2019

Multimodal Approach for Malware Detection

Jarilyn M. Hernandez Jimenez
West Virginia University, jhernan7@mix.wvu.edu

Multimodal Approach for Malware Detection

Jarilyn Marie Hernández Jiménez

Dissertation submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

Katerina Goseva-Popstojanova, Ph.D., Chair
Vinod Kulathumani, Ph.D.
Roy S. Nutter, Ph.D.
Stacy Prowell, Ph.D.
Yanfang Ye, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2019

Keywords: malware detection, power consumption, network traffic data, system logs, code-based static data, multimodal learning, feature level fusion, decision level fusion

Abstract

Multimodal Approach for Malware Detection

Jarilyn Marie Hernández Jiménez

Although malware detection is a very active area of research, few works were focused on using physical properties (e.g., power consumption) and multimodal features for malware detection. We designed an experimental testbed that allowed us to run samples of malware and non-malicious software applications and to collect power consumption, network traffic, and system logs data, and subsequently to extract dynamic behavioral-based features. We also extracted code-based static features of both malware and non-malicious software applications. These features were used for malware detection based on: feature level fusion using power consumption and network traffic data, feature level fusion using network traffic data and system logs, and multimodal feature level and decision level fusion.

The contributions when using feature level fusion of power consumption and network traffic data are: (1) We focused on detecting real malware using the extracted dynamic behavioral features (both power-based and network traffic-based) and supervised machine learning algorithms, which has not been done by any of the prior works. (2) We ran a large number of machine learning experiments, which allowed us to identify the best performing learner, DC voltage rails that led to the best malware detection performance, and the subset of features that are the best predictors for malware detection. (3) The comparison of malware detection performance was done using a comprehensive set of metrics that reflect different aspects of the quality of malware detection.

In the case of the feature level fusion using network traffic data and system logs, the contributions are: (1) Most of the previous works that have used network flows-based features have done classification of the network traffic, while our focus was on classifying the software running in a machine as malware and non-malicious software using the extracted dynamic behavioral features. (2) We experimented with different sizes of the training set (i.e., 90%, 75%, 50%, and 25% of the data) and found that smaller training sets produced very good classification results. This aspect of our work has a practical value because the manual labeling of the training set is a tedious and time consuming process.

In this dissertation we present a multimodal deep learning neural network that integrates different modalities (i.e., power consumption, system logs, network traffic, and code-based static data) using decision level fusion. We evaluated the performance of each modality individually, when using feature level fusion, and when using decision level fusion. The contributions of our multimodal approach are as follow: (1) Collecting data from different modalities allowed us to develop a multimodal approach to malware detection, which has not been widely explored by prior works. Even more, none of the previous works compared the performance of feature level fusion with decision level fusion, which is explored in this dissertation.

(2) We proposed a multimodal decision level fusion malware detection approach using a deep neural network and compared its performance with the performance of feature level fusion approaches based on deep neural network and standard supervised machine learning algorithms (i.e., Random Forest, J48, JRip, PART, Naive Bayes, and SMO).

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Dr. Katerina Goseva-Popstojanova for her continuous support during my PhD journey and for her patience, motivation, and immense knowledge. I could not have imagined having a better advisor and mentor.

I would also like to thank the rest of my dissertation committee: Dr. Vinod Kulathumani, Dr. Roy Nutter, Dr. Stacy Prowell, and Dr. Yanfang Ye for their encouragement and for the hard questions which helped me to widen my research from various perspectives.

My sincere thanks go to Dr. Jeffrey A. Nichols and again to Dr. Stacy Prowell from Oak Ridge National Laboratory for their support with the initial hardware and software configuration of the experimental machine. Also, I would like to thanks my lab mates, Lingwei Chen, Mohammad Ahmad, and Yasser Alshehri, for their valuable time when discussing ideas and for their comments which helped me to strengthened the research.

Last but not the least, I would like to thank my family: my parents, my siblings, and my grandparents for supporting me throughout writing this dissertation. I will always be grateful for everything they have done for me. Specially, I want to dedicate this dissertation to my grandma who sadly passed away while I was writing it. This is for you my sweet angel!

This work was funded in part by the National Science Foundation under the grant CNS-1618629 and by the Lane Graduate Fellowship.

Contents

List of Figures

List of Tables

List of Abbreviations

AC	Alternate Current
ACF	Autocorrelation Function
AGP	Accelerated Graphics Port
ANN	Artificial Neural Network
API	Application Programming Interface
ATX	Advanced Technology eXtended
AV	Anti-Virus
CFG	Control Flow Graphs
COS	Cosine Similarity
CNN	Convolutional Neural Network
CPI	Cycle Per Instruction
CPU	Central Processing Unit
CSV	Comma-Separated Value
DAQ	Data Acquisition System
DC	Direct Current
DDoS	Distributed Denial of Service
DIMM	Dual Inline Memory Module
DKOM	Direct Kernel Object Manipulation
DLL	Dynamically Linked Library
DNS	Domain Name System
DoS	Denial of Service
EPROM	Erasable Programmable Read-Only
FN	False Negatives
FP	False Positives
FDR	False Detection Rate

FPR	False Positive Rate
GPU	Graphics Processing Unit
HIDS	Host-Based Intrusion Detection System
IAT	Import Address Table
ICS	Industrial Control System
IDS	Intrusion Detection System
IQR	Interquartile Range
IRC	Internet Relay Chat
ISA	Industry Standard Architecture
LDA	Linear Discriminant Analysis
LKM	Loadable Kernel Module
mA	Milliampere
MBR	Master Boot Record
MFCC	Mel Frequency Cepstrum Coefficient
NIDS	Network-Based Intrusion Detection System
OCP	Overcurrent Protection
OS	Operating System
P2P	Peer-to-Peer
PC	Personal Computer
PCA	Principal Component Analysis
PCAP	Packet Capture
PCB	Print Circuit Board
PCI	Peripheral Computer Interface
PCIe	PCI Express
PDA	Personal Digital Assistant
PE	Portable Executable
PFP	Power Fingerprinting
PLC	Programmable Logic Controller
POS	Point of Sales
PSU	Power Supply Unit
ROM	Read Only Memory
RBF	Radial Basis Function

RNN	Recurrent Neural Network
SCADA	Supervisory Control and Data Acquisition
SDR	Software Defined Radio
SIMM	Single Inline Memory Module
SSD	Sum of Square Distance
SSDT	System Service Descriptor Table
SVM	Support Vector Machine
TLS	Thread Local Storage
TN	True Negatives
TP	True Positives
URL	Uniform Resource Locator
WMI	Windows Management Instrumentation

Chapter 1

Introduction

1.1 Background & Motivation

Malware is a malicious software that is developed and propagated by cyber criminals to launch a wide range of security attacks, such as stealing confidential data, hijacking devices remotely to deliver massive spam emails, launching denial of service attacks, identity theft, and so on [254]. A recent study [235] showed the average time to resolve a malicious attack is fifty days and the average time to resolve a ransomware attack is twenty three days. Moreover, the financial consequences of cyber-attacks are worsening [235]. In order to protect the computer systems against the evolving threat malware poses, malware detection is imperative to both anti-malware industry and users.

Typically, malware uses polymorphic techniques to avoid detection. Polymorphic malware can bypass current detection methods by slightly changing the instructions of an existing malware sample. These new malware instances are called *variants*. Although these variants appear to be different programs from the viewpoint of anti-virus (AV) software, they exhibit similar functionality to their predecessor. Consequently, these new malware variants can bypass traditional detection methods until a pattern-matching for them can be identified and incorporated into the detection system.

Today anti-malware industry uses data mining techniques to detect malware.

These techniques include two stages: feature extraction and classification. For data collection and feature extraction, there are two methods used in the malware detection research area: *static* and *dynamic* [300]. The static methods extract features based on the analysis of the binary code of malware examples without executing the malware. On the other side, dynamic methods require the execution of a given malware example, typically in a sandbox environment [305, 275], and extract behavior-based features that represent the actions performed by the malicious software.

Both static (code-based) and dynamic (behavioral-based) feature extraction methods have their own advantages and disadvantages. Static methods contains very useful information from the binary code and are easy to extract, but they are prone to obfuscation techniques, which are commonly used by polymorphic malware [270], and attacks based on packer-based encryption [116]. In the case of dynamic methods, the main advantage is that they reflect the runtime behavior of a program which is hard to obfuscate [286, 65], but the data collection process is time and resource consuming [283, 143].

