a better feature to discriminate malicious behavior.



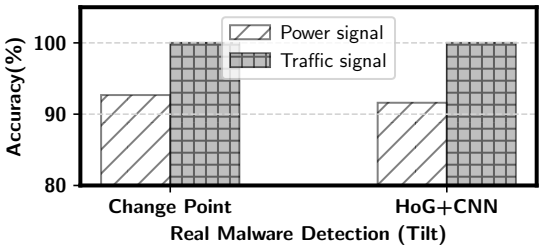
(a) 1% Duty Cycle and Only once activated malware behavior



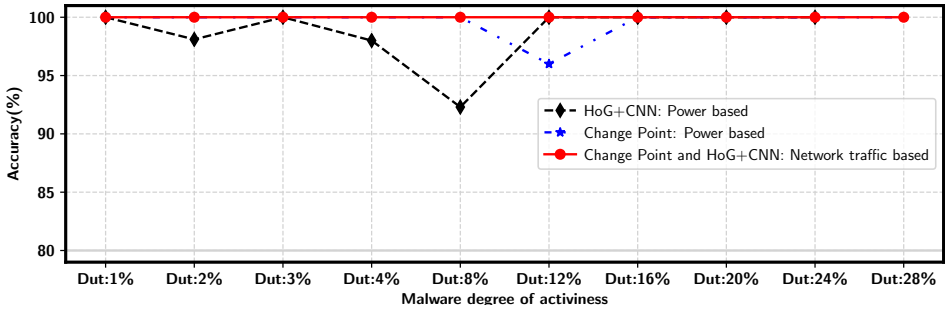
(b) Changepoint technique results: considering Network traffic



(c) Only one activation



(d) Real Malware Detection



(e) Comparison of Power and Network based approaches

Figure 5.12: Results of both techniques: Changepoint detection and HOG using Network traffic and power consumption data

5.9 Summary

In this chapter, we proposed a methodology that leverages the power consumption of devices, signal processing and machine learning to detect when they behave anomalously. The idea is to transform the problem of detecting anomalously behaving device into an image classification problem. The work investigated the utilization of HOG on TFR images to classify power signals of a device. Power consumption signals of devices were transformed to CQT images and further to HOG images. Then, a CNN model was learnt to classify signals. The results show the proposed anomalous behavior detection framework is very promising in detecting in detecting when a devise starts to behave anomalously. We validated our approach using: (i) real malwares taken from the Derbin dataset;(ii) a wide spectrum of emulated malware; (iii) well-known encountered security attacks on/from IoT devices(DDOS and Crypto Mining malware); and a faulty IoT device. The performance of the proposed methodology across all of the datasets shows considerably low number of false negatives, which is an important measure in evaluating such a methodology. Moreover, the framework outperformed previously published works by significant margins.As a next step, we plan to expand the validation to cover how quick we can detect certain attacks. For example,if there is a DDOS attack, how long the attack can be before we can detect it. We aim to expand the validation to include the detection of anomalous behavior of a device due to the change of the execution environment. Finally,we will exploring the transfer learning domain, that is, training HOG images on pre-trained CNNs and compare the performance.

Chapter 6

Solving the Limitation of Labeled Dataset Problem: An Unsupervised Approach

This chapter present our unsupervised based detection methodology. Similar to our supervised technique, this methodology also comprises of stages. In Feature Extraction Stage, we utilize utilizing stacked RBM AutoEncoders (AE) to process the collected information and unsupervisedly extract features. Then, in classification Stage, we train an OC-SVM on the extracted features to detect possible security threats.

6.1 Problem Description

In the field of this study, a major problem that is frequently faced in the practice is the lack of the availability of labeled datasets. In contrast to the proposed technique in Chapter 5, where the assumption was both classes (normal and anomalous) are available, a more common case is the availability of only one class, namely the normal behavior class. The challenge in this case is how to extract discriminative robust features in an unsupervised

fashion. Moreover, how to learn a classifier that generalizes well even though if it was trained on one class only (i.e., the normal behavior class).

6.2 Solution Strategy

Unlike our work in Chapter 5 and other studies that use the power consumption of devices to detect malware [123, 122, 241, 231, 134], the proposed approach in this chapter requires only the normal behavior of a device in the training phase. Therefore, this methodology aims at detecting new/unseen anomalous behaviors (e.g., DDOS, zero-day attacks). The way we achieve our aim in this chapter is as follows. Our detection methodology is based on (i) applying a sliding window to the signals to increase the number of observations of our dataset; (ii) utilizing stacked RBM AutoEncoders (AE) to process the collected information and unsupervisedly extract/learn features based on the reconstruction errors of AE; (iii) applying PCA to the reconstruction errors of AE, thereby isolating noise and outliers from the training of the reconstruction; and (iv) training an OC-SVM on the extracted/selected features to detect possible security threats. The solution strategy in this chapter adopts the same idea discussed in section 3.2.2 and uses the following two hypotheses: (i) Every piece of software, whether malware or legitimate, would have a fingerprint in the power consumption of the wireless device that makes it inevitable for malware to go undetected. (ii) The power consumption signals obtained from the wireless device that is "malware free" should have similar patterns (normal/expected behaviors). Such behaviors can be characterized by training AutoEncoders (AE). On the other hand, if the device starts to behave anomalously due to certain causes (security threats in our case), its power consumption signals should reflect that as a form of deviation from the norm. Consequently, the resultant AE output - reconstruction error - should be minimal if the wireless device is still behaving normally and maximal once the device starts to behave anomalously (maliciously).

6.3 Methodology

In this section we start by providing an overview on the tools used throughout our methodology. Then, we describe the proposed detection methodology and the idea behind it in details.

6.4 Preliminaries

Our methodology makes use of three well-known algorithms. In the following subsections we give the necessary background on: (i) the used technique to extract features - Restricted Boltzmann Machine (RBM); (ii) the employed dimensionality reduction technique - Principle Component Analysis (PCA), and finally (iii) the used classifier - One-Class Support Vector Machines (OC-SVM). Then, in section 6.5, we give the details of how these tools and algorithms are used to make up our detection framework.

6.4.1 Restricted Boltzmann Machine (RBM)

RBMs were firstly introduced by [109]. They are intended for nonlinear dimensionality reduction and feature extraction. Later, and due to their capability to efficiently model the training data distribution, they have been found to be a suitable option for anomaly detection applications [215, 157, 128].

RBM is a stochastic graphical model which learns a probability distribution over input dataset, while restricting its visible units and hidden units from forming a fully connected bipartite graph. As shown in Fig. 6.1, RBM AutoEncoder is comprised of two layers; visible layer and hidden interconnected using symmetrically weighted connections (weights). Units ($\mathbf{h}$) in the hidden layer capture higher-order correlations of the visible units ($\mathbf{v}$) in visible layer connecting it.

The training procedure minimizes the overall energy so that the data distribution can be well captured. The used energy function is defined by:

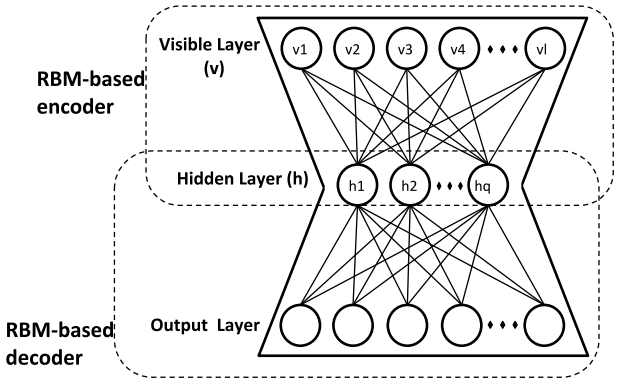


Figure 6.1: An RBM layer

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \mathbf{v}} bv_i - \sum_{j \in \mathbf{h}} ch_j - \sum_{i,j} v_i h_j w_{ij} \tag{6.1}$$

where $\mathbf{v}$ is the input signal vector that forms the visible units and h is a vector that forms the hidden units (features); $\theta = \{w, b, c\}$ are model parameters. Starting with random weights, the state of hidden units $\mathbf{h}$ is set based on the joint probability defined using:

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{1}{Z(\theta)} exp(-E(\mathbf{v}, \mathbf{h}; \theta)) \tag{6.2}$$

where $Z(\theta)$ is a normalizing factor called partition function. The marginal distribution over the visible layer $\mathbf{v}$ is:

$$p(\mathbf{v}; \theta) = \frac{1}{Z(\theta)} \sum_{h} exp(-E(\mathbf{v}, \mathbf{h}; \theta)) \tag{6.3}$$

We can build a more robust model that can sufficiently extract high level abstract features by stacking layers of RBMs to form stacked RBM AutoEncoders [106]. The stacked AE can be trained via a greedy layer-wise procedure. Each layer is trained as an RBM using contrastive divergence (CD) strategy (Gibbs sampling for example [92, 108]);

after each RBM layer of the stacked AE has been trained, weights are clamped and a new layer is added. For more details on the structure of RBM and stacked RBM AEs and the way they are trained, the reader is directed to [106, 108, 73].

6.4.2 Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique that reduces the number of features in the dataset, and at the same time, it tries to retain as much information as possible by preserving the maximum variance in the original dataset [204]. Therefore, by linearly transforming the original data points to another space, the inherent structure of the data makes it easier to recognize/classify. PCA is performed by using singular value decomposition (SVD). Given $\mathbf{X}$ which is a matrix of signals of size $M \times n$. Each row is signal/observation x that forms a vector of a size $1 \times n$. The first step of PCA is to mean center the matrix of signals, which is accomplished by subtracting the mean of signals from each signal x_i as shown in Eq. 6.4.2

$$\hat{x_i} = x_i - \mu_i, \quad \mu_i = \frac{1}{M} \sum_i^M \boldsymbol{x}_i$$

The resultant mean centred matrix $\hat{\mathbf{X}}$ (it is of the same size as $\mathbf{X}$) is then decomposed using SVD,

$$\hat{\mathbf{X}} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}$$

Here, $\mathbf{U}$ is a $n \times M$ unitary matrix, $\mathbf{V}$ is a $n \times n$ unitary matrix, and $\boldsymbol{\Sigma}$ is a diagonal matrix comprising the singular values of $\hat{\mathbf{X}}$ in decreasing order [75]. A reduced dimensional representation of $\hat{\mathbf{X}}$ can be obtained by discarding columns of $\mathbf{U}$ and $\mathbf{V}$,

$$\hat{\mathbf{X}} \approx \mathbf{U}_{0:j}\boldsymbol{\Sigma}_{0:j,0:j}\mathbf{V}^{\mathrm{T}}_{0:j}$$

Here, j denotes the number of columns (principle components) retained.

To this end, we have explained the theory of computing PCA using SVD; however, in practice computing SVD is computationally complex. Thus, several efficient algorithms are widely used to compute PCA efficiently. In the implementation of our methodology, we use the well-known scikit-learn Python library [176] to compute the PCA. For more details on the scikit-learn PCA's implementation, the reader is directed to [165].

6.4.3 Classification - One Class SVM

One-Class Support Vector Machines (OC-SVM) is an anomaly detection technique. It was originally developed by scholkopf [193] to identify novelty unsupervisedly. In his formulation [193], given a training set $\mathbf{X} = \{x_i\}_{i=1}^{m}$, the algorithm learns a function (soft boundary) that returns $+1$ in a region capturing "majority" of the training observations if x_i falls within "normal region", and -1 elsewhere. In the case where the data is not linearly separable, the so called kernel trick is applied, where each point is implicitly projected to a higher dimensional feature space (through linear/nonlinear kernel) to separate the data set from the origin. Then one needs to solve the following quadratic optimization problem:

$$Minimize \;\; \frac{1}{2}||W||^2 + \frac{1}{\upsilon m}\sum_{i=1}^{m}\xi_i - \rho \tag{6.4}$$

subject to

$$\langle W, \Phi(x_i)\rangle \geq \rho - \xi_i, \quad \xi_i \geq 0 \tag{6.5}$$

The parameter $\upsilon \in (0, 1]$ sets an upper bound on the fraction of outliers and a lower bound on the number of training examples used as support vectors.

The idea behind OC-SVM is to implicitly project the input data space to a higher dimensional feature space through linear/nonlinear kernel function. These projected vectors in feature space are called feature vectors. Then, OC-SVM tries to find a curve or a

curved surface in the feature space separating the feature vectors into two parts of normal or abnormal events.

Since only data from one-class is available (power consumption signals obtained from normal behaving wireless devices), the objective of using OC-SVM is to learn a function that returns $+1$ for the normal behavior region (**not anomalous**) and -1 elsewhere - malicious behavior region (**not anomalous**). In order to obtain more versatile decision boundaries with OC-SVM, we use two kernel functions (namely Sigmoid and RBF kernel) to investigate their effectiveness on the detection performance. We train the OC-SVM using the a feature matrix E' with M rows (number of observations) and W' columns (number of features).

6.5 Model Pipeline

The intuition behind our methodology is that power consumption signals obtained from a wireless device that is "malware free" should have similar patterns (normal/expected behaviors). Such behaviors can be characterized by training Stacked RBM AutoEncoders. On the other hand, if the device starts to behave anomalously due to certain causes (malware presence in our case), its power consumption signals should reflect that as form of deviation from the norm. Consequently, the resultant AE's output - reconstruction error - should be minimal if the wireless device is still behaving normally and maximal once the device starts to behave anomalously (maliciously). The framework illustrated in Figure 6.2 is designed based on the above stated intuition.

The basic concept of our methodology can be explained using two main procedures. Procedure 1, shown in Fig. 6.2 compromises of two main stages, namely *Feature Extraction* stage and *Model Generation* stage. Procedure 1 is used to train and test the model on the wireless devices collected dataset. Procedure 2, shown in Fig. 6.5, which makes use of the learned models from Procedure 1, describes the process of labeling a new unlabeled measurement obtained from a wireless device as whether it came from a compromised

device or not. Below, we briefly explain these two procedures, and then in subsection 6.5.1 we describe the methodology in more details.

The structure of our Feature Extraction stage starts with data augmentation to increase the number of observations used in the training phase. Following the data augmentation, we use a pretrained RBM AE to extract the features. The structure of the pretrained RBM AE is kept simple with two RBMs only. Such a simple/shallow structure is chosen to reduce the complexity of the methodology. Going for a deeper structure increases the computational complexity. Therefore, in order to make the methodology suitable and efficient for online and real-time detection, we chose to keep the model size minimal as long as it provided us with a good detection performance across the validation scenarios. Finally, we apply PCA to reduce the dimensions of the extracted features space so that the classifier avoids the over-fitting problem, and the resultant detection performance is optimized.

Although combining RBM AE and PCA structure seems redundant, as both RBM and PCA are commonly used for dimensionality reduction, the way we use the pretrained RBM AE does not overlap with the functionality of PCA. We use the pretrained RBM AE to extract features in a novel way, as we show in the following sections. In our particular case, PCA complements the feature extraction process in a flexible and lightweight manner. The flexibility is in terms of exploring the number of features (of principal components) that can be used in the classification phase. However, if we used stacked RBMs to do the whole process (i.e., (i) extracting the features and then lowering the dimensionality in one shot, and (ii) exploring how many features are needed to achieve a satisfactory detection performance) that will require a deep stacked RBMs' structure, which will be computationally costly. Moreover, in practice, having a deep stacked RBM is not suitable for real-time applications, like detecting security threats in a quick manner.