With respect to malware detection, nowadays it is typically conducted via the implementation of machine learning methods. The basic idea behind machine learning is to train a model based on a specific algorithm to perform the classification (i.e., classify between malware and non-malicious software). The training of the algorithm is done based on the input dataset, and the model that is built is subsequently used to make classifications. The performance of malware detection approaches depends critically on both the extracted features and the classification techniques.

Considering the limitations of static and dynamic methods, in this dissertation we explore the effectiveness of using different modalities for malware detection. Our dataset was created by using features from multiple sources (i.e., code-based static data, power consumption, system logs, and network traffic data). Each of these sources is called a *modality* [79].

To collect the static data we used the PE Explorer software tool [51] and to

collect the dynamic behavioral data we used our testbed [171]. The power consumption, system logs, and network traffic data were collected while the malware and non-malicious applications ran separately, in a controlled sandbox environment, on the experimental machine. Thus, based on the place of analysis our approach belongs to the commonly used remote server/cloud detection approach (e.g., [305, 275, 300]). Power data was collected by using a Data Acquisition System (DAQ) which measured the power consumption from four different voltage rails (+3.3V, +5V, +12V on the motherboard, and +12V CPU rails). System logs were collected using CaptureBAT [32] and the network traffic data was collected using Wireshark [42].

For our experiments we selected examples of recent malware with different traits, such as viruses, worms, trojans, backdoors, rootkits, and ransomware. In the case of the non-malicious software, we used some applications that are network intensive and other that are CPU and memory usage intensive. Compared to datasets from previous works [91, 201] which included power-based features for malware detection, our dataset is the largest. Bridges et al. [91] used five malware examples and Luckett et al. [201] used four malware examples, while here we are using fifty one malware examples and twenty two non-malicious applications.

With respect to the classification stage of malware detection, we used the supervised machine learning approach. To classify any unknown file, which could be malicious or non-malicious, the classification process has two steps: model construction (i.e., training) and model usage (i.e., testing). In the training step, samples of labeled (i.e., known) malware and non-malicious software are provided to the system and the feature vectors are extracted. Both the feature vectors and the class label (i.e., malicious or non-malicious) are used to build a classification model (or a classifier). During the model usage phase (i.e., testing), the classifier generated in the training phase is used to classify a new collection of previously not seen applications, which could be either malicious or non-malicious.

When multiple sources are integrated to perform an analysis the task is referred to as *multimodal fusion* [74]. Two levels of multimodal fusion exist: *feature level*

and *decision level*. Feature level (also known as early fusion) is the most widely used approach as it fuses all the extracted features into one feature vector, while decision level fusion (also known as late fusion) fuses multiple modalities in the semantic space [74].

1.2 Contributions

The contributions of this dissertation can be summarized as follows:

- We developed a testbed [171], which was used to collect power consumption, network traffic data, and system logs when running samples of malware and non-malicious software applications. Power data was collected by using a Data Acquisition System (DAQ) which measured the power consumption from four different voltage rails, while the system logs were collected using CaptureBAT [32], and the network traffic data was collected using Wireshark [42].

- In addition to dynamic behavioral-based features (i.e., power-based features, network traffic-based features, and system logs-based features) we extracted code-based static features (i.e., headers-based, data directories-based, and DLL dependencies-based features), which are typically used for malware detection.

- With respect to power consumption-based features, we identified the best performing DC voltage rails that led to the best malware detection performance. Our dataset is the largest when we compared it to prior works [91, 201] that used power-based features for malware detection.

- Most of the previous works that have used network flows-based features [124, 120, 98, 61, 316, 73, 289, 117, 125, 84, 308, 204, 148, 149] have done classification of the network traffic, while in this dissertation we focused on classifying the software running in a machine as malware and non-malicious

software using the extracted code-based static and dynamic behavioral-based features.

- We explored feature selection using information gain and identified the smallest number of features sufficient to distinguish malware from non-malicious software [156, 171]. We also experimented with different sizes of the training set (i.e., 90%, 75%, 50%, and 25% of the data) and found that smaller training sets produced very good classification results [171]. This aspect of our work has a practical value because the manual labeling of the training set is a tedious and time consuming process.

- Collecting data from different sources allows us to develop a multimodal approach to malware detection, which has not been widely explored by the prior works. Exceptions are [193, 184]. Kumar et al. [193] used two modalities and Kim [184] divided the code-based static features into 7 feature vectors, and used each of them as an individual modality. Both of the prior works [193, 184] monitored mobile devices, while here we monitored a general-purpose computer. None of these works compared the performance of feature level with decision level fusion, which is explored in this dissertation.

- We proposed a multimodal decision level fusion malware detection approach using a deep neural network. We compared its performance with the performance of feature level fusion approaches based on deep neural network and standard supervised machine learning algorithms (i.e., Random Forest, J48, JRip, PART, Naive Bayes, and SMO). Kim et al. [184] used only code-based static features, while we are combining behavioral-based with code-based static features.

1.3 Main Findings

We first experimented with power consumption and network traffic data and used ten supervised machine learning algorithms (i.e., J48, Random Forest, Random

Tree, OneR, Naive Bayes, JRip, PART, Multilayer Perceptron, SMO, and Decision Table) for classification. The main findings include: (1) Among the best performing learners, Random Forest had the highest F-score and close to the highest G-score. (2) Power data extracted from the +12V CPU rails led to better performance than power data from the other three voltage rails. (3) Using only power-based features provided better performance than using only network traffic-based features; using both types of features had the best performance. (4) Feature selection based on information gain was used to identify the smallest numbers of features sufficient to successfully distinguish malware from non-malicious software. The top eleven features provided the same performance as using all 25 features. Five out of seven power-based features were among the top eleven features.

We also experimented with network traffic data and system logs by evaluating four supervised machine learning algorithms (i.e., J48, Naive Bayes, Random Forest, and PART) for malware detection and identified the best learner. Furthermore, we used feature selection on information gain to identify the smallest number of features needed for classification and experimented with different training sets of different sizes. The main findings include: (1) Adding network flows-based features improved significantly the performance of malware detection. (2) J48 and PART were the best performing learners, with the highest F-score and G-score values. (3) Using J48, the top five features ranked by information gain attained the same performance as when using all 88 features. In the case of PART, the top fourteen features ranked by information gain led to the same performance as when all 88 features were used. None of the system logs-based features were included in these two models. (4) The classification performance when training on 75% of the data was comparable to training on 90% of the data. As little as 25% of the data can be used for training at an expense of somewhat higher, but not very significant performance degradation (i.e., less than 7% for F-score and 6% for G-score compared to when 90% of the data were used for training).

In addition, we explored the effectiveness of integrating all four modalities (i.e., power consumption, network traffic data, system logs, and code-based static data)

for malware detection by using a deep learning neural network. To evaluate the performance of our multimodal approach, we conducted various experiments. We compared the performance of our multimodal fusion approach with each modality individually and to other learners (i.e., Random Forest, J48, JRip, PART, Naïve Bayes, and SMO) when using feature level fusion. Furthermore, we compared the performance of our deep learning neural network when using feature level and decision level fusion. The main findings include: (1) When using multimodal feature level fusion, the performance of the deep neural network was worse than Random Forest, J48, JRip, PART, Naive Bayes, and SMO. (2) Using deep learning neural network for multimodal decision level fusion outperformed these standard supervised machine learning algorithms.

1.4 Dissertation Overview

The rest of this dissertation is organized as follows: Chapter 2 provides a detailed background on malware. Chapter 3 presents a literature review on existing malware detection methods that used similar code-based static and dynamic behavioral-based features, and includes the state of the art with respect to multimodal fusion. Chapter 4 describes the preliminary experimental set-up and presents a proof of concept study that shows the feasibility of our testbed. Chapter 5 explains the modifications done in the experimental set-up to collect simultaneously data from multiple modalities and explains the malicious and non-malicious software selection. Chapter 6 explains the data pre-processing and feature extraction process. A description of the used supervised machine learning experiments and performance metrics is given in Chapter 7. The conducted machine learning experiments when combining the power-based and network traffic-based features is described in Chapter 8, while the experiments when combining network traffic-based and system logs-based features is given in Chapter 9. Our multimodal malware detection approach and results are described in Chapter 10. Threats to validity are given in Chapter 11. The conclusion and future work are presented in Chapter 12.

Chapter 2

Background on Malware

This chapter defines what is malware, describes the common malware types, explains why cyber criminals use malware, and provides several examples of cyber-attacks that were caused by malware.