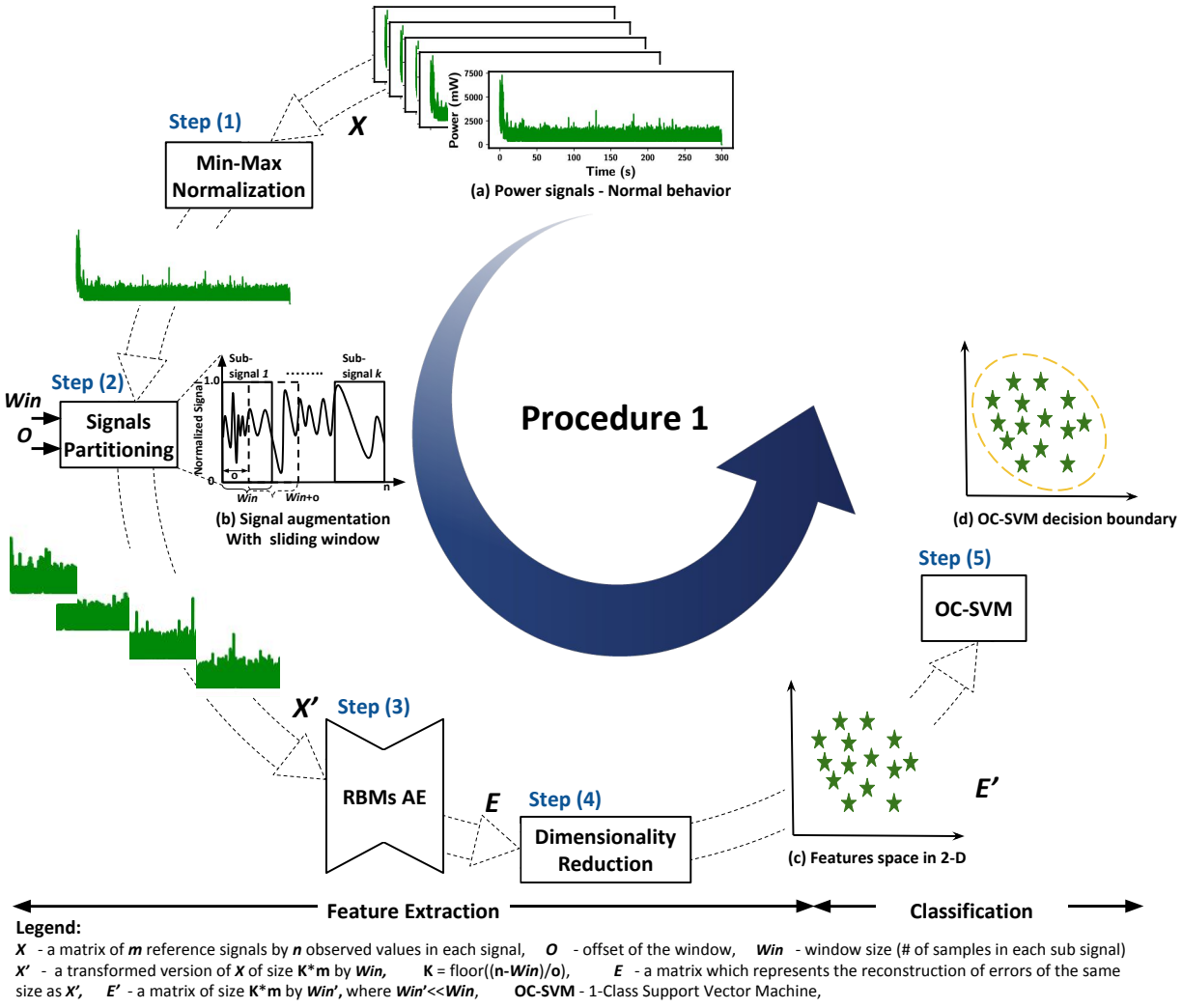


Figure 6.2: Overview of the methodology: Signals Transformation and Model Training

6.5.1 Procedure 1: Features Extraction and Model Generation

The input data to Procedure 1 is $\mathbf{X}$, which represents the raw dataset used for training and testing. The formal definition of $\mathbf{X}$ is as follows: $\mathbf{X} = \{x_1^{(t)}, x_2^{(t)}, ..., x_i^{(t)}, ..., x_m^{(t)}\}$, which is a matrix of size $m \times n$. $x_i^{(t)}$ represents one reference signal (time-series) and is taken by keeping the same time duration and conditions in terms of configuration of a

wireless device. Each of the rows which forms the matrix $\mathbf{X}$ is represented by $x_i^{(t)}$, which is described as follows: $x_i^{(t)} = [x_1, x_2, ..., x_i, ..., x_n]$. Hence, $x_i^{(t)}$ is a vector of size $1 \times n$. In other words, the elements in $x_i^{(t)}$ represent the values of the power consumed by a device sampled at frequency F_s for a time $t = T$. Next, we show how this data is transformed (features extraction) and used to train the models (model generation) in the two stages of our framework.

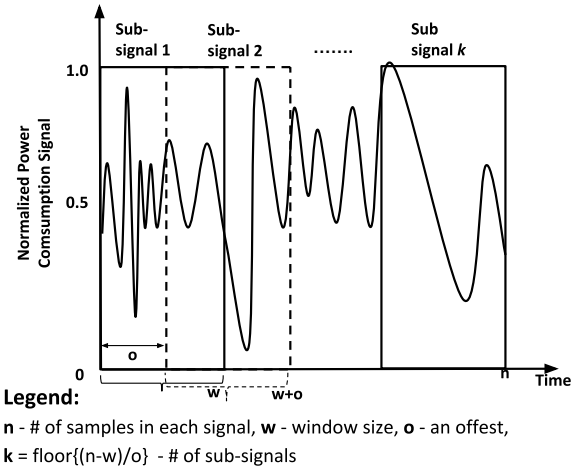


Figure 6.3: Data Augmentation with Sliding Window

Stage 1: Feature Extraction (FE)

The first step in this stage is data pre-processing where we start with *Min-Max Normalization*. Since power consumption signals from different devices have different scales, we normalize (using the formula described in Eq. 6.6) all signals to be in the range [0,1]. This is also necessary for a faster convergence when training the AE.

$$x^{(t)\prime} = \frac{x^{(t)} - \min(x^{(t)})}{\max(x^{(t)}) - \min(x^{(t)})}; \quad where \quad x^{(t)} \in \mathbf{X} \tag{6.6}$$

Signals Partitioning with Sliding Window: Then, for each normalized power signal we apply a sliding window to partition the signals into sub-signals as shown in Fig. 6.3. This

is step (2) in our data pipeline model shown in Fig. 6.2. Based on a window size W and an offset o, we partition each signal accordingly (we call this process " Data Augmentation". The output of this process is a transformed version of our input dataset $\mathbf{X}'$. $\mathbf{X}'$ is also a matrix of size $m * K \times W$, where $K = \frac{n-W}{o}$. The reason behind this step is that in the case where the dataset is limited (small dataset), implementing a way to augment the data is very important to deploy a robust model and avoid the over-fitting problem. Next, we use the subsignals in $\mathbf{X}'$ to train the RBM AE and compute the reconstruction error.

Pre-trained Stacked RBM AutoEncoders: This is the main block in our FE stage (Step (3) in Fig. 6.2). As shown by Hinton et al. in [109], an RBM AutoEncoder can be built using a pretrained RBMs. Such an approach, RBM based AutoEncoders, showed a good performance in different domains such as speech recognition [168], obstacle detection [65], text categorization [83], and fault diagnosis [198]. The structure of our stacked RBMs AE includes 2 pretrained RBMs, each of a size (number of hidden units, h) 3000 and 500, respectively. We use power consumption signals obtained from a "malware free" device to perform greedy layer-wise unsupervised training to the two RBM layers. After data pre-processing, we use X' as an input to train the stacked RBM AE. Specifically, we train each of our RBMs (RBM1 and RBM2 Fig. 6.4- (a)) individually, as suggested in [109, 107]. Then, we create a stacked RBMs AE, as shown in Fig. 6.4– (b), by stacking the two RBMs and unrolling them to create the RBM based AE.

As shown in Fig. 6.4, the input of the AE is the pretrained RBM1's visible layer, and the learned feature's activation of the RBM1 (h1) are set to the visible layer of the next layer, RBM2. This is what makes the encoder part. The decoder is formed by adding an equal number of the opposite layers using transposes of the base encoder's weights. Finally, the entire system can be treated as a feedforward traditional AE, at this point. We fine-tune the RBM AE using the backpropagation algorithm described in Algorithm 2. The objective is to minimize the mean squared error (i.e., obtain an optimal reconstruction of the normal behavior of the device) and iteratively updates the parameters of the stacked AE. After initializing the network parameters with the pertained RBMs' weights, we compute the mean squared error and then update the parameters accordingly until they qualitatively

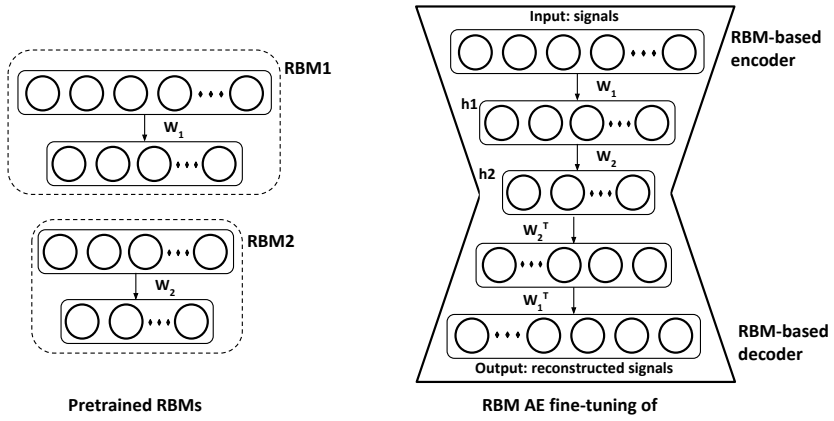


Figure 6.4: The used stacked RBM AE

converge using the stochastic gradient descent method.

Algorithm 2: Stacked AE training algorithm.

Input : Dataset $\mathbf{X}'$:
$x'_i,\ i^{th}\ training\ example,\ i \in \{1, ..., k\}$
Output: Optimal parameters $\Theta^* = \{\Theta^*_E, \Theta^*_D\}$
Initialization of Network Parameters Θ_E, Θ_D;
while *epoch* $< N_{epochs}$ **do**
  Compute mean square error (MSE)
  $L_{MSE}(\Theta; x'_i) = \frac{1}{k}\sum_{i=1}^{k} ||x'_i - \mathcal{D}(\mathcal{E}(\mathbf{x} \mid \Theta_E) \mid \Theta_D)||^2$
  Update Parameters Θ_E, Θ_D using SGD
end

In detection procedure (Procedure 2), the input is the pre-processed/manipulated observations $\mathbf{X}'$, and the output is the resultant reconstruction residual error $\mathbf{E}$.

Dimensionality Reduction (DR): Given that we are using a large window size and offset to augment the data, as pre-processing step, we can generate huge number of sub-signals. In order for us to avoid the over-fitting problem when training the OC-SVM classifier, and also to extract main components (which contain most of the discriminative information) from the AE's output, we apply dimensionality reduction techniques, namely Principal Component Analysis (explained earlier), to the AE's reconstruction residual error vectors

before training the classifier. The input is the resultant reconstruction error $\mathbf{E}$, and the output is $\mathbf{E'}$, which is a reduced version of $\mathbf{E}$ of a size $m \times K'$, where K' is the number of selected principle components, and $K' \lll W$, as show in Fig. 6.2 - step (4).

Putting FE steps together: To give an illustrative example of the Feature Extraction stage, assume that: (1) we obtained 15 signals from a device, i.e., m=15; (2) each signal represents the power consumed by a device for 300 seconds, and that the power is sampled at 5,000 samples/second. This gives us a power trace that has 1,500,000 samples, i.e., $\mathbf{n} = 1,500,000$. So our dataset X is a matrix of size $(\mathbf{15 \times 1,500,000})$. After applying the explained normalization, we use the formula given in the sliding window step. To compute $\mathbf{K}$, assume further that we use $Win = \mathbf{15,000}$ and an offset $O = 2000$. This gives us $\mathbf{K} = \mathbf{742}$ subsignals for each original signal. In other words, each signal x_i becomes 742 subsignals (x'_i). Thus, after the data augmentation step, the 15 power signals of our became 11,280 subsignals. To keep the notations consistent, the size of our dataset X' becomes $(\mathbf{11,280 \times 15,000})$. The next step is to use the trained RBM AE to extract the features. To generate the features, we pass these subsignals (x'_i), each is a vector of size $(\mathbf{1 \times 15,000})$, through the trained RBM AE. The trained RBM AE's output should be a reconstructed version of the input (x'_i). We then compute the loss/error for each subsignal using the mean squared error (mse). Thus, the generated features vector for each signal x_i is 742 features. Each feature represents the computed reconstruction error of the subsignals of the original signal. Refereeing back to the notations used in Fig. 6.2, the size of our dataset matrix (E) at this point is $(\mathbf{15 \times 742})$. The final step is to use the PCA to lower the dimensions of the dataset to the desired number of principal components $(\mathbf{K\prime})$. We use the obtained dataset $(E\prime)$ to train the classifier, as we explain in the following section.

Classification: OC-SVM

The task of detection in our methodology is performed using a semi-supervised learning technique. As shown in Fig. 6.2, we employ One-Class Support Vector Machines (explained earlier) as an anomaly detection technique. Since only data from one-class is available

(power consumption signals obtained from normal behaving wireless devices), the objective of using OC-SVM is to learn a function that returns +1 for the normal behavior region (**not anomalous**) and −1 elsewhere - malicious behavior region (**anomalous**). In order to obtain more versatile decision boundaries, we use two kernel functions (namely Sigmoid and RBF kernel) to investigate their effectiveness on the detection performance. We train the OC-SVM using the feature matrix E' with M rows (number of observations) and W' columns (number of features).

6.5.2 Procedure 2: Detection Procedure

The detection step shown in Fig. 6.5 also includes 2 stages, reconstruction error concatenation and classification. The reconstruction error concatenation stage is concatenating the reconstruction errors of every window split from each signal. Then, using PCA to do feature extraction and finally perform the classification using the SVM algorithm.

6.6 Results

The problem of detecting malicious behavior in wireless devices is a semi-supervised classification problem. Consequently, given the datasets that we have, we perform binary classification to validate our method. The healthy behavior – Normal class - is tested against its anomalous behavior version – Malware class. Based on the this formulation, we have conducted several experiments to train and test our methodology. The analysis covered three main components, namely, (1) visualization of the extracted features, (2) justification for the use of OC-SVM, and (3) detection performance of the methodology and a comparison. The training and testing experiments were performed on a powerful analysis platform (Colab) [53]. The deep learning tasks were carried out in Python using Google Tensorflow framework and Keras [63] Python libraries. The PCA and OC-SVM were performed using the well-know scikit-learn Python library [176].

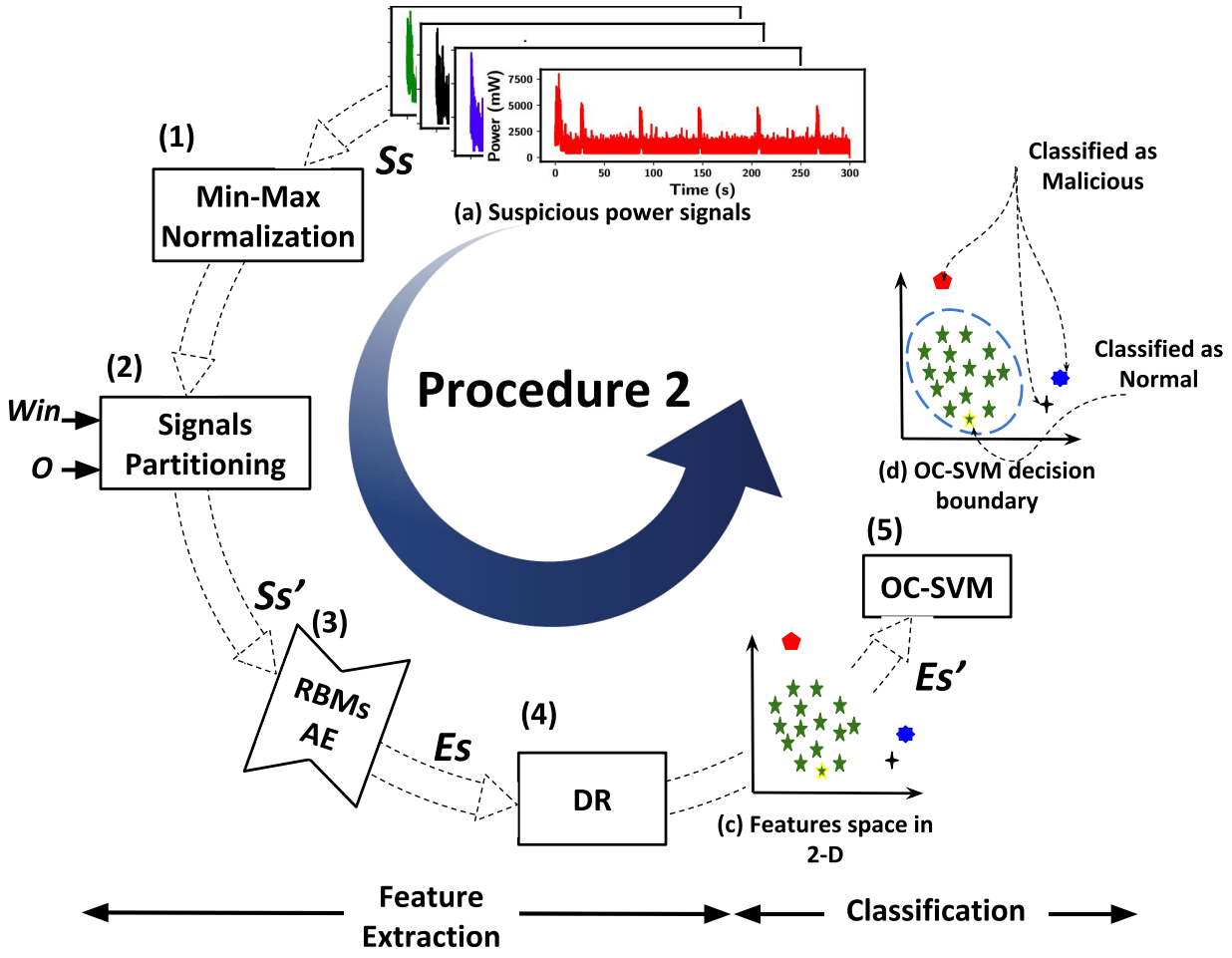


Figure 6.5: Detection of Anomalous Behavior

6.6.1 Features' visualization and results

The dataset used to generate the following visualizations is E-DC. We choose the malware family 50UL50DL, which represents malware that utilize % of its active period to upload information to the cloud and the remaining 50% to download from a remote server. In this family of malware we have 6 different malware instances each with specific "activity Duty

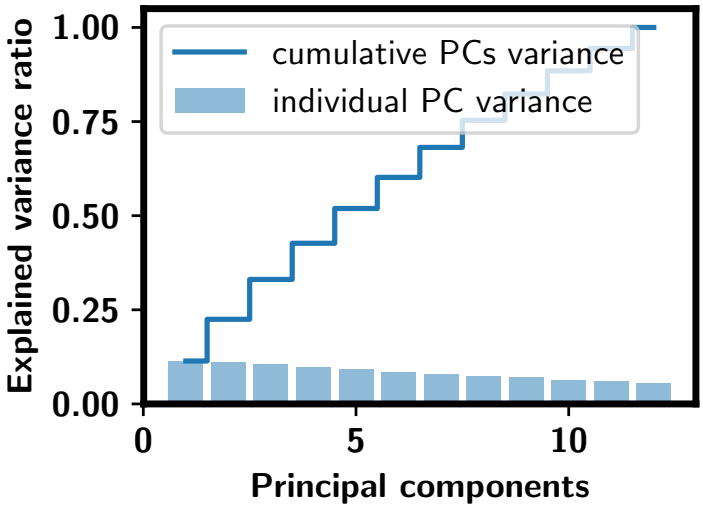


Figure 6.6: The percentage of explained variance for each pc and the cumulative variance

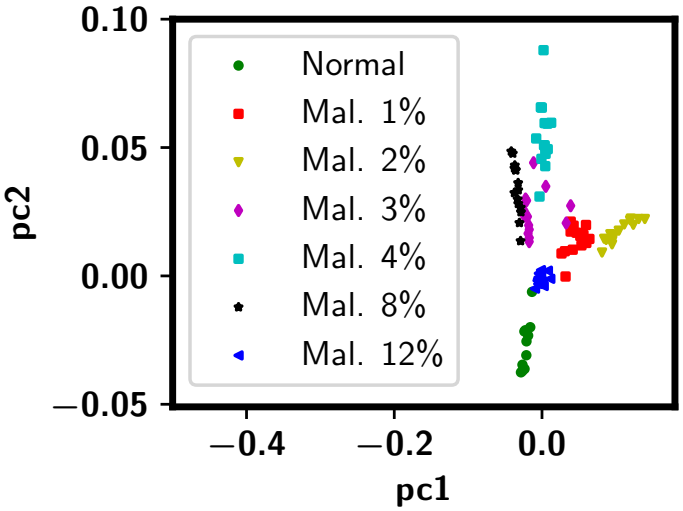


Figure 6.7: Malware classes distribution - (Mal. = Malware)

Cycle" (DC), as we explained in Chapter 4.8. The duty cycles (DC) of the malware in this family are $DC = i\%, i \in \{1, 2, 3, 4, 8, 12\}$. Since we use PCA as a dimensionality reduction technique, the first step we took to investigate how effective the RBM-AE's features are is to visualize the percentage of variance of Normal class of this dataset (50UL50DL).

Figure 6.6 shows the percentage of the variance that the first 12 principal components contain. It also show the cumulative PCs variance, which shows that most of the variance (almost %) is contained in these 12 components. Then in Fig. 6.7 we show how the data points from normal class and the six malware instances (from 50UL50DL malware family) are distributed using PCA's first two components. As can be noticed in Fig. 6.7, although dimensionality reduction loses some information and visualization results of low-dimensional space cannot fully reveal what are in the high-dimensional space, it is still insightful that the extracted features carry good information that can be used to construct an effective detection model. In the next part of the results, we show the impact of the number of the chosen principle components on the methodology overall detection performance.

6.6.2 Classifier performance analysis

After visualizing the output of our methodology's feature extraction stage, next we evaluate the performance of the chosen classifier - OCSVM. To training OC-SVM, there are some model specific hyper-parameters that needs to be tuned in order to achieve the best detection performance. We use cross-validation and grid search to optimise the performance of the classifier. We start by randomly splitting the dataset, the Normal class data, into training and testing datasets. Then, we use the training portion to perform 5-fold cross-validation. In this step we basically split the training dataset into 5 portions and use 4 of them to training the OC-SVM and evaluate the learned model on the 5th fold. We repeat this step 5 times and finally using the grid-search we pick the parameters that gives us the best detection performance.

Before we report the results, we explain the metrics used to evaluate the performance of our methodology. Since our problem is anomaly detection problem, which means, in practice, we have much data from the Normal class and few observations from the anomalous class, Malware in this case. We report the accuracy (Acc), recall (R), and precision (P) in some of the results; however, we have chosen to use the F1-score (F1) as the main

metric to evaluate the detection performance of our methodology, since it combines both recall and precision. These measures are computed as shown in Table 6.1. We find that reporting the precision and recall individually is very important as well since they put more weight to a false positive and false negative, respectively.

Table 6.1: Performance evaluation metrics

$Acc = \frac{TP+TN}{(TP+FP+TN+FN)}$	$P = \frac{TP}{TP+FP}$	$R = \frac{TP}{TP+FN}$	$F1 = 2 \times \frac{P \times R}{(P+R)}$ %

Figure 6.8 illustrates the decision boundary found by the kernelized OC-SVM. The used kernel function is Radial Basis Function (RBF), which is a kind of Gaussian kernels. The decision boundary shows that the model is able to separate most of the positive (Normal class observations) from the negative points correctly. It also shows that the learned model is able to correctly the unseen observations from the Normal class as well (green circles with black edges in the figure).

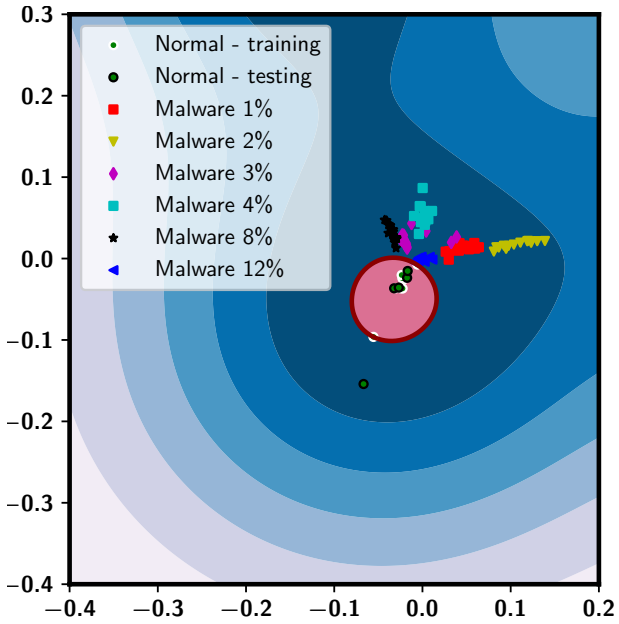


Figure 6.8: OC-SVM decision boundary and malware classes distribution

Figure 6.9 highlights the impact of the OC-SVM kernel functions on the accuracy of the model for different evaluation metrics. It can be noted that Radial Basis Function (RBF) gives the best detection performance over the Linear and Sigmoid kernels. It is, hence, used in all of the following results. Furthermore, the impact of PCA on the classification performance was verified. The way we investigate that is to vary the number of principle components (pc) used to train our OC-SVM and observe the detection performance accordingly, as shown in Fig. 6.10. The results show that first 2 pc(s) produce the best detection performance. This confirms with the observation made from Fig. 6.8 that the first 2 pc(s) contain much of the variance (Figure -a) and the fact that visualizing the first 2 pc(s) showed a good separation of the malware instances of the 50UL50DL malware family. Moreover, in Fig. 6.10, it can be noted that the more the number of pc(s) is the lower the detection performance we get. This is attributed to the problem of over fitting, where having more features (high dismissions) in the case of limited-sized datasets can get the trained model to fall in the over fitting problem quickly.

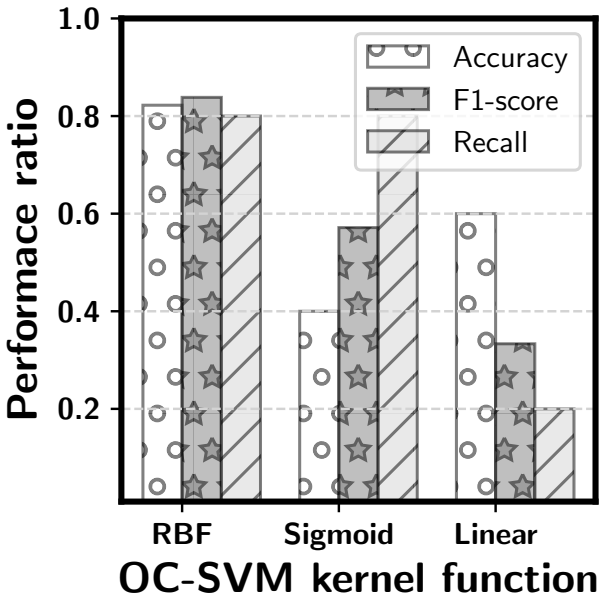


Figure 6.9: OC-SVM performance analysis for different kernels

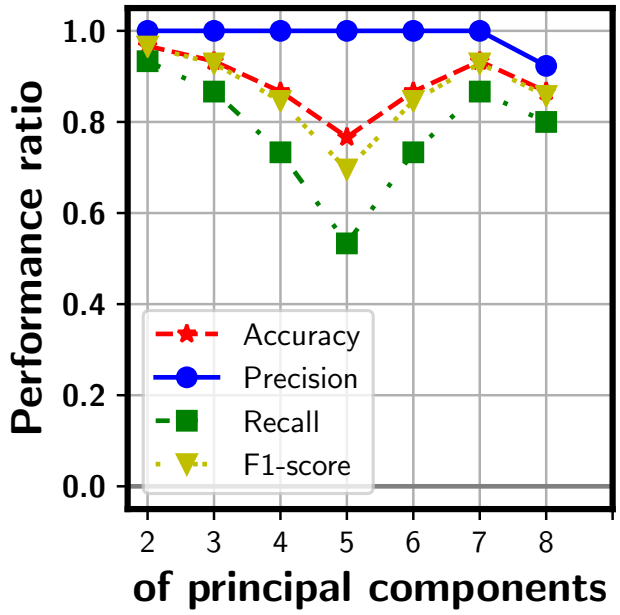


Figure 6.10: # of principle components impact on the detection performance

6.6.3 Comparison results

To demonstrate the effectiveness of our AE based features extraction technique, we compare the average accuracy and F1-score of our methodology against a baseline model, as well as against previously reported techniques [241, 123, 8]. A brief description of the used baseline model is as follows: We extract two features (namely, the mean and standard deviation) out of each power trace. Then we train the traditional Support Vector Machine (SVM [206]) on the extracted features. Since the objective in [241, 123, 8] is similar to the objective of our the work in this chapter (malware detection in wireless devices), and the nature of the raw data is the same as the data considered in this study (i.e., the power consumed by devices treated as signals), this technique was chosen to be the best fit to have a fair comparison with. In the literature, there are a couple of studies that share the same objective; however, we felt that the comparison would not be fair since the type of the raw data (EM emissions [130, 131, 172]) is completely different.

Table 6.2: Comparison results

Model	Detection Accuracy %	F1 Score %
Our Model: {RBM AE + OC-SVM}	98 (± 2%)	96.55 (± 2%)
Baseline: {mean&STD} + SVM	65 (± 6%)	67 (± 5%)
[123]: {ICA+RF}	88.0 (± 2%)	87 (± 0%)
[8]: {ANN}	90.1 (± 5%)	-
[241]: {FCC + GMM}	68.3 (± 5%)	72.2 (± 5%)

Table 6.2 shows that AE based features outperform two of the techniques, the baseline and [241], with at least 30% performance gain. To compare with the [241], we have implemented their approach and run it on our dataset. The obtained accuracy of our approach reached 97%, while the accuracy of [123, 8] was 88% and 90%, respectively. We argue that such a good detection performance is due to: (1) the ability of the stacked RBM AE to learn good features that capture the complexity of the normal behavior of the device under study; and (2) the high sampling rate that is used in our monitoring methodology. Such a rate makes it possible to capture information about the short-lived events executed onboard a device. Since [241] depends on data sampled at a rate of 5 samples per second, the approach fails to perform well as the results in Table 6.2 show.

Although [123, 8] were based supervised learning and used the same dataset as ours (dataset E-DC in Table 4.2), which is sampled at the same sampling rate as the one used in this study, the performance of our methodology shows at least 7% detection improvement over [123] and [8]. The main reason goes back to the point (1), i.e., the good features extracted by the stacked RBMs. It is important to note that the performance of our methodology is not very far from the performance reported in [123]; however, our methodology has two main advantages over [123]: (1) it requires fewer computation resources, which makes it more suitable for real-time detection and resource constrained applications, and (2) it is

a semi-supervised based technique which makes it more practical. A final remark on the impact of sampling rate, our conclusion based on the comparison with [241] showed that the higher the sampling rate is, the better the is detection performance. This is aligned with the findings reported in [8], where they show that the detection performance can worsen as the sampling rate is lowered.

Figure 6.11 shows a comparison we performed to ensure that the chosen classifier is a good pick. In Figure 6.11, we compare OC-SVM with two other unsupervised methods namely, Robust covariance (RC) [177] and Isolation Forest (IF) [149]. In this comparison we show how the three methods compare in terms of the F1-score performance as well as the time to recall the model. It can be noticed that Robust covariance and Isolation Forest performance is always 0.8 irrespective of the malware activeness degree (DC %). The reason is that the normal behavior is used in training all of the classifiers, so the decision's boundary of the modeled data (normal behavior) does not change as the malware DC % changes.

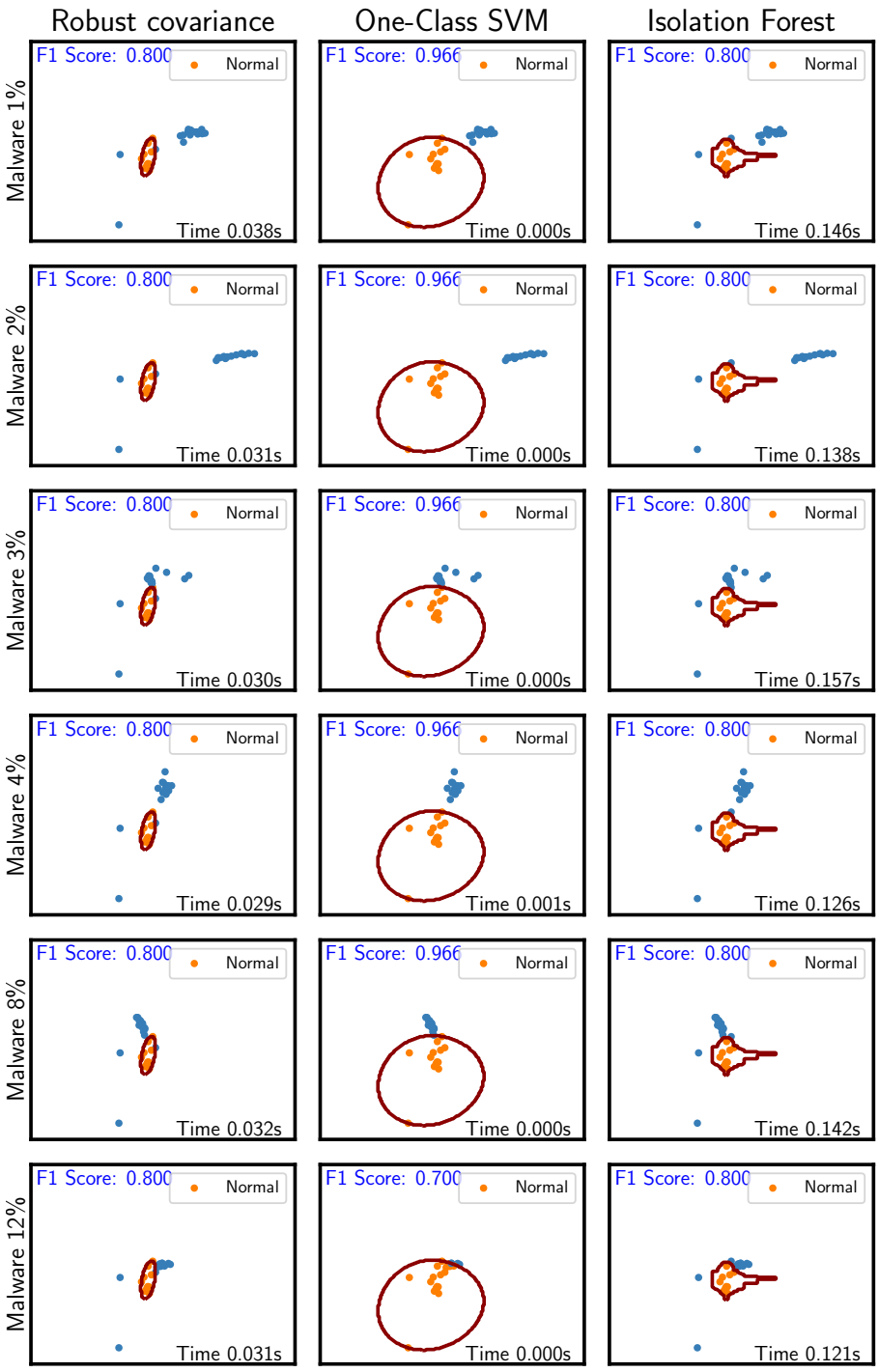


Figure 6.11: Classifier performance comparison

The distribution of the malware points falls outside of the normal region across the version of malware scenarios in Fig. 6.11. It is also worth mentioning that the OC-SVM sudden drop in the case of malware with DC = 12 %; some points for an unrecognized reason, seemed similar to the normal behavior and fall in the normal region, i.e., classified wrongly as not malicious. Clearly, OC-SVM is the fastest in terms of predicting new observations and also out performs the RC and IF classifiers with significant gain throughout the 6 malware instances ($DC = i\%, i \in \{1, 2, 3, 4, 8, 12\}$). Whether the malware is with $DC = 1\%$ - the least active case, or the malware with $DC = 12\%$ - the most active case; the model shows good detection performance regardless how active the malware is, where the obtained F1-score reaches 96.6%, thanks to the good features learned by the RBM AE.