2.1 What is Malware?

The term malware is a combination of the words *mal*icious and soft*ware*. Malicious software is any software that is used to disrupt the operations of a machine, to gather sensitive data, or gain access to private computer systems [50, 254]. Malware is created by cyber criminals with the objective of achieving particular goals. These goals can include stealing confidential data, harvesting logins and passwords, sending spam emails, launching denial of service attacks (DoS), and extortion or identity theft [254]. An example is the malware called *CryptoLocker*, which has been and is still used by cyber criminals to infect and encrypt all the files on the computer, so that they can later ask for a ransom in order to decrypt these files [34].

2.2 Common Malware Types

Malware can fall into many different categories, depending on the method of transmission, its mechanism of operation and what actions are taken once it gains a foothold [179]. Most common malware types are: *viruses*, *worms*, *trojans*, *backdoors*, *rootkits*, and *ransomware*. A computer virus is a piece of code that typically needs human action to spread itself into one or more files and then performs some action [87].

A worm is a program that copies itself from one computer to another [87]. The difference between a virus and a worm is that a worm spreads on its own through the network, that is, a worm does not need human action to spread [41]. Most of the time, worms cause at least some harm to the system network while viruses typically corrupt or modify files on a targeted computer [45].

A trojan horse is a malicious computer program which has a hidden functionality and typically misrepresents itself as useful, routine, or interesting in order to persuade a victim to install it [53]. A difference between a virus, a worm, and a trojan is that trojans do not attempt to inject themselves into other files or otherwise propagate themselves [1]. Typically, backdoors are left after using a trojan or a worm. As the name implies, backdoors, open a "backdoor" into a computer with the objective of leaving a network connection for the cyber criminal or other malware to enter the system or to spread spam [36]. In other words, a backdoor is a type of malware that consists of a method for bypassing normal authentication or encryption in a computer system [44, 31]. Furthermore, many trojan's payload act as a backdoor by contacting a controller which can then have unauthorized access to the affected system [53]. In the context of malware, a *payload* refers to the portion of the malware which performs the malicious action(s) [39].

A well-known malware type are rootkits. Rootkits are a "kit" consisting of small and useful programs that allow an attacker to escalate to maximum privileges [159]. Rootkits are designed to hide the existence of certain processes or programs from normal methods of detection and enable continued privileged ac-

cess to a computer [206]. Typically, a rootkit has three goals: run, hide, and act [274]. Rootkits run other malware on the target machine without restrictions to avoid detection by an anti-virus (AV) or other security tools, and to get information (e.g., user's passwords) from the compromised computer. They work by using a basic concept called *modification*. Essentially, a rootkit locates and modifies the software with the purpose of changing the software behavior. An example of a type of modification that can be made by a rootkit is *patching*, which is a technique that modifies the data bytes encoded in a executable code [159].

While a rootkit hides from detection, a ransomware (also known as crypto-virus, crypto-trojan or crypto-worm) threatens to publish the victim's data or perpetually block access to it unless a ransom is paid. Ransomware attacks are often carried out by using trojans [53]. Typically, a ransomware encrypts all the files of the victim's system and then demands a ransom payment in return for the decryption key which is required to decrypt the encrypted files [59]. Most of the time they are installed in the system through a malicious email attachment, an infected software download, or by visiting a malicious website or Uniform Resource Locator (URL). Once the system is infected with ransomware, the user's files are encrypted, and/or the user is restricted from accessing the computer's main features. Some ransomware-based applications disguise themselves as an authority figure (e.g., a police or a government agency such as the Federal Bureau of Investigation or the Department of Defense) claiming that the user's system was locked down for security reasons and that a ransom or a fee is required to reactivate it [280]. The ransom message usually includes instructions on how to pay the ransom (most of the time is either through credit card or bitcoins). Ransom amounts range from one hundred dollars to several thousand dollars [281]. Figure 2.1 shows the message that appeared after we executed the Locky [49] ransomware on the experimental machine.

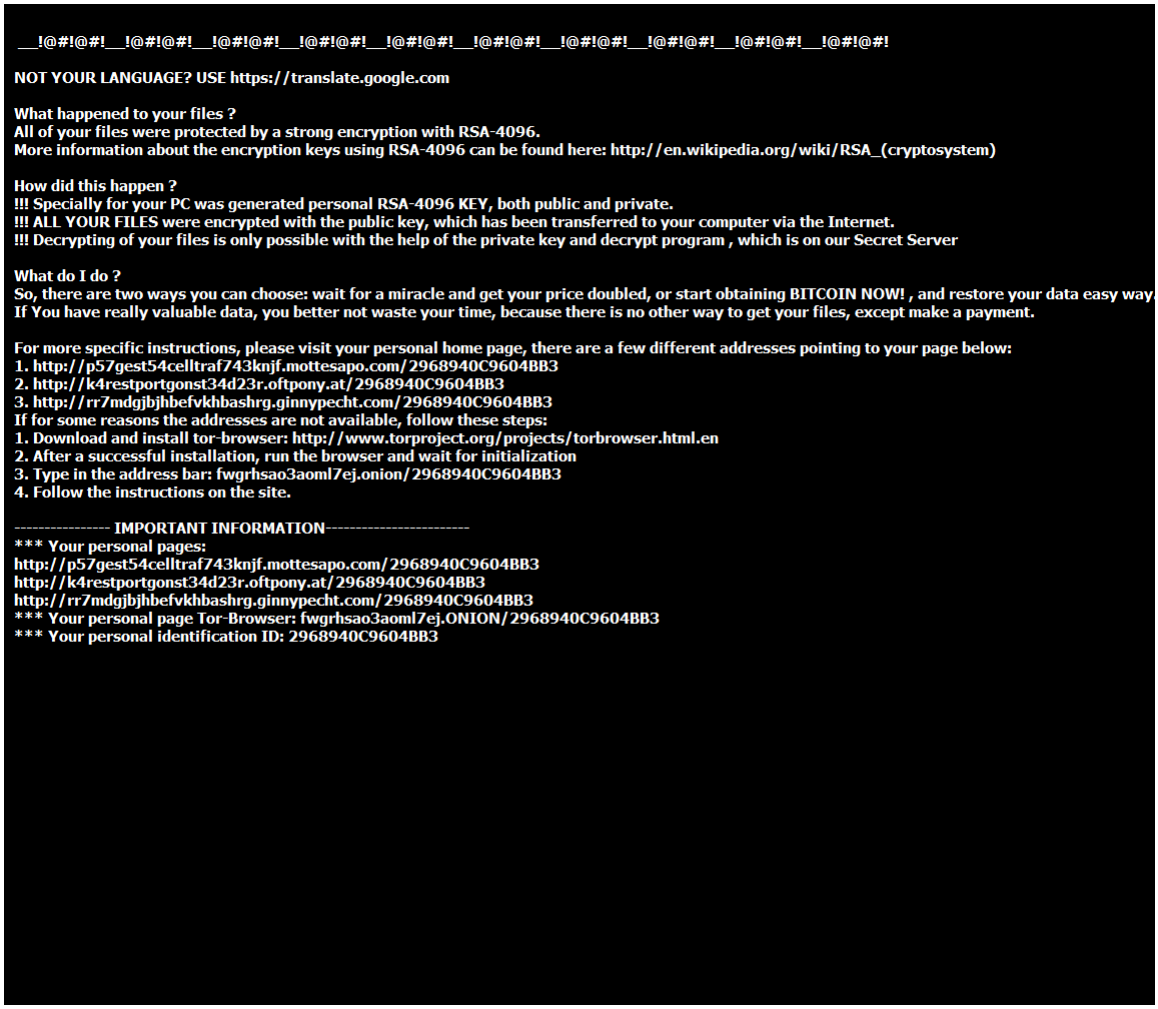

Figure 2.1: System message after infecting the experimental machine with the Locky ransomware

2.3 Why Cyber Criminals use Malware?

There are many ways in which cyber criminals use malware. For instance, malware authors are increasingly taking advantage of the trust that exists between users and software providers to inject malware on these updates, thus potentially infecting the users through trusted official software distribution channels. A report by FireEye iSIGHT Intelligence stated that at least there were five cases in which malware authors compromised software providers [136].

Rootkits and ransomware are preferred among cyber criminals because of their

effectiveness in achieving their goal (e.g., gaining access to the system) and because, at least in the case of rootkits, they can hide to avoid detection by using modification techniques that are hard to detect. Most of the time cyber criminals installed the rootkits once they have obtained root or administrative access to the system. Obtaining root or administrative access to a system is a result of a direct attack on the system. An example of direct attack is when a cyber criminal gains full control over a system by exploiting a known vulnerability or a password. Full control over a system means that existing software can be modified, including software that might be used to detect malware.

In the case of a ransomware, cyber criminals use them as a convenient payment system because it is hard to trace. Hence, they use them to commit financial fraud and extort money from computer users. However, not every type of ransomware will demand a cryptocurrency (digital currency that uses encryption techniques such as bitcoins) payment. For example, some types of ransomware demand a gift card code or other anonymous online payment option [94]. Examples of the most common payments methods for ransomware are wire transfers, premium-rate text messages, pre-paid voucher services (e.g. Paysafecard), and bitcoins [40].

Overall, cyber criminals use malware to steal passwords or network bandwidth, or to install other malicious software [274], and to gain and maintain unauthorized access to a system. By gaining unauthorized access to a system, the cyber criminals can obtain privileges to access sensitive data and conceal its own existence. Furthermore, a cyber criminal could use any type of malware that has rootkit capabilities to hide other malware types. Malware hidden by rootkits often monitor, filter, and steal data, or could abuse the computer's resources [27].

2.4 Cyber-attacks Caused by Malware

A cyber-attack refers to any act or attempt, successful or unsuccessful, to gain unauthorized access to, disrupt or misuse a Licensee's electronic systems or information stored on such systems [12]. Not all cyber-attacks are caused by malware,

they could also be caused by exploits (a vulnerability in the system) or by other types of attacks such as a DoS (a type of attack in which the cyber criminal seeks to make a machine or a network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host that is connected to the Internet). However, in this section we focus on those cyber-attacks that were caused only by malware.

A well-known cyber-attack caused by malware is *Stuxnet*. Stuxnet is a worm with rootkit capabilities that was first uncovered in 2010 [282]. Stuxnet has three modules: a worm that executes all routines related to the main payload of the attack; a link file that automatically executes the propagated copies of the worm; and a rootkit component responsible for hiding all malicious files and processes [285]. It was introduced to the target environment via an infected USB flash drive. Once the machine was infected, the malware spread across the network scanning for Siemens Step-7 software on computers controlling a Programmable Logic Controller (PLC) [282]. This worm subverts the Step-7 software application that was used to reprogram these devices. This worm collected information on industrial control systems (ICS) and caused the fast-spinning centrifuges to tear themselves apart [195]. Siemens stated that the worm did not cause any damage to its customers, but it is believed that the Iran nuclear program was damaged by this cyber-attack [4, 203]. A report by Symantec showed that 60% of the infected computers worldwide were located in Iran [102, 135, 83].

In addition, a nuclear power plant in Russia was also infected by this worm. However, since the power plant was not connected to the public network, the system remained safe [267]. Like Stuxnet there are many malware that has targeted and keep targeting industrial control systems. Some examples are *Shamoon* [115] and *Dragonfly* [13, 257]. However, to this day the latter has not been used to attack ICS. Rather it has been used for counterfeiting [13] and cyber espionage purposes [257]. Even though these cases affected mainly countries that are not the United States (US), is imperative to be aware of them since the critical infrastructure of the US could be affected by similar threats.

Furthermore, there have been cyber-attacks that targeted other systems as well. Some of the biggest data breaches happened to companies such as *Target*, *The Home Depot*, and *Anthem*. In the case of Target, cyber criminals installed malicious software on the point of sales (POS) systems in the self-checkout lanes from nearly 2,000 Target stores [255]. The objective of the cyber-attack was to gain access to customer credit and debit card numbers. This malware compromised the identities of 70 million customers and 40 million credit and debit cards [255, 276]. The same malware was later used to target The Home Depot [256]. On the other hand, Anthem (a health insurance plan provider) was a victim of cyber criminals when they stole approximately 80 million of medical records [276]. The attack began with phishing emails that were sent to Anthem's employees and it did not became successful until some of these employees were tricked and downloaded a trojan with a keylogger capability that enable the cyber criminals to acquire the passwords for accessing the unencrypted data.

In addition, there has been malware that targeted regular users (people that use general-purpose computers from the comfort of their home). Some examples of these malware are *Alureon* and *GameOver Zeus*. Alureon, also known as TDL, is a trojan with rootkit capabilities that was first discovered in 2008. It was created to steal data by intercepting a system's network traffic and whose objective was to search for personal information such as banking usernames and passwords, credit card data, social security numbers and other sensitive user data. It was not until 2012 in which a new variant of this malware was discovered. Like its predecessor, it was used to steal personal information from its victims by redirecting them away from trusted websites. The number of computers that probably were infected was more than 277,000 worldwide, but the FBI believes that about 64,000 computers were infected only in the United States [294].

Similarly, GameOver Zeus (GOZ), a variant from the Zeus trojan was used by cyber criminals to send spam and phishing messages, to participate in Distributed Denial of Service (DDoS) attacks, and harvest banking information, such as login credentials, from a victim's computer [292, 11]. As many as 1.2 million computers

were infected with this trojan prior to the takedown of the Zeus malware [186]. More examples of malware that targeted general-purpose computers can be found in Chapter 5. Particularly, Table 5.1 lists the malware examples that were chosen for our experiments.

Chapter 3

Literature Review

Malware is a malicious software that is developed by cyber criminals in order to steal confidential data, hijack devices remotely to deliver massive spam emails, launch denial of service attacks and so on. Typically, they avoid detection by constantly changing the program's appearance while keeping its functionality the same. This malicious behavior is attained by manipulating the code using multiple obfuscation techniques, such as inserting junk code and reordering instructions [247].

Lately, many authors of malware detection systems have attempted to address this problem by using different detection approaches, such as byte frequency [312], byte randomness [237], and behavioral patterns identified in the binary code of malware examples (i.e., behavioral analysis) [229, 154, 65, 283, 133]. The byte frequency of software refers to the frequency of the different unsigned bytes in the corresponding file, byte randomness refers to the bytes distribution value of the instruction sequences that are obtained from randomness tests, and behavioral analysis refers to the type of analysis that identifies the actions performed by the malware rather than their binary code patterns.

Next we provide an overview about previous works that have used code-based static and dynamic behavioral-based features, as well as those previous works that have used multimodal fusion techniques for their classification.

3.1 Malware Detection using Behavioral-based Features

The evolving evasion techniques being used by malware writers led to the usage of dynamic behavioral-based features for detection of malicious software [143]. Extracting behavioral-based features involves the execution of the PE file in a controlled environment (e.g., virtual machine and sandbox) [143]. Some types of dynamic behavioral-based features include function call monitoring, function parameter analysis, information flow tracking, and instruction traces [128, 143].

In addition, there are several online automated tools that helps to collect behavioral data from malware and non-malicious software. Some examples of such tools are CWSandbox [293], TTAnalyzer [82], Cuckoo sandbox [30], and Payload Security [26]. The behavioral reports generated by these tools helps malware analysts to understand the malware behavior and provide valuable insight into the actions performed by them. Some details to consider when extracting dynamic behavioral-based features are: (1) each malware example should be executed within a secure environment for a specific time to ensure malware examples behave as intended [270]; (2) a secure environment is different from a real runtime environment as the malware may behave differently on each of these environments, leading to inaccurate behavior [128]; and (3) some actions of the malware example may only be activated or triggered under certain conditions (e.g., system date and time or direct input from the user) [167].

Compared to code-based static features, dynamic behavioral-based features are more costly. However, dynamic behavioral-based features are more resilient to obfuscation techniques because they extract behavior actions performed by the malware rather than their binary code patterns. All dynamic behavioral-based features vary in the execution environment for the malware and analysis granularity. For example, a debugger (e.g., GDB [200] and WinDbg [244]) can be used for fine-grained analysis of binary code at the instruction level and other tools, such

as Detours [162], CWSandbox [293], TTAnalyzer [82], Cuckoo sandbox [30], and Payload Security [26], run the malware example in a controlled environment and monitor its behavior.

The behavior of a software application, including a malicious application, can be characterized by its system and network activities, as well as by the analysis of its physical properties (e.g., power consumption). Although malware detection is a very active area of research [165, 275, 305] and dynamic behavioral-based features has been used widely [229, 154, 65, 283, 133, 300], few works were focused on using physical properties, such as power consumption. In this dissertation we extracted dynamic behavioral-based features from the power consumption, network traffic data, and system logs.