6.6.4 Detection coverage results

In this section of the results, we show the performance of the proposed methodology in detecting different security threats that targets wireless/IoT devices. In the applications of anomaly detection, including detection of security threats, we are more interested in finding out when a security threat is present but does not get detected, the so-known as false negative (FN). Therefore, when analyzing the performance of our methodology we do not only report the accuracy, rather we show the F1-scoare, recall, and precision as well, as they are more indicative in our case. Table 6.3 shows a summary of our the performance of our methodology on the seven datasets that we have. The most challenging detection task was encountered on the E-1S dataset. The obtained F1-score is not as high as on the other datasets; however, having a high recall for in this particular case indicates a low FN. This means in most of the cases, our methodology was able to correctly label and detect the presence of malware, even though in the case of E-1S the malware is only active once in a random fashion. The performance on E-RA dataset is also considerably lower than the other datasets, which is something we expected given the fact that its activation times and periods are randomized. Both of E-1S and E-RA are adaptive malware, which means that they tend to hide their operation and activity so that they go undetected.

Table 6.3: Summary of the results across all of the datasets - (* averaged results)

Metric \ Threat	E-DC*	E-1S	E-RA	Real*	DDOS-A	DDOS-V	Cry-M
F1-score	0.91	0.7	0.80	0.87	0.985	0.98	0.90
Accuracy	0.89	0.6	0.83	0.81	0.98	0.98	0.86
Recall	0.933	0.93	1.00	0.93	0.97	0.97	0.97
Precision	0.90	0.56	0.67	0.83	1.00	1.00	0.85

In the real malware dataset, Real, the reported results in Table 6.3 are averaged over the five malicious apps that the dataset contains. Overall, the results are in the range of 85%. Figure 6.12 demonstrates the detection performance in terms of the selected four measures (F1-score, accuracy, recall, and precision). These results justify that the used emulated datasets cover a wide range of malicious behaviors/attacks, hence their value in evaluating new detection techniques. In all of the IoT device datasets, namely distributed of service attacks (DDOS-V and DDOS-A) and the crypto mining (Cry-M) datasets, the detection accuracy as well as the F1-score reach 90% in most of the cases. So whether a devices is under attack or it is hacked to serve as the source of the attack, the device's owner can be notified and alarmed of such a cyber-threat.

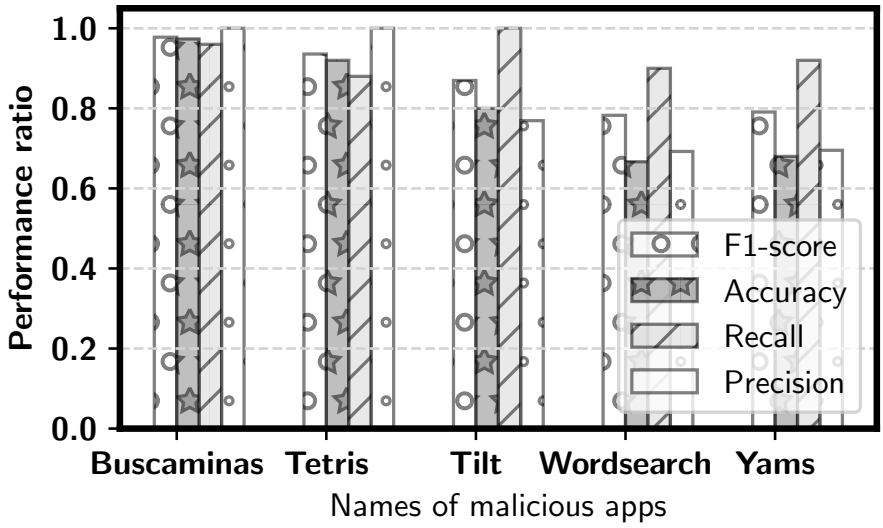


Figure 6.12: Real malware per application results

One might argue that DDoS and cryptocurrency mining malware are high power malicious attacks by nature, thus are easy cases to detect using the proposed approach. This argument makes the need to implement emulated malware, where we can control the amount of time a malware can be active, even more compelling. As we explain in the Validation Datasets section, a wide range of malicious behaviors/activities (e.g., a malware that is active for one short time only: dataset E-1S; a malware with 1% DC activity: dataset E-DC, or a malware with random activation periods: dataset E-RA) are generated to represent malware that does not consume/use a lot of power. The detection performance, as shown in Table 4, of the proposed approach on these scenarios confirms that our approach is not only effective on security threats that are caused by attacks/malware that consume much of power; rather, it is also effective on the attacks/malware that is rarely active and consumes a small amount of power.

To further elaborate on the effectiveness and the practicality of the proposed approach, a common question in the security field is: can the detection be evaded if attackers find out about the proposed detection approach? Generally, this is a difficult task since there is

an inherent connection between the tasks a device performs and the power it consumes. In the side-channel attacks, the whole idea of revealing secret keys depends on the aforementioned fact. However, one way that an attacker might seek to evade detection is through the attempt to hide some of their malware's activities behind a legitimate power consumption bursts/spikes. In our validation datasets scenarios, specifically in the emulated malware families, we simulate such a strategy by having a high power activity application (YouTube) running as the normal/legitimate behavior, while the emulated malware runs in the background. In most of the scenarios (E-DC, E-RA, and E1S) shown in Fig. 6.13 and Table 6.3, our methodology was performing well (91, 80, and 70% F1-score, respectively). The 1Sipke (E-1S) emulated malware case shows a relatively low precision, where we have high false positives (i.e. normal behavior identified as malicious).

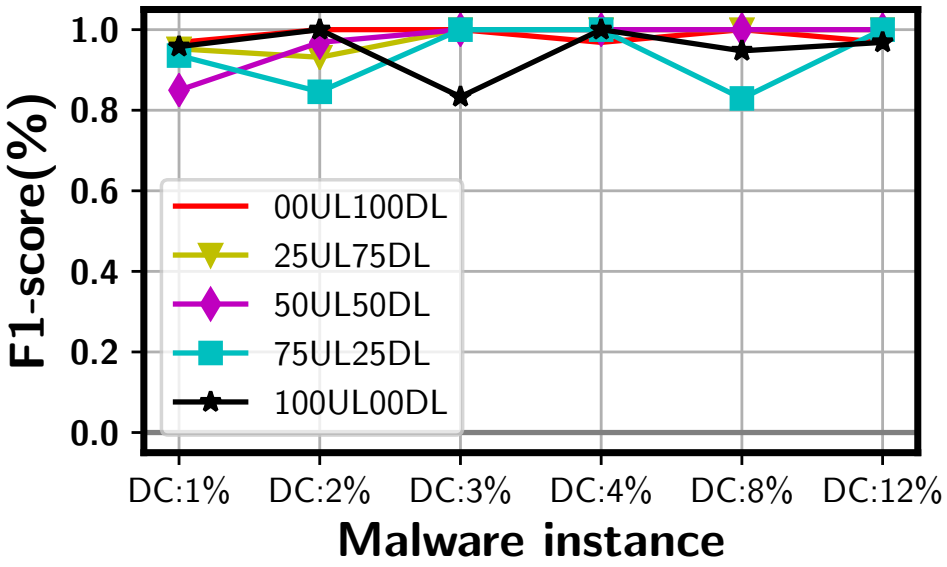


Figure 6.13: Results of non-adaptive emulated malware families

Finally, Figure 6.13 depicts the performance of our methodology on our non-adaptive families of malwares (E-DC dataset). This figure can be interpreted as follows: each malware family, ex., 00UL100DL family, has six malwares, namely malware with $i\%$ *DC* (duty cycle), where $i \in 1, 2, 3, 4, 8, 12$. The 1% *DC* Malware is malware that is only active

1% of the cycle period. So if the cycle period is 60 seconds, then this malware is only active for 1 second and goes to sleep for the remaining 59 seconds. The obtained F1-score for this malware family is very high throughout all the malware families and ranges between 83% to 100%. Whether the malware is only active for short period of time, the case of 1% *DC* malware, or the most active malware 12% *DC*, where the malware stays active for case 7.2 seconds, the methodology was capable of detecting all of them. A similar trend is noticed for the other malware families, as the results in Fig. 6.13 illustrate. The results of detection coverage across the wide range of validation datasets were very good, and hence, confirm the usefulness and applicability of the proposed approach.

6.7 Summary

In this chapter, we introduced a new malware detection technique that is based on a non-intrusive device's power consumption monitoring. The measured power readings of a device are treated as signals carrying discriminative information that can be learned. The idea is to learn robust features out of the power signals using stacked RBM AutoEncoders and OC-SVM. We validated our approach using real malwares taken from the Derbin dataset as well as from wide spectrum of emulated malware. The obtained results from real malware as well as emulated malware confirm the effectiveness of the proposed technique. The performance of the proposed methodology was compared to previously reported approaches. Finally, the methodology was also applied on well-known attacks (namely, DDOS attack and Crypto Mining malware) on IoT application dataset where the obtained results show a robust detection performance.

Chapter 7

Conclusions and Future Work

This chapter provides a brief summary of the thesis, together with recommendations for future directions. Section 7.1 concludes the thesis while Section 7.2 highlights the main points for further research that will extend this work.

7.1 Conclusion

This work is motivated by the fact that since IoT and CPS devices found in safety-critical applications run on limited computing resources, they have been the target of different kinds of cyber-security attacks on a global level. As a result, effective and robust approaches for detecting anomalous device behavior can minimize a significant amount of financial and life losses. Previous studies and results have demonstrated some success; however, several recent incidents confirm that the need for a second line of defence is preeminent. Therefore, the objective of this thesis is to propose new approaches that leverage the power consumption of devices and deep learning techniques to make these devices more resilient against different kinds of attacks and failures.

Several challenges are addressed in order to fill the research gaps and achieve the objective of this thesis. The main contributions of this thesis are summarised as follows:

- This thesis extensively reviews the associated literature and identifies the tools as well as the proposed techniques that mainly consider *side-channel information* and also conduct *dynamic analysis* in order to detect the anomalous behavior of IoT/CPS devices. Such a review has never before been conducted. This survey provides information such as the nature of information used for analysis, the type of anomalies causing such behavior, the tools used for analysis, and the reported detection performance.

- Since the work in this thesis is mainly concerned with the reasons why a device may behave anomalously, it is important to model such behavior. Therefore, this work models the main factors that affect a device's normal behavior as an anomaly of one of the following forms: (i) security threats (e.g., DDOS attack, malware, ransomware, resources hijacking); (ii) change in the execution environment (e.g., communication interference); or (iii) a faulty device (e.g., faulty component or hardware aging). This thesis also investigates the effectiveness and usefulness of using the power consumption signals of devices to infer some effective insights about the "operational health" of these devices, specifically in detecting security threats and faulty devices.

- A main block to achieving the goals of this thesis is overcome by building power consumption-based datasets that can be utilized by AI and security research communities to validate newly developed detection techniques. The collected datasets cover a wide range of anomalous device behavior, namely three main aspects of device security (i.e., confidentiality, integrity, and availability) and partial system failures. The extensive experiments include: a wide spectrum of various emulated malware scenarios; five real malware applications taken from the well-known Drebin dataset; distributed denial of service attack (DDOS), where an IoT device is treated as: (1) a victim of a DDOS attack and (2) a source of a DDOS attack; cryptomining malware, where the resources of an IoT device are being hijacked to be used to advantage of the attacker wish; and faulty CPU cores. The wide range of the collected datasets allow extensive validation of the proposed detection approaches to be performed. These datasets, and details of the experiments, will be made public and available to the

research community in the near future. It is noteworthy that this level of extensive validation has not yet been reported in any study in the literature.

- This thesis presents a novel *supervised technique* to detect anomalous device behavior based on transforming the problem into an image classification problem. The main aim of this methodology is to improve the detection performance. For this reason, the employed technique is a supervised model (i.e., requiring labeled data). In order to achieve the goals of this study, the proposed methodology combines two powerful computer vision tools, namely Histograms of Oriented Gradients (HOG) and a Convolutional Neural Network (CNN). In this methodology, the 1-D instantaneous power consumption signals of devices are transformed into a time-frequency representation (TFR), using Constant Q Transformation (CQT), to form 2-D images. The argument is that CQT images capture valuable information about the tasks performed on board a device, e.g., events' location in time and frequency domains, frequency of events, and the shape and duration of events. Applying HOG to the CQT images enables the extraction of more robust features that preserve the edges of time-frequency structures (i.e., a description of local information) as well as the directionality of the edge information (i.e., how these structures/events evolve with time). Finally, a CNN model was trained on images containing HOG features to classify these signals and detect anomalously behaving devices. Such a detection technique is not only useful for this particular case, but can contribute to most time-series classification (TSC) problems.

- The final component of this thesis is the proposal of a novel *semi-supervised detection technique* that requires only the normal behavior of a device in the training phase. This methodology, which aims at detecting new/unseen anomalous behaviors, leverages the power consumption of a device and Restricted Boltzmann Machine (RBM) AutoEncoders (AE) to build a model that makes them more robust to the presence of security threats. The methodology makes use of stacked RBM AE and Principal Component Analysis (PCA) to a extract feature vector based on AE reconstruction errors. A One-Class Support Vector Machine (OC-SVM) classifier was then trained

to perform the detection task. Such a technique is important since obtaining sufficient labeled data for training/validation of the models used by anomaly detection techniques is an acknowledged major issue in this field of study.

7.2 Future Work

This work can be extended in several directions. Some of these directions, which complete and compliment the contributions made in this thesis, emerged at later stages of the study. Other directions can be considered as orthogonal or parallel to this study and target different domains.

- *Achieving a wider validation coverage:* The validation of this work can benefit from the design of further experiments to cover all the causes of anomalous device behavior as discussed in this thesis. In particular, this would include the design of experiments related to the anomalous behavior that is the result of hardware aging and change of execution environment. Moreover, covering more scenarios of anomalously behaving devices due to a faulty components can explored in the future. Maintaining these datasets can strengthen the validation of the proposed approaches and prove robustness to the work.

- *Studying the time needed to detect an event/anomaly (speed):* In the context of this thesis, it is important to investigate the time required to detect a security threat. For instance, if a DDOS attack is initiated, it is important to know how long it will take before it can be flagged.

- *Increasing the size of the datasets using generative adversarial network (GAN) [94]* It will be interesting to Investigate the use of generative adversarial network (GAN) to overcome the challenge of insufficient/limited "power behaviors" of devices for the training phase. This can be very useful to improve the detection performance and

help detect known and unseen behaviors (e.g., attacks or failures) on different IoT devices.

- *Localizing the timestamp of an anomaly (event) in a time-series signal:* In certain applications, given power (time-series) signals, can the methodology identify the location of the anomalous behavior in these signals? This is an interesting task and worth exploring in the future.