3.1.1 Power-based Features

Monitoring power consumption has been explored by previous works for the development of new approaches to help with energy efficiency [164, 189, 134, 144, 147, 233, 219], energy theft [103, 207], and to help for integrity assessment [132]. However, for these approaches power consumption was monitored for a different purpose than malware detection. For instance, power consumption was monitored to help data centers understand how much power was used among the running applications across the network (in case of servers) [164, 189, 134, 144, 147], to extend the life of the cellphone's battery (in case of mobile devices) [233], to prevent energy theft on embedded devices [103, 207], and to improve the power consumption on house appliances [219].

From these power-based approaches [164, 189, 134, 144, 147, 233, 219, 103, 207, 132], the work by Feng et al. [134] is the most relevant to our work as they used a similar hardware configuration to collect power consumption data. Like us, they used an ATX extender cable to attach the power supply unit (PSU) of the nodes to a sensor resistor on the circuit board. Specifically, they used a RadioShack 46-range digital multimeter (manufacturer part number 22-812) that led to a sampling rate

of 0.25 (i.e., four samples per second). Similarly, the work by Dawson et al. [114] used a multimeter and current clamp, which limited the sampling rate to 1Hz for the collection of power consumption data on a general-purpose computer for malware detection. While these works used a multimeter, our testbed used a data acquisition system (DAQ) with a sampling rate of 0.01 second (i.e., one sample every 10 milliseconds). Thus, in comparison to the work by Feng et al. [134] and Dawson et al. [114], our testbed provides fine grain power consumption data.

The following subsections discusses related works that have used power-based features for malware detection. These works focused on specific devices, such as mobile devices [158, 313, 303, 77, 122, 123, 303, 169], embedded devices [108, 155, 107, 212], software defined radio [63, 64, 62, 62, 242], and general-purpose computers [114, 91, 201].

A comparison of the most relevant works that have used power-based features is given in Table 3.1. Note that the main difference among the prior works that used power-based features is with respect to what was classified. Some works classified sub-segments of the power consumption data as malicious or non-malicious [77, 63, 242, 62, 114], distinguished malicious from non-malicious operations (i.e., turning the pump on/off [62, 108] or turning the lights of PLC on/off [155]), and carried on malware detection (i.e., classified the unknown applications to malware and non-malicious software) [158, 303, 91, 201].

Table 3.1: Most relevant works that used power-based features

Ref. #	*Device*	*Prediction*	*Technique*	*Learners*	*Classification*	*Place of Analysis*	*Features*	*Performance Metrics*
[158]	M	No	O	N/A	malware vs. non-malicious software	N/A	N/A	N/A
[303]	M	Yes	St	GMM	malware vs. non-malicious software	remote servers	N/A	NR, PR, A
[77]	M	Yes	ML-S	DTW, KNN	malicious vs. non-malicious sub-segments	remote servers	N/A	A, R, P, F
[169]	M	Yes	St	correlation	malware vs. non-malicious software	remote servers	N/A	Approach was not evaluated.
[108]	ED	Yes	ML-S	3-NN, MLP, RF	malware vs. normal operations	remote servers	mean, var, max, min, skew, Kurt, RMS, IQR	mean of A, P, R
[155]	ED	No	O	N/A	cyber-attacks vs. normal operations	N/A	N/A	N/A
[212]	ED	O	No	N/A	buffer overflow attacks vs. normal operations	N/A	N/A	N/A
[63]	SDR	Yes	St	correlation	malicious vs. non-malicious sub-segments	remote servers	N/A	Approach was not evaluated.
[242]	SDR	No	O	correlation	malicious vs. non-malicious sub-segments	remote servers	N/A	Approach was not evaluated.
[62]	SDR	Yes	AD	spectral periodogram	malicious vs. non-malicious sub-segments	remote servers	N/A	Approach was not evaluated.
[114]	PC	Yes	AD	non-linear phase space algorithm	malicious vs. non-malicious sub-segments	remote servers	max, min, mean	Approach was not evaluated.
[91]	PC	Yes	AD,ML-S	ensemble learning, SVM	malware vs. non-malicious software	remote servers	mean, var, DSD, skew, L^2Norm, Kurt, and permutation entropy	R, FDR
[201]	PC	Yes	ML-S	nested network	malware vs. non-malicious software	remote servers	N/A	mean A, AUC

Table description for each column that has abbreviations:

- *Device* column: M = mobile; ED = embedded device; SDR = software-defined radio; and PC = general-purpose computer
- *Technique* column: O = observations; St = statistic-based ; ML-S = supervised machine learning; and AD = anomaly detection
- *Performance Metrics* column: A = Accuracy; P = Precision; R = Recall; F = F-score; FDR = False Detection Rate; NR = Negative Rate; PR = Positive Rate; and AUC = Area Under Curve

Mobile Devices

Before smartphones arrived to the market, previous works explored if cyber-attacks could be detected on personal digital assistants (PDAs) by monitoring its power consumption [168, 182, 93]. Later, with the arrival, popularity, and usability of smartphones they became the perfect target for cyber-attacks (e.g., malicious code) [300].

Malware detection approaches based on power consumption for mobile devices showed inconsistent results. The approach proposed by Hoffman et al. [158] was not successful due to the noise caused by unpredictable factors, such as user interaction and the strength of the mobile signal. On the other hand, the methods proposed by Yang et al. [303] and Zefferer et al. [313] were able to detect malware by monitoring the power consumption of smartphones. A recent work presented by Azmoodeh et al. [77] demonstrated that a specific type of malware, ransomware, can be detected on Android devices by monitoring only the power consumption.

Furthermore, the works by Dixon et al. [122, 123] explored the effectiveness of detecting malicious code by combining the mobile power profiles with user's location [122], while an extended version of this work [123] integrated time as a feature. Results on both works demonstrated the effectiveness of these features for finding malware with a low false positive rate and a little impact to the battery life of smartphones.

Even though these works [77, 122, 123, 158, 303, 313] were able to detect malware by using power consumption as a feature, it is important to note that all of them used software-based monitoring (i.e., the PowerTutor tool) to collect the power consumption data, which may distort the power profiles and/or be affected by successful malicious attacks. An exception is the work by Robin et al. [169] which built their own testbed using a Monsoon power meter for the acquisition of power consumption data. Although preliminary results were promising in detecting malware from non-malicious applications, the development and validation process were not completed.

Embedded Systems

Several approaches have been proposed to detect malware targeting embedded devices (i.e., devices with a dedicated function within a larger mechanical or electrical system) [107, 108, 155, 212].

The work by Clark et al. [108] explored whether power consumption could be used to detect the presence of malware on two embedded devices, an embedded medical device and a pharmaceutical compounder (i.e., an industrial-control workstation). They monitored the alternate current (AC) outlet and showed that malware can be detected based on the power consumption of embedded devices using supervised machine learning algorithms. Same author (Clark et al.) presented in [107] two case studies in which it was proved that AC power traces can be both harmful to privacy and beneficial for malware detection, the latter of which may be beneficial for embedded devices (i.e., medical devices). However, the main issue when monitoring AC relies on periodic changes in the current direction, which leads the voltage to reverse itself, making the analog circuits much more susceptible to noise. To avoid this problem, we monitored the direct current (DC) channels, as some other prior works [158, 303, 77, 63, 242, 62, 114, 155, 91, 201].

Similarly, our previous work [155] presented a proof of concept study which demonstrated through observations based on illustrative examples that cyber-attacks can be detected by monitoring the power consumption of a Programmable Logic Controller (PLC). Power consumption data was collected using a data acquisition system (DAQ), but the hardware configuration was different than the one used in this dissertation. The main difference relies on the sensors that were attached to the DAQ, since the maximum voltage for the PLC rails were +24V and here our testbed monitored four voltage rails (i.e., +3.3V rails, the +5V rails, the +12V rails on the motherboard and the +12V rails on the CPU) whose maximum value is +12V. Moreover, for the experiments in [155] we simulated three SCADA-specific cyber-attacks (i.e., command injection, replay, and Denial of Service), while here in our experiments we used real malware examples.

Furthermore, another work that monitored the power consumption for an embedded device was presented by Moore et al. [212]. While it is important to mention that their objective was to detect buffer overflow attacks and not malware detection, this work is still related as they used power-based features for anomaly detection. Power consumption data was collected by using an I-jet module, a device capable of providing power to the target board and measuring its power consumption during program execution in real time. Power segments were analyzed and they demonstrated that it is possible to distinguish some cases of buffer overflow attacks (i.e., a program crash and injection of executable code) from normal operations.

Software Defined Radio

A software defined radio (SDR) is a radio communication system in which those components that were typically implemented in hardware (e.g., mixers, filters, and amplifiers) are instead implemented by means of software on a general-purpose computer or embedded system [118]. Few works explored the usage of power consumption for SDR [63, 64, 62, 242].

González et al. [63] proposed an approach that relies on a mechanism that enables an integrity assessment on SDR by capturing fine-grained measurements of the processor's power consumption and comparing them against signatures from trusted software. Their method collects fine-grained measurements from the power consumption during the execution of trusted code. Later, different signal processing techniques were applied to extract dissimilarity measures from the power segments. After the feature extraction, these power segments were passed through a supervised classifier or detector that has been previously trained using power segments from trusted software. Finally, a detector compares the test segments against all known signatures, and if no single test is enough to determine that authorized code was executed, then an intrusion is reported. This method was adapted by the Power Fingerprinting (PFP) firm (http://pfpcyber.com/) and can also be applicable to embedded systems [64, 62, 242]. Nonetheless, we must em-

phasize that these works [212, 63, 64, 62, 242] used only power segments, while in this dissertation we are using power-based features from the whole power signal and we are combining these features with other dynamic behavioral-based and code-based static features.

General-Purpose Computers

With respect to general-purpose computers, power consumption has received little attention as a feature for anomaly detection due to its noisiness which prevents fine-grained analysis of power traces [107]. Nevertheless, literature shows there are a few power-based approaches focused on general-purpose computers for identifying web pages by tapping the electrical outlet [106], and for malware detection [239, 290, 114, 91, 201]. From these previous works, the most relevant to our research are [114, 91, 201].

Dawson et al. [114] proved the algorithm developed in [157] can be used to detect the presence of malware (i.e., rootkits) through the collection and analysis of data from voltage measurements taken from one of the power supply rails. They collected power consumption data using a multimeter and current clamp [114, 201], which limited the sampling rate to 1Hz. While here we used a sampling rate of 100Hz to collect the power consumption data. Using hardware-based monitoring is more accurate and, unlike software-based monitoring tools (i.e., software used on mobile devices to collect power consumption data [158, 303, 77]), does not affect the power consumption on the experimental machine and is harder to be manipulated by successful malicious attacks. Similarly, Luckett et al. [201] extended the work in [114] by proposing a model using nested neural networks. When compared to traditional machine learning algorithms they demonstrated that the proposed model outperformed previous methods.

Another relevant work related to this dissertation is our previous work [91] in which we proposed an unsupervised anomaly detection ensemble using only the +12V CPU rails and compared its performance with several supervised kernel-based SVM classifiers (trained on clean and infected profiles) for detecting previ-

ously unseen malware. While we used the same hardware configuration as in [91] to collect the power consumption data, our software tools were different. In addition, our previous work used data only for the +12V CPU rails, while here we evaluate which voltage rail leads to best performance.

While all these works [114, 91, 201] used only power-based features and very small sets of malware (e.g., five [91] and four [201] malware examples), in this dissertation we used a larger set of malware and non-malicious applications (i.e. fifty one malware examples and twenty two non-malicious applications).

3.1.2 Network Traffic-based Features

Network traffic analysis is challenging due to the dynamic nature of network traffic. However, prior works proposed solutions to address this problem by using statistics, data mining, and machine learning techniques [166]. Anomalies in the network traffic data can be due to cyber-attacks, but also because of malfunctioning devices or network overloads. Thus, using reliable network traffic data is imperative.

In this dissertation, we used the dynamic behavioral data collected from our experimental set-up and extracted network traffic-based features, which can be divided into two categories: commonly used network traffic features and network flows-based features. A comparison of the most relevant works that used network traffic-based features is given in Table 3.2. Previous works that explored commonly used network traffic-based features have focused on network traffic classification for botnet detection [124, 61, 316, 98, 148, 204], detection for specific types of cyber-attacks such as Denial of Service [253], detection of anomalies related to specific protocols (e.g., HTTP) [229, 236], classification of the network traffic itself to malicious and benign [84], and malware detection (i.e., malicious vs. non-malicious software) [95, 210, 230].

Besides the commonly used network traffic-based features, we are also exploring the usage of network flows-based features for malware detection as some previous works [124, 120, 98, 61, 316, 148, 149, 73, 289, 117, 125, 308]. Network flows-based

features have been mainly used for detection of botnets [124, 120, 98, 61, 316, 148, 149, 227], for detection of anomalous network traffic [80], and for classification of malware families [240, 67]. With respect to malware detection, network flows-based features have not been extensively explored, except for a few works that focused on network traffic classification for malware detection in Android devices [73, 289], detection of worms [117, 125], and detection of other types of malware, such as trojans and viruses [84, 308]. From these approaches, we focused on those methods that used similar network traffic-based features and malware types [95, 210, 230, 117, 125, 84, 308, 240, 67].

Some of these works [95, 210, 230] executed the malware examples in a controlled environment to collect the dynamic behavioral-based features for the classification of malicious and non-malicious software [95] and for malware families classification [210, 230]. Burnap et al. [95] used machine learning techniques with behavioral-based features (i.e., CPU, RAM, processes and network traffic) derived from the footprint that was left behind on a computer system after the execution of a software to classify malware from non-malicious software. Mohaisen et al. [210] described a technique that relies on the order and frequency with which malware examples conduct specific actions on the system. Collected .pcap files were parsed for relevant events and subsequently n-grams features were extracted and used for malware classification. Radu et al. [230] proposed a malware classification approach based on features such as DNS-based, accessed files, mutexes, and Registry keys-based. The integration of these features helped to maintain the Accuracy of the used supervised machine learning algorithm.

With respect to network flows-based features, the most relevant prior works to ours are [117, 125, 84, 67, 308, 240]. Dubendorfer et al. [117] proposed an approach that used network flows from high speed Internet backbones demonstrating worms can be detected by tracking the cardinality of sudden changes in the network traffic. Dressler et al. [125] developed a pattern based on the correlation of flow-based features with system logs data for worms detection. Bekerman et al. [84] presented a malware detection approach that classified malicious and non-malicious

network traffic recorded in sandbox environments and in real networks. AlAhmadi et al. [67] proposed an approach that analyzed and classified network traffic of malware variants based on their network flow sequence behavior. Yeo et al. [308] classified network packets by botnets, trojans, and viruses using a convolutional neural network (CNN), while Rahul et al. [240] presented a CNN for classifying network traffic of malware families.

While prior works have integrated network flows-based features with system logs-based features for botnet detection [204], worm detection [125], malware detection [95], and malware families classification [210, 230], none of these works integrated the network traffic data and system logs with code-based static and power-based features for the classification of unknown applications to malware and non-malware.

Table 3.2: Most relevant works that used network traffic-based features

Ref. #	*Device*	*Prediction*	*Technique*	*Learners*	*Classification*	*Place of Analysis*	*Features*	*Performance Metrics*
[117]	PC	No	St	N/A	malicious vs. non-malicious network traffic	N/A	network flows-based	N/A
[125]	PC	No	O	N/A	malicious vs. non-malicious network traffic	N/A	network flows-based	N/A
[84]	PC	Yes	ML-S	NB, RF, J48	malicious vs. non-malicious network traffic	remote servers	network traffic-based	A, AUC
[308]	PC	Yes	ML-S, DL	CNN, MLP, RF, SVM	malicious vs. non-malicious network traffic	remote servers	network flows-based	A, P, R
[67]	PC	Yes	ML-S	KNN, RF	malware families classification	remote servers	network flows-based	P, R, F
[240]	PC	Yes	DL	CNN	malware families classification	remote servers	network flows-based	A
[95]	PC	Yes	ML-S	NB, RF, SVM, ANN	malware vs. non-malicious	remote servers	network traffic-based, CPU-based, RAM, swap usage, and processes-based	P, R, F
[210]	PC	Yes	ML-S	decision tree, KNN, SVM	malware families classification	remote servers	network traffic-based	A, P, R, F
[230]	PC	Yes	ML-S	RF	malware families classification	remote servers	DNS-based, mutexes, Registry keys, and accessed files	R, P, F, FPR, AUC

Table description for each column that has abbreviations:

- *Device* column: M = mobile and PC = general-purpose computer
- *Technique* column: O = observations; St = statistical-based; ML-S = supervised machine learning; and DL = deep learning
- *Performance Metrics* column: A = Accuracy; P = Precision; R = Recall; F = F-score; FPR = False Positive Rate; and AUC = Area Under Curve

3.1.3 System logs-based Features

Monitoring system's behavior is of great importance for malware analysts because it provides valuable information about the software, hardware, system processes and system components as well as information such as error and warning events related to the computer operating system. Previous works have used system logs-based features for intrusion detection [72, 202, 211, 311, 133, 126], to classify malware from non-malicious software [251, 250, 113, 163, 279, 119, 197, 295, 99, 222], for malware families classification [230, 131, 178], and for both malware detection and malware families classification [170, 97].