- *Exploring transfer learning [70]:* Transfer learning is defined as "the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned" [220]. In the supervised detection technique in this present work, a convolutional neural network (CNN) is trained to perform the detection task. Given the fact that obtaining labeled data is always a challenge, it would be interesting to explore the impact of using pre-trained CNNs on the detection performance.

- *Monitoring that is strongly non-intrusive:* A previous study [11] defines a *strongly non-intrusive* method based on the following three criteria: i) the method does not run any extra code on the device, therefore does not incur any computing overhead; ii) the device is not explicitly connected with the monitoring tool; iii) the monitoring device is not explicitly communicating with the device under observation for monitoring purposes. A method is strongly non-intrusive if it satisfies all three criteria. While the monitoring method in this thesis falls under the category of non-intrusive methods, it is not strongly non-intrusive. Therefore, another direction that can be explored is the use of thermal images to detect a device's anomalous behavior. Following the same idea of using side-channel information analysis, the heat dissipated during the operation of a device can be used as source information to detect security threats. Such a monitoring approach satisfies the aforementioned definition, hence will be interesting to incorporate with the detection techniques proposed in this thesis.

- *Addressing wider applicability in different domains:* Since the nature of our collected

data is discrete and sequential (i.e., time-series signals), such data can be found across different applications and domains. In fact, time-series signals can be captured by monitoring physical systems (e.g., earth seismic dynamics [161], vibration of a mechanical system [35], or a robot arm in an automobile plant [13]), human bodies (e.g., motion, electrocardiogram (ECG), or electroencephalographic(EEG)), and acoustic sources (e.g., sound and audio). This argument gives the proposed detection techniques a broader scope of applicability that is worth investigating.

References

[1] Proof-of-concept ransomware for smart thermostats demoed at defcon. *[Online: last accessed in Jun. 2018]. Available: https://boingboing.net/2016/08/08/proofof-concept-ransomware-fo.html*, 2016.

[2] State of enterprise iot security: A spotlight on manufacturing, Sep. 2019.

[3] Z. Abbasi, M. Kargahi, and M. 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] H. Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[5] H. al-Khateeb, G. Epiphaniou, A. Reviczky, P. Karadimas, and H. Heidari. Proactive threat detection for connected cars using recursive bayesian estimation. *IEEE Sensors Journal*, 18(12):4822–4831, June 2018.

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

[7] 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.

[8] A. Albasir, R. S. R. James, K. Naik, and A. Nayak. Using deep learning to classify power consumption signals of wireless devices: An application to cybersecurity. In *2018 IEEE Int. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2018.

[9] A. Albasir, R. Manzano, and K. Naik. Deep learning based approach for classifying power signals and detecting anomalous behavior of wireless devices. In *2019 IEEE Services Workshop On Cyber Security And Resilience In The Internet Of Things (CSRIoT)*.

[10] Abdurhman Albasir, Kshirasagar Naik, and Ricardo Manzano. Towards improving the dependability and security of iot and cps devices: An ai approach. *ACM Trans. on Digital Threats: Research and Practice (DTRAP)*, 1(3), 2020.

[11] Abdurhman Albasir, Kshirasagar Naik, Ricardo Manzano, Part Shah, and Nitin Naik. A strongly non-intrusive methodology to monitor and detect anomalous behaviour of wireless devices. In *2020 International Symposium on Systems Engineering (ISSE)*, pages 1–8. IEEE, 2020.

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

[13] Riyadh Nazar Ali Algburi and Hongli Gao. Health assessment and fault detection system for an industrial robot using the rotary encoder signal. *Energies*, 12(14):2816, 2019.

[14] C. Alippi, V. D'Alto, M. Falchetto, D. Pau, and M. Roveri. Detecting changes at the sensor level in cyber-physical systems: Methodology and technological implementation. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1780–1786, May 2017.

[15] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. A just-in-time adaptive classification system based on the intersection of confidence intervals rule. *Neural Networks*, 24(8):791–800, 2011.

[16] A. Alnasser, H. Sun, and J. Jiang. Cyber security challenges and solutions for v2x communications: A survey. *Computer Networks*, 151:52–67, 2019.

[17] S. Althunibat, A. Antonopoulos, E. Kartsakli, F. Granelli, and C. Verikoukis. Countering intelligent-dependent malicious nodes in target detection wireless sensor networks. *IEEE Sensors Journal*, 16(23):8627–8639, Dec 2016.

[18] Abdelfattah Amamra, Chamseddine Talhi, and Jean-Marc Robert. Smartphone malware detection: From a survey towards taxonomy. In *2012 7th International Conference on Malicious and Unwanted Software*, pages 79–86. IEEE, 2012.

[19] O. Amft and K. Van Laerhoven. What will we wear after smartphones? *IEEE Pervasive Computing*, 16(4):80–85, October 2017.

[20] 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.

[21] A. Antonini, F. Maggi, and S. Zanero. A practical attack against a knx-based building automation system. In *2nd International Symposium for ICS & SCADA Cyber Security Research 2014 (ICS-CSR 2014)*, 2014.

[22] J. Antunes and N. F. Neves. Using behavioral profiles to detect software flaws in network servers. In *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, pages 1–10, Nov 2011.

[23] A. Arabo and B. 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, May 2013.

[24] 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.

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

[26] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, 2014.

[27] 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.

[28] Gsm Association et al. The mobile economy 2018. *London: GSM Association*, 2018.

[29] A. Azmoodeh, A. Dehghantanha, M. Conti, and R. 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.

[30] Amin Azmoodeh, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE Trans. on Sustainable Computing*, 2018.

[31] 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*, Aug 2017.

[32] A. Bagnall, A. Bostrom, J. Large, and J. Lines. The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version. *arXiv preprint arXiv:1602.01711*, 2016.

[33] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference IMC '09*. ACM, 2009.

[34] R.J. BAXLEY, C.J. Rouland, and M.T. Engle. Anomalous behavior detection using radio frequency fingerprints and access credentials, December 3 2015. US Patent App. 14/610,659.

[35] N Baydar and Andrew Ball. Detection of gear failures via vibration and acoustic signals using wavelet transform. *Mechanical Systems and Signal Processing*, 17(4):787–804, 2003.

[36] E. Bellini, F. Bagnoli, A. A. Ganin, and I. Linkov. Cyber resilience in iot network: Methodology and example of assessment through epidemic spreading approach. In *2019 IEEE World Congress on Services (SERVICES)*, volume 2642-939X, pages 72–77, July 2019.

[37] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[38] Yogi Berra. *You Can Observe a Lot by Watching: What I've Learned about Teamwork from the Yankees and Life*. Wiley, 2008.

[39] M. Birkmose, L. Kristensen, Jakob M. Nielsen, and H. Odborg. Applying web services as middleware for integrating embedded systems in loosely coupled environments. *Project report, Aalborg University*, 2004.

[40] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[41] Abdur Rahim Biswas and Raffaele Giaffreda. Iot and cloud convergence: Opportunities and challenges. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 375–376. IEEE, 2014.

[42] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. An android application sandbox system for suspicious software detection. In *2010 5th International Conference on Malicious and Unwanted Software*, pages 55–62. IEEE, 2010.

[43] B. Boashash. *Time-Frequency Signal Analysis and Processing, Second Edition: A Comprehensive Reference*. Eurasip and Academic Press Series in Signal and Image Processing. Academic Press, second edition edition, 2016.

[44] 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.

[45] 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.

[46] Leo Breiman and JH Friedman. Ra olshen and cj stone,". *Classification and regression trees*, 1984.

[47] 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.

[48] R. Bridges, J. Jiménez, and J. Nichols. Towards malware detection via cpu power consumption: Data collection design and analytics (extended version). *arXiv preprint arXiv:1805.06541*, 2018.

[49] J. Brown. Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1), 1991.

[50] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM, 2011.

[51] I. Butun, P. Österberg, and H. Song. Security of the internet of things: vulnerabilities, attacks and countermeasures. *IEEE Communications Surveys & Tutorials*, 2019.

[52] J. Capella, J. Campelo, A. Bonastre, and R. Ors. A reference model for monitoring iot wsn-based applications. *Sensors*, 16(11):1816, 2016.

[53] T. Carneiro, R. V. Medeiros Da Nóbrega, T. Nepomuceno, G. Bian, V. H. C. De Albuquerque, and P. P. R. Filho. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 6:61677–61685, 2018.

[54] 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.

[55] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3), 2009.

[56] V. Chatzigiannakis and S. Papavassiliou. Diagnosing anomalies and identifying faulty nodes in sensor networks. *IEEE Sensors Journal*, 7(5):637–645, May 2007.

[57] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, Franziska Roesner, and Tadayoshi O. K. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, volume 4, pages 447–462. San Francisco, 2011.

[58] K. Chen, S. Gupta, E. Larson, and S. Patel. Dose: Detecting user-driven operating states of electronic devices from a single sensing point. In *2015 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 46–54. IEEE, 2015.

[59] P. Chen, S. Lin, and C. Sun. Simple and effective method for detecting abnormal internet behaviors of mobile devices. *Information Sciences*, 321:193–204, 2015.

[60] J. Choi, H. Jeoung, J. Kim, Y. Ko, W. Jung, H. Kim, and J. Kim. Detecting and identifying faulty iot devices in smart home with context extraction. In *2018 48th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2018.

[61] J. Choi, M. O. Mughal, Y. Choi, D. Kim, J. A. Lopez-Salcedo, and S. Kim. Cusum-based joint jammer detection and localization. In *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–5, Oct 2018.

[62] F. Chollet. *Deep learning with python*. Manning Pub. Co., 2017.

[63] F. Chollet et al. Keras, 2015.

[64] 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.

[65] Clement Creusot and Asim Munawar. Real-time small obstacle detection on highways using compressive rbm road reconstruction. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 162–167. IEEE, 2015.

[66] B. Cui, H. Jin, G. Carullo, and Z. Liu. Service-oriented mobile malware detection system based on mining strategies. *Pervasive and Mobile Computing*, 24:101–116, 2015.

[67] 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.

[68] I. Cvitić, M. Vujić, et al. Classification of security risks in the iot environment. *Annals of DAAAM & Proceedings*, 26(1), 2015.

[69] 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.

[70] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*, pages 193–200, 2007.

[71] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. IEEE, 2005.

[72] Anusha Damodaran, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H Austin, and Mark Stamp. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, 13(1):1–12, 2017.

[73] Maisa Daoud and Michael Mayo. Using swarm optimization to enhance autoencoders images. *CoRR*, abs/1807.03346, 2018.

[74] T. Dargahi, A. Dehghantanha, P. Bahrami, M. Conti, G. Bianchi, and L. Benedetto. A cyber-kill-chain based taxonomy of crypto-ransomware features. *Journal of Computer Virology and Hacking Techniques*, Aug 2019.

[75] R. M. A. Dawson, Z. Shen, D. A. Furst, S. Connor, J. Hsu, M. G. Kane, R. G. Stewart, A. Ipri, C. N. King, P. J. Green, R. T. Flegal, S. Pearson, W. A. Barrow, E. Dickey, K. Ping, C. W. Tang, S. Van. Slyke, F. Chen, J. Shi, J. C. Sturm, and M. H. Lu. Design of an improved pixel for a polysilicon active-matrix organic LED display. In *SID Tech. Dig.*, volume 29, pages 11–14. 1998.

[76] Anthony Desnos and Patrik Lantz. Droidbox: An android application sandbox for dynamic analysis. *Lund Univ., Lund, Sweden, Tech. Rep*, 2011.

[77] Bryan Dixon, Shivakant Mishra, and Jeannette Pepin. Time and location power based malicious code detection techniques for smartphones. In *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, pages 261–268. IEEE, 2014.

[78] 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.

[79] M. Dong, P. Lai, and Z. Li. Can we identify smartphone app by power trace?[extended abstract for special session]. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 373–375. IEEE, 2013.

[80] R. N. Duche and N. P. Sarwade. Sensor node failure detection based on round trip delay and paths in wsns. *IEEE Sensors Journal*, 14(2):455–464, Feb 2014.

[81] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):6, 2012.

[82] U.K. Egham. 20.8 billion iot devices by 2020, Feb. 2018.

[83] Fatima-Zahra El-Alami, Abdelkader El Mahdaouy, Said Ouatik El Alaoui, and Noureddine En-Nahnahi. A deep autoencoder-based representation for arabic text categorization. *Journal of Information and Communication Technology*, 19(3):381–398, 2020.

[84] A. El Attar, R. Khatoun, and M. 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.

[85] Daniel R Ellis, John G Aiken, Kira S Attwood, and Scott D Tenaglia. A behavioral approach to worm detection. In *Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 43–53, 2004.

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

[87] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. Technical report, CISCO white paper, 2011.

[88] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[89] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. Gaur, M. Conti, and M. Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 17(2):998–1022, 2015.

[90] H. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller. Deep learning for time series classification: a review. *arXiv preprint arXiv:1809.04356*, 2018.

[91] I. Garitano, R. Uribeetxeberria, and U. Zurutuza. A review of scada anomaly detection systems. In *Soft Computing Models in Industrial and Environmental Applications, 6th International Conference SOCO 2011*, pages 357–366. Springer, 2011.

[92] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:721–741, 1984.

[93] R. Gilbert. *Defending Against Malicious Software*. UNIVERSITY OF CALIFORNIA - Santa Barbara, 2011.

[94] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[95] P. Gope and T. Hwang. Bsn-care: A secure iot-based modern healthcare system using body sensor network. *IEEE Sensors Journal*, 16(5):1368–1376, March 2016.

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

[97] D. Guo, A. Sui, Y. Shi, J. Hu, G. Lin, and T. Guo. Behavior classification based self-learning mobile malware detection. *JCP*, 9(4):851–858, 2014.

[98] Mordechai Guri, Gabi Kedma, Boris Zadov, and Yuval Elovici. Trusted detection of sensitive activities on mobile phones using power consumption measurements. In *2014 IEEE Joint Intelligence and Security Informatics Conference*, pages 145–151. IEEE, 2014.

[99] R. Hamming. *Digital filters*. Courier Corporation, 1998.

[100] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1095–1108, New York, NY, USA, 2017. ACM.

[101] T Hastie. Tibshirani, r. and friedman, j.(2009): The elements of statistical learning. data mining, inference, and prediction, 2008.

[102] Z. Hau and E. 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.

[103] K. Haynes, I. A Eckley, and P. Fearnhead. Efficient penalty search for multiple changepoint problems. *arXiv preprint arXiv:1412.3617*, 2014.

[104] Tobias Heer, Oscar Garcia-Morchon, René Hummen, Sye Loong Keoh, Sandeep S Kumar, and Klaus Wehrle. Security challenges in the ip-based internet of things. *Wireless Personal Communications*, 61(3):527–542, 2011.

[105] S. Hilton. Dyn analysis summary of friday october 21 attack. [online: last accessed in oct. 2017]. http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, October 2016.

[106] G. E. Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009. revision #91189.

[107] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[108] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2012.

[109] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[110] J. Hoffmann, S. Neumann, and T. Holz. Mobile malware detection based on energy fingerprints—a dead end? In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2013.

[111] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Enhancing security and privacy in traffic-monitoring systems. *IEEE Pervasive Computing*, 5(4):38–46, 2006.

[112] 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.

[113] J. Horsch. *Leveraging Logical and Physical Separation for Mobile Security*. Dissertation, Technische Universität München, München, 2019.

[114] Badr Hssina, Abdelkarim Merbouha, Hanane Ezzikouri, and Mohammed Erritali. A comparative study of decision tree id3 and c4. 5. *International Journal of Advanced Computer Science and Applications*, 4(2):13–19, 2014.

[115] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *Twenty-fifth international symposium on fault-tolerant computing. Digest of papers*, pages 381–390. IEEE, 1995.

[116] M. Hung. Leading the iot: Gartner insights on how to lead in a connected world, Feb. 2017.

[117] M. Huzaifah. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *arXiv preprint arXiv:1706.07156*, 2017.

[118] Google Inc. Nest the learning thermostat. https://nest.com/ca/, 2017.

[119] 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.

[120] Grant A Jacoby, Randy Marchany, and NathanielJ Davis. Battery-based intrusion detection a first line of defense. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 272–279. IEEE, 2004.

[121] R. James, A. Albasir, K. Naik, M. Dabbagh, P. Dash, M. Zaman, and N. Goel. Detection of unknown applications in smartphones: A signal processing perspective. In *Canadian Conference on Electrical and Computer Engineering*, pages 1–6, 2017.

[122] R. James, A. Albasir, K. Naik, and et. al. 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)*, 2017.

[123] R. James, A. Albasir, K. 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.

[124] X. Jiang, X. Wang, and D. Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*, 2007.

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

[126] S. Jin, Z. Zhang, K. Chakrabarty, and X. Gu. Anomaly detection and health-status analysis in a. *IEEE Design Test*, pages 1–1, 2019.

[127] Q. Jing, A. Vasilakos, J. Wan, J. Lu, and D. Qiu. Security of the internet of things: Perspectives and challenges. *Wireless Networks*, 20(8):2481–2501, 2014.

[128] Yohei Kawaguchi and Takashi Endo. How can we detect anomalies from subsampled audio signals? In *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2017.

[129] B. Khalfi, B. Hamdaoui, and M. Guizani. Extracting and exploiting inherent sparsity for efficient iot support in 5g: Challenges and potential solutions. *IEEE Wireless Communications*, 24(5):68–73, October 2017.

[130] H. Khan, N. Sehatbakhsh, L. Nguyen, M. Prvulovic, and A. Zajić. Malware detection in embedded systems using neural network model for electromagnetic side-channel signals. *Journal of Hardware and Systems Security*, pages 1–14, 2019.

[131] H. A. Khan, N. Sehatbakhsh, and et. al. Idea: Intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2019.

[132] Rafiullah Khan, Peter Maynard, Kieran McLaughlin, David Laverty, and Sakir Sezer. Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid. In *4th International Symposium for ICS & SCADA Cyber Security Research 2016 4*, pages 53–63, 2016.

[133] 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.

[134] H. Kim, J. Smith, and K. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the 6th int. conference on Mobile systems, applications, and services*. ACM, 2008.

[135] T. Kim, B. Kang, M. 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.

[136] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, 2014.

[137] D. Klonoff. Cybersecurity for connected diabetes devices. *Journal of diabetes science and technology*, 9(5), 2015.

[138] 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.

[139] S. Kumar, A. Viinikainen, and T. Hamalainen. Machine learning classification model for network based intrusion detection system. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 242–249, Dec 2016.

[140] H. Kurniawan, Y. Rosmansyah, and B. Dabarsyah. Android anomaly detection system using machine learning classification. In *Int. Conf. on Electrical Engineering and Informatics (ICEEI)*. IEEE, 2015.

[141] M. La Polla, F. Martinelli, and D. Sgandurra. A survey on security for mobile devices. *IEEE Communications Surveys Tutorials*, 15(1):446–471, First 2013.

[142] 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.

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

[144] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[145] T. Lee, C. Wen, L. Chang, H. Chiang, and M. Hsieh. A lightweight intrusion detection scheme based on energy consumption analysis in 6lowpan. In *Advanced Technologies, Embedded and Multimedia for Human-centric Computing*, pages 1205–1213. Springer, 2014.

[146] 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.

[147] Y. Lin, Y. Lin, J. Lin, and H. Hung. Sensortalk: An iot device failure detection and calibration mechanism for smart farming. *Sensors*, 19(21), 2019.

[148] I. Linkov and A. Kott. Fundamental concepts of cyber resilience: Introduction and overview. In *Cyber resilience of systems and networks*, pages 1–25. Springer, 2019.

[149] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[150] Lei Liu, Guanhua Yan, Xinwen Zhang, and Songqing Chen. Virusmeter: Preventing your cellphone from spies. In *RAID*, volume 5758, pages 244–264. Springer, 2009.

[151] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. 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.

[152] V. Lou. Application behavior based malware detection, August 17 2010. US Patent 7,779,472.

[153] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon, and D. Gan. Cloud-based cyber-physical intrusion detection for vehicles using deep learning. *Ieee Access*, 6:3491–3508, 2017.

[154] James M. The internet of things: Mapping the value beyond the hype. Technical report, McKinsey Global Institute, 2015.

[155] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[156] R. Manzano, A. Albasir, K. Naik, and et al. Detection of anomalous behavior in wireless devices using changepoint analysis. In *2019 IEEE International Congress on Internet of Things (ICIOT)*. IEEE, 2019.

[157] Erik Marchi, Fabio Vesperini, Felix Weninger, Florian Eyben, Stefano Squartini, and Björn Schuller. Non-linear prediction with lstm recurrent neural networks for acoustic novelty detection. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2015.

[158] E. D. Martin, J. Kargaard, and I. Sutherland. Raspberry pi malware: An analysis of cyberattacks towards iot devices. In *2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pages 161–166, June 2019.

[159] F. Martinelli, F. Mercaldo, and A. 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.

[160] V. Mashkov, J. Barilla, and P. Simr. Applying petri nets to modeling of many-core processor self-testing when tests are performed randomly. *Journal of Electronic Testing*, 29(1):25–34, 2013.

[161] M Massa, E Eva, D Spallarossa, and C Eva. Detection of earthquake clusters on the basis of waveform similarity: An application in the monferrato region (piedmont, italy). *Journal of seismology*, 10(1):1–22, 2006.

[162] D. Meng and C. WU. Security architecture and key technologies for iot/cps. *ZTE technology journal*, 1(1), 2011.

[163] A. Merlo, M. Migliardi, and P. Fontanelli. Measuring and estimating power consumption in android to support energy-based intrusion detection. *Journal of Computer Security*, 23(5):611–637, 2015.

[164] J. Milosevic, M. Malek, and A. Ferrante. A friend or a foe? detecting malware using memory and cpu features. In *Proceedings of the 13th Int. Joint Conf. on e-Business and Telecommunications*. SCITEPRESS-Science and Technology Publications, Lda, 2016.

[165] Thomas P Minka. Automatic choice of dimensionality for pca. In *Advances in neural information processing systems*, pages 598–604, 2001.

[166] R. Mitchell and I. Chen. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. *IEEE Transactions on Dependable and Secure Computing*, 12(1):16–30, Jan 2015.

[167] Robert Mitchell and Ray Chen. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. *IEEE Transactions on Dependable and Secure Computing*, 12(1):16–30, 2014.

[168] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE transactions on audio, speech, and language processing*, 20(1):14–22, 2011.

[169] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 421–430, Dec 2007.

[170] A. Mukhija. *CASA-a framework for dynamically adaptive applications*. PhD thesis, University of Zurich, 2007.

[171] D. Nash, T. Martin, D. Ha, and M. Hsiao. Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 141–145. IEEE, 2005.

[172] 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. IEEE, 2017.

[173] Zainib Noshad, Nadeem Javaid, Tanzila Saba, Zahid Wadud, Muhammad Qaiser Saleem, Mohammad Eid Alzahrani, and Osama E Sheta. Fault detection in wireless sensor networks through the random forest classifier. *Sensors*, 19(7):1568, 2019.

[174] H. Olufowobi, U. Ezeobi, E. Muhati, G. Robinson, C. Young, J. Zambreno, and G. 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.

[175] Naman Kamleshbhai Patel, Prashanth Krishnamurthy, Hussam Amrouch, Jörg Henkel, Michael Shamouilian, Ramesh Karri, and Farshad Khorrami. Towards a new thermal monitoring based framework for embedded cps device security. *IEEE Transactions on Dependable and Secure Computing*, 2020.

[176] F. Pedregosa and et. al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[177] Daniel Peña and Francisco J Prieto. Multivariate outlier detection and robust covariance matrix estimation. *Technometrics*, 43(3):286–310, 2001.

[178] Wei Peng, Juhua Chen, and Haiping Zhou. An implementation of id3-decision tree learning algorithm. *From web. arch. usyd. edu. au/wpeng/DecisionTree2. pdf Retrieved date: May*, 13, 2009.

[179] Postscapes. Hardware comparison, 2018.

[180] Praetox Project. Low orbit ion cannon - an open source network stress tool. https://github.com/NewEraCracker/LOIC. Accessed: 2019-08-30.

[181] Pranav Rajpurkar, Awni Y Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y Ng. Cardiologist-level arrhythmia detection with convolutional neural networks. *arXiv preprint arXiv:1707.01836*, 2017.

[182] A. Raman, A. Zaks, J. Lee, and D. August. Parcae: a system for flexible parallel execution. *ACM SIGPLAN Notices*, 47(6):133–144, 2012.

[183] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015.

[184] P.P. Ray. A survey on internet of things architectures. *Journal of King Saud University - Computer and Information Sciences*, 2016.

[185] S. Raza, L. Wallgren, and T. Voigt. Svelte: Real-time intrusion detection in the internet of things. *Ad hoc networks*, 11(8):2661–2674, 2013.

[186] 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.

[187] 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.

[188] Ummer Sahib. Smart dubai: Sensing dubai smart city for smart environment management. In *Smart Environment for Smart Cities*, pages 437–489. Springer, 2020.

[189] J. Sahs and L. Khan. A machine learning approach to android malware detection. In *2012 European Intelligence and Security Informatics Conference*, pages 141–147, Aug 2012.

[190] Anam Sajid, Haider Abbas, and Kashif Saleem. Cloud-assisted iot-based scada systems security: A review of the state of the art and future challenges. *IEEE Access*, 4:1375–1384, 2016.

[191] Tara Salman and Raj Jain. Networking protocols and standards for internet of things. *Internet of Things and Data Analytics Handbook*, pages 215–238, 2015.

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

[193] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. *NIPS*, 12:582–588, 1999. Cited By :41.

[194] Padriag Scully. Top 10 iot applications in 2020, 2020.

[195] N. Sehatbakhsh, M. Alam, A. Nazari, A. Zajic, and M. Prvulovic. Syndrome: Spectral analysis for anomaly detection on medical iot and embedded devices. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 1–8, April 2018.

[196] N. Sehatbakhsh, A. Nazari, H. Khan, A. Zajic, and M. Prvulovic. Emma: Hardware/software attestation framework for embedded systems using electromagnetic

signals. In *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pages 983–995, New York, NY, USA, 2019. ACM.

[197] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 2012.

[198] Haidong Shao, Hongkai Jiang, Huiwei Zhao, and Fuan Wang. A novel deep autoencoder feature learning method for rotating machinery fault diagnosis. *Mechanical Systems and Signal Processing*, 95:187–204, 2017.

[199] V. Sharma, I. You, K. Yim, I. Chen, and J. Cho. Briot: Behavior rule specification-based misbehavior detection for iot-embedded cyber-physical systems. *IEEE Access*, 7:118556–118580, 2019.

[200] Z. Shi, H. Lin, L. Liu, R. Liu, and J. Han. Is cqt more suitable for monaural speech separation than stft? an empirical study, 2019.

[201] Anastasia Shuba, Evita Bakopoulou, Milad Asgari Mehrabadi, Hieu Le, David Choffnes, and Athina Markopoulou. Antshield: On-device detection of personal information exposure. *arXiv preprint arXiv:1803.01261*, 2018.

[202] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146–164, 2015.

[203] E. Simmon, S. K. Sowe, and K. Zettsu. Designing a cyber-physical cloud computing architecture. *IT Professional*, 17(3):40–45, May 2015.

[204] Lindsay I Smith. A tutorial on principal components analysis. Technical report, 2002.

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

[206] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. Technical report, STATISTICS AND COMPUTING, 2003.

[207] Monsoon Solutions. Monsoon power monitor. = https://www.msoon.com/.

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

[209] Statista. Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions).

[210] Statista. Number of mobile phone users worldwide from 2015 to 2020 (in billions).

[211] J. Sterbenz, D. Hutchison, E. Çetinkaya, A. Jabbar, J. Rohrer, M. Schöller, and P. Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245–1265, 2010.

[212] D. Stiawan, M. Idris, and et. al. Anomaly detection and monitoring in internet of things communication. In *Information Technology and Electrical Engineering (ICITEE), 2016 8th International Conference on*. IEEE, 2016.

[213] F.t Stroud. Cryptomining malware. Accessed: 2019-10-30.

[214] R. Sun, X. Yuan, and et al. Leveraging uncertainty for effective malware mitigation. *arXiv preprint arXiv:1802.02503*, 2018.

[215] Takaaki Tagawa, Yukihiro Tadokoro, and Takehisa Yairi. Structured denoising autoencoder for fault detection and analysis. In *Asian Conference on Machine Learning*, pages 96–111, 2015.

[216] A. G. Tartakovsky, A. S. Polunchenko, and G. Sokolov. Efficient computer network anomaly detection by changepoint detection methods. *IEEE J of Selected Topics in Signal Processing*, 7(1), 2013.

[217] G. Taylor. *Composable, distributed-state models for high-dimensional time series*. University of Toronto Toronto, 2009.

[218] Kleanthis Thramboulidis, Danai C Vachtsevanou, and Alexandros Solanos. Cyber-physical microservices: An iot-based framework for manufacturing systems. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 232–239. IEEE, 2018.

[219] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi. Malware detection with deep neural network using process behavior. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 577–582, June 2016.

[220] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[221] E. Tragos, A. Fragkiadakis, V. Angelakis, and H. Pöhls. Designing secure iot architectures for smart city applications. In *Designing, Developing, and Facilitating Smart Cities*, pages 63–87. Springer, 2017.

[222] JW Tukey. Exploratory data analysis. addison-wesley, reading, ma. *Exploratory data analysis. Addison-Wesley, Reading, MA.*, 1977.

[223] M. Usman, M. Asghar, I. Ansari, and M. Qaraqe. Trust-based dos mitigation technique for medical implants in wireless body area networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[224] L. Utkin, V. Zaborovskii, and S. Popov. Detection of anomalous behavior in a robot system based on deep learning elements. *Automatic Control and Computer Sciences*, 50(8), 2016.

[225] T. Vuong, G. Loukas, and D. Gan. Performance evaluation of cyber-physical intrusion detection on a robotic vehicle. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 2106–2113. IEEE, 2015.

[226] Shijia Wei, Aydin Aysu, Michael Orshansky, Andreas Gerstlauer, and Mohit Tiwari. Using power-anomalies to counter evasive micro-architectural attacks in embedded systems. In *HOST*, pages 111–120, 2019.

[227] 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.

[228] I. Witten, E. Frank, M. Hall, and C. Pal. *Data Mining: Practical ML tools and techniques*. Morgan Kaufmann, 2016.

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

[230] 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.

[231] Hongyu Yang and Ruiwen Tang. Power consumption based android malware detection. *J. of Electrical and Computer Eng.*, 2016.

[232] 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.

[233] D. Yao, X. Shu, L. Cheng, and S. Stolfo. *Anomaly detection as a service : challenges, advances, and opportunities*. [San Rafael, California] Morgan Claypool Publishers, 2018.

[234] Q. Yaseen, F. AlBalas, Y. Jararweh, and M. Al-Ayyoub. A fog computing based system for selective forwarding detection in mobile wireless sensor networks. In *2016*

IEEE 1st International Workshops on Foundations and Applications of Self Systems (FAS* W)*, pages 256–262. IEEE, 2016.

[235] Y. Yen and H. Sun. An android mutation malware detection based on deep learning using visualization of importance from codes. *Microelectronics Reliability*, 93:109–114, 2019.

[236] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik. A new android malware detection approach using bayesian classification. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pages 121–128, March 2013.

[237] 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.

[238] I. You and K. Yim. Malware obfuscation techniques: A brief survey. In *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 297–300, Nov 2010.

[239] 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.

[240] 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.

[241] T. Zefferer, P. Teufl, D. Derler, K. Potzmader, A. Oprisnik, H. Gasparitz, and A. Höller. Towards secure mobile computing: Employing power-consumption information to detect malware on mobile devices. *Int. journal on advances in software*, 7, 2014.

[242] Haibin Zhang, Jiajia Liu, and Nei Kato. Threshold tuning-based wearable sensor fault detection for reliable medical monitoring using bayesian network model. *IEEE Systems Journal*, 2016.

[243] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*. IEEE, 2010.

[244] G. Zheng, R. Shankaran, M. A. Orgun, L. Qiao, and K. Saleem. Ideas and challenges for securing wireless implantable medical devices: A review. *IEEE Sensors Journal*, 17(3):562–576, Feb 2017.

[245] 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.

[246] D. Zubrow. Ieee standard classification for software anomalies. *IEEE Computer Society*, 2009.