From these previous works, we focused on those approaches that did malware detection [251, 250, 113, 119, 197, 295, 170, 99, 97, 279, 163, 222]. These malware detection approaches [251, 250, 113, 119, 197, 295, 170, 99, 97, 279, 163, 222] can be divided based on the device being monitored (i.e., mobile devices [119, 197, 295, 170, 99, 97, 222] and general-purpose computers [251, 250, 113, 279, 163]). A comparison of previous works that used system logs-based features for malware detection is given in Table 3.3.

Salehi et al. [251] conducted several machine learning experiments using API names and arguments for malware detection and for malware families classification. For evaluation purposes, both API names and arguments were investigated separately and then combined. Results demonstrated the Accuracy of the learners improved by 6% when all features were used. Sainju [250] presented observations about specific system events triggered after infecting the experimental machine with different types of malware (e.g., trojans, worms). Dahl et al. [113] proposed the used of random projections to further reduce the dimensionality of the original input space before feeding the data to a neural network. This reduction technique allowed to train the neural network with one or more hidden layers reducing the two-class error rate by 43% when compared to Logistic Regression trained with all features. Huynh et al. [163] proposed an online algorithm for malware detection under concept drift when the behavior of malware changes over time. While a

most recent work, Stiborek et al. [279], proposed a malware detection approach using clustering techniques based on the behavior observed from system logs and network traffic data focused on the HTTP protocol.

Data analysis was conducted via machine learning techniques for most of these works [251, 113, 163, 279, 119, 197, 295, 170, 99, 97, 222], except for [250] which presented observations about the behavior of specific malware examples. Most of these works used only system logs-based features [251, 113, 163, 250, 119, 295, 99, 97, 170] or combined the system logs-based features with network traffic-based features [279]. In the case of mobile devices, some works combined system logs-based features with permissions [197] and with permissions and intent [222]. Interestingly, none of these works combined the system logs-based features with code-based static features or with other behavioral-based features like power consumption, which is explored in this dissertation.

Table 3.3: Most relevant works that used system logs-based features

Ref. #	*Device*	*Prediction*	*Technique*	*Learners*	*Classification*	*Place of Analysis*	*Features*	*Performance Metrics*
[251]	PC	Yes	ML-S	RF, J48, FT, SMO, NB, VFI, HyperPipes	malware vs. non-malicious software	remote servers	API names and API arguments	A, P, R, AUC, RMS
[250]	PC	No	O	N/A	malware vs. non-malicious software	N/A	system log events	N/A
[113]	PC	Yes	ML-S	Logistic Regression Neural Network	malware vs. non-malicious software	remote servers	API calls-based	FPR, FNR
[279]	PC	Yes	ML-S	proposed approach	malware vs. non-malicious software	remote servers	system logs and network traffic data	A, R, FPR
[163]	PC	Yes	ML-S	proposed approach	malware vs. non-malicious software	remote servers	API arguments, file system, and Registry	MCAE
[119]	M	Yes	ML-S	RF, Ridge Regression, SVM, Lasso	malware vs. non-malicious software	remote servers	API traces	R, TNR
[197]	M	Yes	ML-S	Neural Network	malware vs. non-malicious software	remote servers	API arguments and permissions	A
[295]	M	Yes	ML-S	proposed approach	malware vs. non-malicious software	remote servers	API arguments	A, P, R, F, FPR, FNR
[99]	M	Yes	ML-S	RF, NB, SGD	malware vs. non-malicious software	remote servers	API traces	R, P, F, FPR
[97]	M	Yes	ML-S	SVM	malware vs. non-malicious software	remote servers	API traces and permissions	A, FPR, FNR
[222]	M	Yes	ML-S	SVM, ANN, Logistic Regression	malware vs. non-malicious software	remote servers	system logs, intent and permissions	A, P, R, F
[170]	M	Yes	ML-S	proposed approach	malware vs. non-malicious software	remote servers	API arguments and system logs	A, FPR, FNR

Table description for each column that has abbreviations:

- *Device*: M = mobile and PC = general-purpose computer
- *Technique*: O = observations; and ML-S = supervised machine learning
- *Performance Metrics*: A = Accuracy; P = Precision; R = Recall; F = F-score; FPR = False Positive Rate; TNR = True Negative Rate; FNR = False Negative Rate; and MCAE = Mean Cumulative Absolute Error

3.2 Malware Detection Using Code-Based Static Features

Static analysis is the way to extract malicious features or bad code segments without executing the PE file [305, 143]. Before conducting the static analysis, the PE file has to be unpacked and decrypted. There are tools that are either a disassembler (e.g., IDA Pro [35]) or a memory dumper (e.g., OllyDump [38]) that can be used to reverse PE files. Disassembler tools display malware code as assembly instructions, while memory dumper tools are used to obtain protected code located in the main memory and dumps them into a file for further analysis [305, 143].

The main advantage of using code-based static features is that they help to explore and investigate all possible execution paths in malware examples. In addition, the experimental machine cannot be infected by the malware under study. However, a disadvantage of code-based static features relies on the fact that they are susceptible to code obfuscation. Moser et al. [214] explored the limitations of code-based static features and suggested that code-based static features alone are not enough for malware detection and stated that dynamic behavioral-based features can be a necessary and useful complement to code-based static features.

Most of the previous works that have used code-based static features for malware detection [262, 252, 191, 60, 177, 129, 205, 70, 260, 304, 301, 258, 209, 137, 181, 70, 192, 176, 68] focused on the detection of patterns using features extracted from Windows API calls [262, 252], byte n-grams [191, 60, 177, 129, 205, 70, 260], strings [262, 304], opcodes (operational codes) [301, 258, 209, 137, 181], and control flow graphs (CFG) [70, 192, 176, 68].

Windows API calls are used by most of the programs to send specific requests to the operating system. As such, these features are useful for malware detection since they reflect the behavior of program code pieces [305]. These features are commonly used by previous works in combination with other code-based static features [307, 302]. For example, API calls-based features have been combined

with headers-based features [307] and with control flow graphs [302].

Byte n-grams are substrings in the program code with a length of N [305]. These code-based static features can be extracted from malware examples and are used as a signature for recognizing malicious software. Using byte n-grams for malware detection is convenient because they yield high accuracy in detecting unknown malware [241]. Hence, they have been well explored over the last decade by previous works that focused on the binary code content [191, 60, 177, 129, 205, 70].

Strings features are based on encoded plain text and are typically a high-level specification of malicious behavior considering they can show the attacker's intent [304]. However, they are not commonly used by prior works because they can easily be manipulated by an attacker [241]. Operational code (opcode) refers to the portion of a machine instruction that specifies the operation to be performed. Literature on malware detection specifies that opcode-based features are more efficient and successful for classification, since they reveal statistical diversities between malicious and non-malicious applications [241]. Some prior works that explored the usage of n-gram opcode sequence for malware detection are [86, 215, 264, 194].

Finally, CFG are graphs that represent the control flow of a PE file and are commonly used in software analysis [70], malware detection [92, 68], and for both malware detection and malware families classification [176]. The problem with those approaches that use CFG-based features is that they require a database of signatures, meaning that it can detect known malware but may not be able to detect unknown malware [88]. Furthermore, maintaining this database is time and resource consuming.

Another type of code-based static feature used by previous works are the portable executable-based features. This type of code-based static features are extracted using the structural information from an executable file [241]. By structural information, we refer to the following pieces of information from a PE file: (1) file pointer; (2) import section; (3) export section; (4) PE header; and (5) resource

directory.

File pointer is the pointer that denotes the position within the file as it is stored on the hard disk drive. The import section include features, such as functions from which DLLs and object files are used. The export section describes which functions are exported. Header-based features describe the physical and logical structure of a PE binary and may include features like code size and debug size. Resource directory-based features are those features that are indexed by a multiple-level-binary-sorted-tree structure. Examples of these resource directory-based features are the dialogs and cursors which are used by a specific PE file. Resource directory-based features are meaningful for malware analysts because they indicate if the executable file was manipulated to perform malicious activity.