Appendix

Appendix A

A.1 IoT Based Experiments Code

ultrasonic.c:

```
/*
 * ultrasonic.c
 * HC-SR04 ultrasonic sensor logging program for the Raspberry Pi
 *
 * Usage: ./ultrasonic
 *
 * Polls the ultrasonic sensor at a fixed frequency (indicated by
 * SAMPLE_INTERVAL_MS). The measurements are written to standard output.
 * */

#include <wiringPi.h>
#include <stdio.h>

// GPIO pin assignments
#define GPIO_TRIG1 4
#define GPIO_ECHO1 5

#define SPEED_OF_SOUND_MKS 343 // metres per second
```

```
#define SAMPLE_INTERVAL_MS 500 // milliseconds

int main(void) {
    wiringPiSetupGpio();
    pinMode(GPIO_TRIG1, OUTPUT);
    pinMode(GPIO_ECHO1, INPUT);

    unsigned last_measurement_ms = millis();

    while(1) {
        while(millis() - last_measurement_ms < SAMPLE_INTERVAL_MS) {
            delayMicroseconds(200);
        }
        last_measurement_ms = millis();

        // Send a pulse to the ultrasonic sensor
        digitalWrite(GPIO_TRIG1, HIGH);
        delayMicroseconds(10);
        digitalWrite(GPIO_TRIG1, LOW);

        // Wait for the start of the echo pulse
        while(digitalRead(GPIO_ECHO1) != HIGH) { /* wait */ }
        unsigned int t1_us = micros();

        // Wait for the end of the echo pulse
        while(digitalRead(GPIO_ECHO1) == HIGH) { /* wait */ }
        unsigned int t2_us = micros();

        // Compute the length of the echo pulse in seconds
        unsigned int delta_t_us = t2_us - t1_us;
        double delta_t_s = ((double) delta_t_us) / 1000000.0;

        // Compute the round-trip distance and divide by 2 to get the
        // distance to the object
```

```
        double dist_rt_metres = delta_t_s * SPEED_OF_SOUND_MKS;
        double dist_metres = dist_rt_metres / 2.0;
        printf("%f\n", dist_metres);
    }
    return 0;
}
stresstest.c:
/*
 * stresstest.c
 * CPU stress tester for Linux systems
 *
 * Usage: ./stresstest [--infinite | -I] [-n <NUM_CYCLES>]
 *     [(--duty-cycle | -d) <DUTY_CYCLE>] [--crypto | -c]
 *
 * Runs busyloops on all cores.
 *
 * Options:
 *
 *     --infinite, -I: run infinite busyloops.
 *
 *     -n <NUM_CYCLES>: run NUM_CYCLES loops divided evenly across all
 *         cores. (i.e. if N cores are being stress-tested, each core
 *         will run NUM_CYCLES / N loops.
 *
 *     --duty-cycle <DUTY_CYCLE>, -d <DUTY_CYCLE>: runs busyloops with
 *         a duty cycle of DUTY_CYCLE (i.e. actively run for DUTY_CYCLE
 *         of the time), split evenly between all cores.
 *
 *     --crypto, -c: replace the busyloop operation with cryptographic
 *         computations (to simulate a real workload more closely).
 *
 *     --math, -m: replace the busyloop operation with mathematical
 *         computations (calculating the sine of random values) to
 *         simulate a real workload more closely.
 *
```

```
 * Compile with:
 * gcc -lpthread -lgcrypt -std=gnu11 stresstest.c -o stresstest
 * */

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <math.h>
#include <errno.h>
#include <sys/sysinfo.h>
#include <pthread.h>
#include <gcrypt.h>
#include <inttypes.h>

// 12 billion cycles take about 1 minute to perform on a 4-core
// Raspberry Pi Model 3B.
#define DEFAULT_NUM_CYCLES 12000000000

// 120 million cycles take about 50 seconds to perform on a 4-core
// Raspberry Pi Model 3B with a duty cycle of 0.8.
#define DUTY_CYCLE_DEFAULT_NUM_CYCLES 120000000

// 180 million cycles take about 50 seconds to perform on a 4-core
// Raspberry Pi Model 3B.
#define CRYPTO_DEFAULT_NUM_CYCLES 180000000

// 600 million cycles take about 50 seconds to perform on a 4-core
// Raspberry Pi Model 3B.
#define MATH_DEFAULT_NUM_CYCLES 600000000

// The total length of one cycle of the variable-duty-cycle loop
// (one active and one inactive phase).
#define PERIOD_LENGTH_NS 100000000

// The length of messages to be encrypted and decrypted.
```

```
#define MESSAGE_LENGTH 16

// The maximum allowable error in floating-point computations.
#define ERR_THRESHOLD 1e-10

// Stores all the parameters that need to be given to the busy-loop
// function -- the number of cycles and the duty cycle -- as a single
// struct for passing to pthread_create().
typedef struct {
    uint64_t num_cycles;
    double duty_cycle;
} thread_params_t;

// Helper functions

// Returns the number of nanoseconds since the Unix Epoch.
// Will overflow on July 21, 2554.
uint64_t nanos() {
    struct timespec spec;
    timespec_get(&spec, TIME_UTC);
    return 1000000000ull * (uint64_t) spec.tv_sec
        + (uint64_t) spec.tv_nsec;
}

// Busyloop functions

// Runs an endless busy-loop.
// This function's type signature is mandated by the pthread library.
// Neither the parameter nor the return value are ever used.
void* infinite_busyloop(__attribute__((unused)) void *ignored) {
    volatile int x = 0;
    while(1) ++x;
    return NULL;
}
```

```
// Runs a busy-loop for a given number of cycles.
// This function's type signature is mandated by the pthread library.
// The return value is never used.
void* busyloop(void *params_ptr) {
    thread_params_t *params = (thread_params_t*) params_ptr;
    uint64_t num_cycles = params->num_cycles;

    volatile uint64_t x = 0;
    while(x < num_cycles) ++x;
    return NULL;
}

// Runs a partial busy loop with a given duty cycle for a given number
// of cycles.
// This function's type signature is mandated by the pthread library.
// The return value is never used.
void* duty_cycle_busyloop(void *thread_params_ptr) {
    thread_params_t params
        = *((thread_params_t*) thread_params_ptr);

    uint64_t num_cycles = params.num_cycles;
    double duty_cycle = params.duty_cycle;

    uint64_t active_time_ns = (uint64_t)
        (duty_cycle * PERIOD_LENGTH_NS);
    uint64_t inactive_time_ns = PERIOD_LENGTH_NS - active_time_ns;

    volatile uint64_t x = 0;
    while(x < num_cycles) {
        // Active phase: run a busy-loop until active_time_ns
        // nanoseconds have elapsed.
        uint64_t cycle_start = nanos();
        while(nanos() - cycle_start < active_time_ns) {
            for(int i = 0; i < 1000000; ++i) {
                ++x;
```

```
            }
        }

        // Inactive phase: sleep for inactive_time_ns nanoseconds.
        struct timespec sleep_spec;
        sleep_spec.tv_sec = 0;
        sleep_spec.tv_nsec = (long) inactive_time_ns;
        nanosleep(&sleep_spec, NULL);
    }
    return NULL;
}

// Performs a busy-loop that performs cryptographic operations for a
// given number of cycles.
// This function's type signature is mandated by the pthread library.
// The return value is never used.
void* crypto_busyloop(void *params_ptr) {
    thread_params_t *params = (thread_params_t*) params_ptr;
    uint64_t num_cycles = params->num_cycles;

    char *plaintext = calloc(MESSAGE_LENGTH, sizeof(char));
    char *output = calloc(MESSAGE_LENGTH, sizeof(char));

    gcry_cipher_hd_t cipher;
    gcry_error_t err = gcry_cipher_open(&cipher, GCRY_CIPHER_AES,
        GCRY_CIPHER_MODE_ECB, 0 /* no special options */);
    if(err) {
        fprintf(stderr, "Error in libgcrypt\n");
        exit(1);
    }

    char *key = "3141592653897932";
    gcry_cipher_setkey(cipher, key,
        gcry_cipher_get_algo_keylen(GCRY_CIPHER_AES128));
```

```
    for(uint64_t i = 0; i < num_cycles; ++i) {
        strcpy(plaintext, "123456789ABCDEF");

        gcry_cipher_encrypt(cipher, output, MESSAGE_LENGTH, plaintext,
            MESSAGE_LENGTH);
        gcry_cipher_decrypt(cipher, output, MESSAGE_LENGTH, NULL, 0);

        if(memcmp(plaintext, output, MESSAGE_LENGTH) != 0) {
            fprintf(stderr, "Decryption failed!\n");
            exit(1);
        }
    }

    free(plaintext);
    free(output);
    gcry_cipher_close(cipher);
    return NULL;
}

// Performs a busy-loop that calculates the sine of arbitrary values.
// This function's type signature is mandated by the pthread library.
// The return value is never used.
void* math_busyloop(void *params_ptr) {
    thread_params_t *params = (thread_params_t*) params_ptr;
    uint64_t num_cycles = params->num_cycles;

    for(uint64_t i = 0; i < num_cycles; ++i) {
        // The value whose sine we are computing
        double val = ((double) i) / (double) num_cycles * 2 * M_PI
            - M_PI;
        // The sine of val
        volatile double res = val;
        // The current term in the Maclaurin series
        double term = val;
        // The power of x in the current term
```

```
        int power = 1;
        do {
            power += 2;
            // Every other term is of opposite sign
            term *= -1;
            // The power on 'x' increases by 2 each term
            term *= val * val;
            // The denominator goes from (power-2)! to power!
            term /= ((power - 1) * power);
            res += term;
        } while(fabs(term) > ERR_THRESHOLD);
    }
    return NULL;
}

int main(int argc, char **argv) {
    // Parse command-line arguments
    int infinite = 0;
    uint64_t num_cycles = 0;
    int crypto = 0;
    int use_partial_duty_cycle = 0;
    double duty_cycle = 0;
    int math = 0;

    // Start looping at 1 (argv[0] is the program name)
    for(int i = 1; i < argc; ++i) {
        char *arg = argv[i];
        if(strcmp(arg, "--infinite") == 0 || strcmp(arg, "-I") == 0) {
            infinite = 1;
        }
        // In the next two options, the check against argc makes sure
        // that there is at least one argument after the flag (since
        // these two options require an argument).
        else if(strcmp(arg, "-n") == 0 && i + 1 < argc) {
            ++i;
```

```
            arg = argv[i];
            num_cycles = strtoull(arg, NULL, /* base */ 10);
            // Check for invalid input
            if(num_cycles == 0 || errno == ERANGE) {
                fprintf(stderr, "Invalid parameter to --n: %s\n", arg);
                fprintf(stderr, "(Expected a positive integer)\n");
                exit(2);
            }
        }
        else if((strcmp(arg, "--duty-cycle") == 0
            || strcmp(arg, "-d") == 0)
            && i + 1 < argc) {
            ++i;
            arg = argv[i];
            use_partial_duty_cycle = 1;
            duty_cycle = strtod(arg, NULL);
            // Check for invalid input
            if(duty_cycle <= 0 || duty_cycle > 1.0 || errno == ERANGE) {
                fprintf(stderr, "Invalid parameter to --duty-cycle: "
                    "%s\n", arg);
                fprintf(stderr, "(Expected a real number between "
                    "0 and 1)\n");
                exit(2);
            }
        }
        else if(strcmp(arg, "-c") == 0
            || strcmp(arg, "--crypto") == 0) {
            crypto = 1;
        }
        else if(strcmp(arg, "-m") == 0
            || strcmp(arg, "--math") == 0) {
            math = 1;
        }
        else {
            // Either this argument wasn't one of the specified options,
```

```
            // or an option needed a parameter but wasn't given one.
            fprintf(stderr, "Invalid option: %s\n", arg);
            exit(2);
        }
    }

    // If the user has not supplied the number of cycles, pick a default
    // value based on the mode used.
    if(num_cycles == 0) {
        if(use_partial_duty_cycle) {
            num_cycles = DUTY_CYCLE_DEFAULT_NUM_CYCLES;
        }
        else if(crypto) {
            num_cycles = CRYPTO_DEFAULT_NUM_CYCLES;
        }
        else if(math) {
            num_cycles = MATH_DEFAULT_NUM_CYCLES;
        }
        else {
            num_cycles = DEFAULT_NUM_CYCLES;
        }
    }

    // Determine the number of threads to start.
    // get_nprocs() returns the number of processors in the system,
    // which is used as the number of threads.
    unsigned num_threads = (unsigned) get_nprocs();
    uint64_t cycles_per_thread = num_cycles / num_threads;

    // Allocate the thread pointer array
    pthread_t *threads = (pthread_t*) malloc(
        num_threads * sizeof(pthread_t));
    if(threads == NULL) {
        fprintf(stderr, "Out of memory!");
        exit(1);
```

```
}

// Determine which function and parameters to use
void *(*loop_func)(void*);
thread_params_t params;

if(infinite) {
    loop_func = infinite_busyloop;
    // Don't set any of params' fields; the function ignores them.
}
else if(use_partial_duty_cycle) {
    loop_func = duty_cycle_busyloop;
    params.num_cycles = cycles_per_thread;
    params.duty_cycle = duty_cycle / num_threads;
}
else if(crypto) {
    loop_func = crypto_busyloop;
    params.num_cycles = cycles_per_thread;
    // Don't set params.duty_cycle; the function ignores it.

    // Initialize gcrypt
    if(!gcry_check_version(GCRYPT_VERSION)) {
        fprintf(stderr, "Failed to initialize libgcrypt\n");
        exit(1);
    }
    gcry_control(GCRYCTL_DISABLE_SECMEM, 0);
    gcry_control(GCRYCTL_INITIALIZATION_FINISHED, 0);
}
else if(math) {
    loop_func = math_busyloop;
    params.num_cycles = cycles_per_thread;
    // Don't set params.duty_cycle; the function ignores it.
}
else {
    loop_func = busyloop;
```

```
        params.num_cycles = cycles_per_thread;
        // Don't set params.duty_cycle; the function ignores it.
    }

    // Create threads
    for(unsigned i = 0; i < num_threads; ++i) {
        int result_code = pthread_create(&threads[i], NULL,
            loop_func, &params);
        if(result_code != 0) {
            fprintf(stderr, "Failed to create thread!\n");
            free(threads);
            exit(1);
        }
    }

    // Wait for all threads to finish
    for(unsigned i = 0; i < num_threads; ++i) {
        pthread_join(threads[i], NULL);
    }

    free(threads);
    return 0;
}
```

miner.c

```
#include <stdlib.h>
#include <time.h>
#include <stdint.h>
#include <stdio.h>
#include <gcrypt.h>
#include <inttypes.h>

#define DEFAULT_DUTY_CYCLE 0.01

// The total length of a cycle, consisting of one active and one
```

```
// inactive period.
#define PERIOD_NS 10000000000ull // 10 s

// The minimum number of zero bytes at the beginning of the digest for
// the input to count as valid "Proof of Work".
#define MIN_ZERO_BYTES 3

#define BUFFER_SIZE 128

// Returns the number of nanoseconds since the Unix Epoch.
// Will overflow on July 21, 2554.
uint64_t nanos() {
    struct timespec spec;
    timespec_get(&spec, TIME_UTC);
    return 1000000000ull * (uint64_t) spec.tv_sec
        + (uint64_t) spec.tv_nsec;
}

// Check if the SHA256 hash of a given buffer starts with MIN_ZERO_BYTES
// bytes of zeroes. This is identical to Bitcoin's proof-of-work
// protocol except for the number of zero bytes.
void mine(char *buffer) {
    char digest[32];
    gcry_md_hash_buffer(GCRY_MD_SHA256, digest, buffer, BUFFER_SIZE);
    int valid = 1;
    for(int i = 0; i < MIN_ZERO_BYTES; ++i) {
        if(digest[i] != 0) {
            valid = 0;
        }
    }
    if(valid) {
        /*
        printf("Found:\n");
        for(int i = 0; i < BUFFER_SIZE; ++i) {
            printf("%.2hhx", buffer[i]);
```

```
        }
        printf("\n");
        for(int i = 0; i < 32; ++i) {
            printf("%.2hhx", digest[i]);
        }
        printf("\n");
        */
    }
}

// Increment the given buffer, handling byte overflow.
void next(char *buffer) {
    // Start by incrementing the last byte
    int curr_pos = BUFFER_SIZE - 1;
    while(curr_pos >= 0) {
        ++buffer[curr_pos];
        // Check if we need to carry a bit
        if(buffer[curr_pos] == 0) {
            --curr_pos;
        }
        else break;
    }
}

int main(int argc, char **argv) {
    double duty_cycle;
    if(argc >= 2) {
        // argv[0] is the executable name; argv[1] is the first argument
        duty_cycle = strtod(argv[1], NULL);
        // Check if the parameter was invalid
        if(duty_cycle == 0.0) {
            duty_cycle = DEFAULT_DUTY_CYCLE;
        }
    }
    else {
```

```
    duty_cycle = DEFAULT_DUTY_CYCLE;
}

// Initialize gcrypt
if(!gcry_check_version(GCRYPT_VERSION)) {
    fprintf(stderr, "Failed to initialize libgcrypt\n");
    exit(1);
}
gcry_control(GCRYCTL_DISABLE_SECMEM, 0);
gcry_control(GCRYCTL_INITIALIZATION_FINISHED, 0);

uint64_t active_time_ns = (uint64_t) (duty_cycle * PERIOD_NS);
uint64_t inactive_time_ns = PERIOD_NS - active_time_ns;

// Start testing values at a random point
char *buffer = calloc(BUFFER_SIZE, sizeof(char));
srand((unsigned) time(NULL));
for(int i = 0; i < BUFFER_SIZE; ++i) {
    buffer[i] = (char) rand() % 256;
}

while(1) {
    uint64_t start_time = nanos();
    while(nanos() - start_time < active_time_ns) {
        mine(buffer);
        next(buffer);
    }

    struct timespec sleep_spec;
    sleep_spec.tv_sec = (time_t) (inactive_time_ns / 1000000000ll);
    sleep_spec.tv_nsec = (int32_t) (inactive_time_ns % 1000000000ll);
    nanosleep(&sleep_spec, NULL);
}

free(buffer);
```

```
}

videotocsv.py:
""" Converts a video taken by the Seekshot Pro into a CSV of thermal data.
Each row represents one frame in the video, and each column represents a pixel
(in row-major order).
The input and output filenames are given as command-line arguments.
"""

import sys
import os.path as path
import numpy as np
import cv2

# The minimum and maximum temperatures for all the images' colour scales.
# All temperature values throughout this program are in degrees Celsius.
MIN_TEMP = 20
MAX_TEMP = 70

def linear_map(in_min, in_max, out_min, out_max, datum):
    """ Rescales a value from [in_min, in_max] to [out_min, out_max].
    """
    return out_min + (datum - in_min) * (out_max - out_min) / (in_max - in_min)

def image_to_temperatures(array):
    """ Converts a thermal image frame into an array of temperature values.
    The input array should be a 1D Numpy array corresponding to a greyscale
    image in row-major order.
    The output array is of the same format, with each entry corresponding to a
    temperature value.
    """
    # Extract the minimum and maximum brightness values from the colour scale
    # legend (located at X~=610, Y=409 to Y=70)
    #img_array = np.array(img)
    #minimum = img_array[409, 610] # Y before X
```

```
    #maximum = img_array[70, 610]
    minimum = 16
    maximum = 235 # These are the same for all images

    # As it turns out, the colour scale used by the camera is (almost) linear,
    # and any apparent nonlinearity could simply be a result of lossy JPEG
    # compression.
    # Therefore, the use of a linear mapping is justified.

    # Call linear_map on the whole array at once, using NumPy broadcasting.
    # This is three orders of magnitude faster than using list comprehension.
    temperature_array = linear_map(minimum, maximum, MIN_TEMP, MAX_TEMP, \
        array)
    return temperature_array

def main():
    """Main function.
    """
    # Validate command-line arguments
    # Check the number of arguments provided
    if len(sys.argv) < 3:
        print("Specify the input and output filenames as arguments.")
        exit(2)

    # Check that the input file exists
    in_file = sys.argv[1]
    if not path.isfile(in_file):
        print("Error: file %s not found" % in_file)
        exit(2)

    # Check that the output file doesn't exist (to avoid accidentally
    #overwriting a file)
    out_file = sys.argv[2]
    if path.isfile(out_file):
        print("Error: file %s already exists" % out_file)
```

```
        exit(2)

    # An array to store all the created numpy arrays
    out_arrays = []

    # Split the video into frames
    video_capture = cv2.VideoCapture(in_file)
    success, image_cv = video_capture.read()

    while success:
        # Make the image greyscale
        image = cv2.cvtColor(image_cv, cv2.COLOR_BGR2GRAY)

        # Reshape the array to 1D (in row-major order)
        # ravel() is used instead of flatten() because ravel() avoids making a
        # copy of the data if possible.
        image = image.ravel()

        # Convert the image to an array of temperature data
        temperatures = image_to_temperatures(image)

        # Append the image to the array
        out_arrays.append(temperatures)

        # Get the next frame
        success, image_cv = video_capture.read()

    # Combine the arrays into one
    out_array = np.vstack(tuple(out_arrays))

    # Write the output array
    np.savetxt(out_file, out_array, delimiter=",", fmt="%.5g")

if __name__ == "__main__":
    main()
```

```
config.py:
# pylint: disable=import-error
# pylint: disable=unused-import
# pylint: disable=undefined-variable
import time

import supybot.utils as utils
from supybot.commands import *
import supybot.plugins as plugins
import supybot.ircutils as ircutils
import supybot.callbacks as callbacks
try:
    from supybot.i18n import PluginInternationalization
    _ = PluginInternationalization('LoicControl')
except ImportError:
    # Placeholder that allows to run the plugin on a bot
    # without the i18n module
    _ = lambda x: x

# The default options to use for LOIC.
LOIC_DEFAULT_OPTS = {
    'target_ip': '127.0.0.1',
    'target_url': 'localhost',
    'method': 'tcp',
    'port': '80',
    'message': 'test',
    'wait': 'false',
    'random': 'true'
}

# pylint: disable=unused-argument
class LoicControl(callbacks.Plugin):
    """Controls LOIC automatically"""
    threaded = True
```

```
    def __init__(self, irc):
        self.__parent = super(LoicControl, self)
        self.__parent.__init__(irc)

        # LOIC options
        self.loic_opts = {
            'target_ip': '127.0.0.1',
            'target_url': 'localhost',
            'method': 'tcp',
            'port': '80',
            'message': 'test',
            'wait': 'false',
            'random': 'true'
        }
        # Are we using an IP or URL as our target?
        self.loic_use_ip = True
        # Have LOIC's settings changed since we last started it?
        self.loic_needs_update = False

    def ip(self, irc, msg, args, ip):
        """<ip>

        Sets LOIC's target IP.
        """

        self.loic_opts['target_ip'] = str(ip)
        self.loic_needs_update = True
        self.loic_use_ip = True
        irc.reply("Set target IP to " + str(ip))
    ip = wrap(ip, ['ip'])

    def url(self, irc, msg, args, url):
        """<url>

        Sets LOIC's target URL.
```

```
        """

        self.loic_opts['target_url'] = str(url)
        self.loic_needs_update = True
        self.loic_use_ip = False
        irc.reply("Set target URL to " + str(url))
    url = wrap(url, ['url'])

    def method(self, irc, msg, args, method):
        """<method>

        Sets LOIC's method. Valid methods: tcp, udp, http, icmp.
        """

        self.loic_opts['method'] = method
        self.loic_needs_update = True
        irc.reply("Set method to " + method)
    method = wrap(method, [('literal', ('tcp', 'udp', 'http', 'icmp'))])

    def port(self, irc, msg, args, port):
        """<port>

        Sets LOIC's port.
        """

        self.loic_opts['port'] = str(port)
        self.loic_needs_update = True
        irc.reply("Set port to " + str(port))
    port = wrap(port, ['int'])

    def message(self, irc, msg, args, message):
        """<message>

        Sets LOIC's message. Must not contain spaces.
        """
```

```
        self.loic_opts['message'] = message
        self.loic_needs_update = True
        irc.reply("Set message to " + message)
    message = wrap(message, ['somethingWithoutSpaces'])

    def defaults(self, irc, msg, args):
        """<no args>

        Resets LOIC's settings to their defaults.
        """
        self.loic_opts = LOIC_DEFAULT_OPTS
        self.loic_needs_update = True
        irc.reply("Reset LOIC options to their defaults.")
    defaults = wrap(defaults, [])

    # pylint: disable=too-many-arguments
    def loic(self, irc, msg, args, num_times, active_time, wait_time):
        """<num_times> <active_time> <wait_time>

        Runs LOIC periodically <num_times> times in total.
        Each time, LOIC is activated for <active_time> seconds,
        and deactivated for <wait_time> seconds.
        """

        if self.loic_needs_update:
            # Build the config message to send to LOIC
            message = ['!lazor']
            if self.loic_use_ip:
                message.append('targetip=' + self.loic_opts['target_ip'])
            else:
                message.append('targethost=' + self.loic_opts['target_url'])
            message.append('message=' + self.loic_opts['message'])
            message.append('port=' + self.loic_opts['port'])
            message.append('method=' + self.loic_opts['method'])
```

```
            message.append('wait=' + self.loic_opts['wait'])
            message.append('random=' + self.loic_opts['random'])
            irc.reply(' '.join(message), prefixNick=False)
            self.loic_needs_update = False

        for _ in range(num_times):
            irc.reply("!lazor start", prefixNick=False)
            # Wait for active_time real-time seconds
            lazor_start_time = time.time()
            while time.time() - lazor_start_time < active_time:
                time.sleep(0.01)

            irc.reply("!lazor stop", prefixNick=False)
            # Wait for wait_time real-time seconds
            lazor_stop_time = time.time()
            while time.time() - lazor_stop_time < wait_time:
                time.sleep(0.01)

    loic = wrap(loic, ['int', 'int', 'int'])

Class = LoicControl


# vim:set shiftwidth=4 softtabstop=4 expandtab textwidth=79:
```

powertest.py:

```
"""Monsoon LVPM Raspberry Pi Power Consumption Test Automator

Automatically performs power consumption tests on a Raspberry Pi, controlling
it through SSH and measuring the results via a Monsoon Low Voltage Power
Monitor (LVPM).
"""

import os
import sys
```

```
import socket
import logging

import usb.core
import Monsoon.LVPM as lvpm
import Monsoon.sampleEngine as se
import Monsoon.Operations as op
import paramiko

# The number of trials to perform.
NUM_TRIALS = 3

# The Raspberry Pi's hostname and port
PI_HOSTNAME = "129.97.11.246"
SSH_PORT = 22

# Login credentials for the Raspberry Pi
PI_USERNAME = "pi"
PI_PASSWORD = "correct horse battery staple"

# The commands invoked to run programs on the Raspberry Pi.
ULTRASONIC_COMMAND = "timeout 300 ./ultrasonic >/dev/null"
GENERIC = 1
CRYPTO = 2
MATH = 3
STRESSTEST_COMMANDS = {
    GENERIC: "./stresstest",
    CRYPTO: "./stresstest --crypto",
    MATH: "./stresstest --math"
}
STRESSTEST_NAMES = {
    GENERIC: "generic",
    CRYPTO: "crypto",
    MATH: "math"
}
```

```
STRESSTEST_MODES = [GENERIC, CRYPTO, MATH]
MINER_COMMAND = "timeout 300 ./miner 0.12"

def setup_logger():
    """Sets up the logger."""
    logging.basicConfig(
        format='[%(asctime)-8s] %(name)-20s [%(levelname)-7s] %(message)-s',
        datefmt="%H:%M:%S",
        level=logging.INFO)

def assert_not_exists(folder_name):
    """Checks to make sure a folder doesn't exist, exiting the program if it
    does.
    """
    logger = logging.getLogger('assert_not_exists')
    if os.path.isdir(folder_name):
        logger.error("Folder %s is already present", folder_name)
        logger.error("Aborting to prevent overwriting data")
        exit(2)

def setup_lvpm():
    """Sets up the Monsoon Low Voltage Power Monitor.
    Returns the SampleEngine object.
    """
    logger = logging.getLogger(    setup_lvpm    )
    monitor = lvpm.Monsoon()
    try:
        monitor.setup_usb()
    except usb.core.USBError:
        logger.error("Error connecting to LVPM through USB:")
        logger.error("(Did you set up a udev rule giving your user access" \
            + " to the LVPM?)")
        exit(1)

    # We aren't using the main channel, but the LVPM docs say to enable it
```

```
    # anyway.
    monitor.setVout(4.0)

    # Create the sample engine
    sample_engine = se.SampleEngine(monitor)

    # Switch to the USB channel
    sample_engine.disableChannel(se.channels.MainVoltage)
    sample_engine.disableChannel(se.channels.MainCurrent)
    sample_engine.enableChannel(se.channels.USBVoltage)
    sample_engine.enableChannel(se.channels.USBCurrent)
    monitor.setUSBPassthroughMode(op.USB_Passthrough.On)

    # Configure triggers to record data for 300 seconds
    sample_engine.setStartTrigger(se.triggers.GREATER_THAN, 0)
    sample_engine.setStopTrigger(se.triggers.GREATER_THAN, 300)
    sample_engine.setTriggerChannel(se.channels.timeStamp)

    return sample_engine

def run_test(filename, sample_engine, ssh_client, commands):
    """Runs a test with the specified commands through the specified SSH client.
    Saves the results to the specified filename.
    """
    logger = logging.getLogger('run_test')
    sample_engine.ConsoleOutput(False)
    sample_engine.enableCSVOutput(filename)

    for command in commands:
        logger.info("Executing command: %s", command)
        ssh_client.exec_command(command)

    # This function blocks until sampling is finished
    sample_engine.startSampling(se.triggers.SAMPLECOUNT_INFINITE)
```

```
def␣irc_command(irc_socket,␣command):
␣␣␣␣"""Sends␣an␣IRC␣command␣to␣the␣given␣socket␣and␣logs␣the␣command."""
␣␣␣␣logger␣=␣logging.getLogger('irc')
␣␣␣␣logger.info("Sending␣IRC␣command:␣%s",␣command)
␣␣␣␣command_bytes␣=␣(command␣+␣"\r\n").encode("utf-8")
␣␣␣␣irc_socket.send(command_bytes)

def␣loic():
␣␣␣␣"""Starts␣LOIC␣through␣IRC,␣using␣Supybot␣to␣schedule␣start/stop␣commands.
␣␣␣␣"""
␣␣␣␣irc_socket␣=␣socket.socket(socket.AF_INET,␣socket.SOCK_STREAM)
␣␣␣␣irc_socket.connect(('localhost',␣6667))
␣␣␣␣irc_command(irc_socket,␣"NICK␣powertest")
␣␣␣␣irc_command(irc_socket,␣"USER␣powertest␣0␣*␣:Power␣Testing␣Script")
␣␣␣␣irc_command(irc_socket,␣"JOIN␣#loic")
␣␣␣␣irc_command(irc_socket,␣"PRIVMSG␣#loic␣:@ip␣%s"␣%␣PLHOSTNAME)
␣␣␣␣irc_command(irc_socket,␣"PRIVMSG␣#loic␣:@loic␣15␣10␣10")
␣␣␣␣irc_socket.close()

def␣main():
␣␣␣␣"""Main␣function."""

␣␣␣␣#␣Setup␣the␣logger
␣␣␣␣setup_logger()
␣␣␣␣logger␣=␣logging.getLogger('main')

␣␣␣␣#␣Check␣the␣number␣of␣command␣line␣args
␣␣␣␣if␣len(sys.argv)␣<␣2:
␣␣␣␣␣␣␣␣print("Specify␣the␣output␣folder␣as␣the␣first␣argument")
␣␣␣␣␣␣␣␣exit(2)

␣␣␣␣#␣Determine␣the␣output␣folder␣name
␣␣␣␣output_folder_name␣=␣sys.argv[1]
␣␣␣␣if␣not␣os.path.isdir(output_folder_name):
␣␣␣␣␣␣␣␣logger.info("Folder␣%s␣does␣not␣exist;␣creating",␣output_folder_name)
```

```
        os.mkdir(output_folder_name)

    # Determine which trials to run
    run_stresstests = "--stresstest" in sys.argv or "--all" in sys.argv
    run_miner_tests = "--miner" in sys.argv or "--all" in sys.argv
    run_dos_tests = "--dos" in sys.argv or "--all" in sys.argv

    # Initialize the Monsoon LVPM
    sample_engine = setup_lvpm()

    # Wait for the Raspberry Pi to start
    print("Press ENTER when the Raspberry Pi has finished booting")
    _ = input()

    # Connect to the Raspberry Pi through SSH
    client = paramiko.SSHClient()
    # Load the private key from the file ~/.ssh/id_rsa
    client.load_system_host_keys()
    client.set_missing_host_key_policy(paramiko.WarningPolicy())

    logger.info("Connecting to Raspberry Pi...")
    client.connect(PI_HOSTNAME, SSH_PORT, PI_USERNAME, PI_PASSWORD)
    logger.info("Connected!")

    if run_stresstests:
        stresstest_folder = output_folder_name + "/stresstests"
        assert_not_exists(stresstest_folder)
        os.mkdir(stresstest_folder)

        for trial in range(1, NUM_TRIALS + 1):
            for stresstest_mode in STRESSTEST_MODES:
                filename = ''.join([
                    stresstest_folder, "/", STRESSTEST_NAMES[stresstest_mode],
                    str(trial), ".csv"])
                run_test(
```

```
                    filename, sample_engine, client,
                    [STRESSTEST_COMMANDS[stresstest_mode]])

    if run_miner_tests:
        miner_test_folder = output_folder_name + "/minertests"
        assert_not_exists(miner_test_folder)
        os.mkdir(miner_test_folder)

        for trial in range(1, NUM_TRIALS + 1):
            # Run tests with and without the miner running
            filename = ''.join(
                [miner_test_folder, "/normal", str(trial), ".csv"])
            run_test(filename, sample_engine, client, [ULTRASONIC_COMMAND])
            filename = ''.join(
                [miner_test_folder, "/miner", str(trial), ".csv"])
            run_test(
                filename, sample_engine, client,
                [ULTRASONIC_COMMAND, MINER_COMMAND])

    if run_dos_tests:
        dos_test_folder = output_folder_name + "/dostests"
        assert_not_exists(dos_test_folder)
        os.mkdir(dos_test_folder)

        for trial in range(1, NUM_TRIALS + 1):
            filename = ''.join(
                [dos_test_folder, "/dos", str(trial), ".csv"])
            # Start LOIC before calling run_test(), because that function blocks
            # until the test is complete.
            loic()
            run_test(filename, sample_engine, client, [ULTRASONIC_COMMAND])

if __name__ == "__main__":
    main()
```