In this dissertation the extracted code-based static features can be grouped into three categories: (1) headers-based features, (2) data directories-based features, and (3) DLL dependencies-based features. Similar PE-based features were extracted and used by prior works for malware detection [265, 261, 190, 81, 220], malware families classification [299], and for both malware detection and malware families classification [190].

The works by Bat-Erdene et al. [81] and Yan et al. [299] extracted and used headers-based features, while the works by Saxe et al. [261] and Narouei et al. [220] extracted and used DLL dependencies-based features. Similarly, Shafiq et al. [265] extracted and combined the headers-based features, data directories-based features, and DLL dependencies-based features into one feature vector for malware detection. On the other hand, Kolosnjaji et al. [190] extracted and combined headers-based features and DLL-based features for both malware detection and malware families classification.

Although similar code-based static features have been explored by previous works, none of these works [265, 261, 190, 81, 220, 299, 190] combined code-based static features with dynamic behavioral-based features nor explored the performance of using different modalities for malware detection, which is explored in this dissertation.

3.3 Multimodal Learning for Malware Detection

Feature level fusion is widely used by prior works for malware detection [205, 70, 259, 167, 306, 299, 277, 199, 315]. However, none of these works called it feature level fusion. These prior works mentioned that "all features were combined into one feature vector", which is the definition of feature level fusion. While other prior works mentioned that both code-based and dynamic behavioral-based features were used for malware detection, which is the definition of multimodal (i.e., using features from multiple sources).

Similarly, previous works [205, 269, 314, 310, 101, 111] have done decision level fusion, but they were unimodal. Thus, we labeled these prior works as feature level and decision level fusion, which are described in Sections 3.3.1 and 3.3.2, respectively.

3.3.1 Feature Level Fusion

Many prior works have done feature level fusion for malware detection [205, 70, 259, 167, 306, 299, 277, 199, 315]. These works can be divided based on the device being monitored (i.e., mobile devices [277, 199] and general-purpose computers [205, 70, 259, 167, 306, 299, 315]).

Both of the mobile devices approaches [277, 199] combined code-based static and dynamic behavioral-based features (i.e., Spreitzenbarth et al. [277] used features like API calls and network traffic data, while Lindorfer et al. [199] used features like class structure, application names, file operations, and network activity) to classify malware from non-malicious software. With respect to those approaches that targeted general-purpose computers [205, 70, 259, 167, 306, 299, 315], some did malware detection [205, 70, 259, 306], malware families classification [167, 315], and both malware detection and malware families classification [299]. Furthermore, the works in [70, 259, 167, 299, 315] used both code-based static and dynamic behavioral-based features, while the works in [205] used only code-based static

features and [306] used file relations. With respect to which learners were used, most of the prior works used SVM [205, 70, 306, 277, 199]. While others explored SVM with other learners like KNN and Decision Trees [259], Random Forest [167], and KNN, Naive Bayes, and Decision Trees [299]. Besides SVM, the work by [315] used K-means with hierarchical clustering.

In this dissertation, we focus on those approaches that used similar features as ours for malware detection on a general-purpose computer [70, 259, 299]. Anderson et al. [70] described how to combine both code-based static (e.g., opcodes and CFGs) and dynamic behavioral-based features (e.g., API traces) using multiple kernel learning methods. Santos et al. [259] proposed a machine learning approach that combined static (opcode sequences) and dynamic data (API traces), while Yan et al. [299] conducted a systematic study using different feature types and experimented with different combinations of feature selection algorithms and classifiers.

3.3.2 Decision Level Fusion

Prior works show two types of integration strategies: *classifier fusion* and *decision strategy*. In classifier fusion (also known as multi-classifier system), the classifier combination process involves merging the individual (weaker) classifier to obtain a single (stronger) expert of superior performance [249, 234]. Examples of classifier fusion methods include boosting [140], bagging [89], and some variations of bagging like Random Forest [90]. Decision strategy combines information in a simple and straightforward way. Let us assume a system that consists of multiple modalities ($\mathbf{M} = \{m_1, m_2, m_3, ..., m_n\}$), each of which uses a trait and makes the authentication decision independently. The decision level strategy is then used to combine the decisions of the subsystem to produce the final decision.

Some previous works have done classifier fusion for malware detection [205, 269, 314, 310, 101, 111, 66, 78, 309, 291]. These works can be divided based on the device being monitored (i.e., mobile devices [269, 310, 101, 111, 309, 291] and

general-purpose computers [205, 314, 66, 78]). From those works that monitored general purpose computers, some did malware detection [205, 78] and malware families classification [314, 66]. The fusion techniques explored by these works were boosting [205], ensemble selection [78], stacking [314], and both boosting and bagging [66].

With respect to previous works that have done decision strategy for malware detection [315, 287, 146, 213], they can also be divided based on the device being monitored (i.e., mobile devices [315], and general-purpose computers [287, 146, 213]). Zhang et al. [315] proposed an unsupervised machine learning approach that used different modalities (i.e., PE-based and API calls-based features) for malware families classification. To combine the information from all modalities, a clustering ensemble based on mixture model was used. Wang et al. [287] presented a malware detection approach that used API calls-based features for malware detection using a linear weighted fusion method. Guo et al. [146] designed a multiple classification algorithm based on the Behavior Knowledge Space (BKS) algorithm using API calls for malware detection. Extracted features were divided into seven subsets (i.e., file I/O, DLL, network, memory, process, Registry, and socket), each subset was classified using several machine learning algorithms, and the data was fused using BKS. More et al. [213] presented a malware detection system consisting of disassemble process, code-based static features extraction, and feature selection using majority voting and veto voting.

Only two prior works have used multimodal learning for malware detection [193, 184]. Both works monitored mobile devices, but [193] focused on feature level fusion, while [184] focused on decision level fusion.

Pramod et al. [193] proposed a machine learning approach that collected various applications files from different sources and pre-processed these files into various images format (i.e., grayscale, RGB, CMYK, and HSL). Extracted image-based features were used to train three machine learning algorithms (i.e., Decision Tree, Random Forest, and k Nearest Neighbor). The performance of each machine learning algorithm were evaluated on various metrics, such as Recall, Precision, F-score,

and Accuracy. Results show Random Forest outperformed the other learners with a detection Accuracy of 91%. Kim et al. [184] presented a multimodal deep learning algorithm using different types of code-based static features (e.g., opcodes, API, permissions, and component environmental). The proposed framework conducts four major steps before doing the malware detection: raw data extraction process, feature extraction process, feature vector generation process, and detection process. Results show that the proposed approach attained an Accuracy rate of 98%.

From these prior works, the work by Kim et al. [184] is the most relevant to our work as they used a multimodal deep learning neural network for malware detection. However, there are several differences that distinguish our work from theirs:

- Kim et al. [184] used only code-based static features, while we are combining dynamic behavioral-based with code-based static features.
- Prior works [193, 184] monitored mobile devices, while here we monitored a general-purpose computer.
- The structure of our deep neural network is different than the one presented in [184]. For instance, [184] used ReLU as activation function, and this function has the problem that turns all negative numbers to zero, which decreases the ability of the model to fit or train the data properly. To avoid this problem, in our multimodal approach we used the exponential linear unit function (ELU) as activation function. Furthermore, the number of hidden layers and neurons is distinct.
- None of these works compared the performance of feature level with decision level fusion, which is explored in this dissertation.

Chapter 4

Preliminary Experimental Set-up & Proof of Concept Study

This chapter describes the initial experimental set-up and explains how we conducted a proof of concept study that explored the use of power consumption for malware detection (i.e., rootkit) in a general-purpose computer. The contribution here is the experimental design and unique solutions to the data collection. The work presented in this Chapter has been published in the 17^{th} IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) [91].

4.1 Initial Experimental Set-up

4.1.1 Hardware Configuration

Our experimental system is a Dell OptiPlex 755 computer with a clean installation of 32-bit Windows 7 Ultimate. The instrumentation for our experiments was a Data Acquisition system (DAQ), Model Number: USB-1608G Series [20]. The DAQ connects to the device's motherboard power connector, and the voltage and current are collected on each of the direct current (DC) power channels. The communication between the experimental machine and the data repository machine (i.e., machine that stores the power consumption data) was established through

USB port. The DAQ provides relatively high-resolution power data, is able to sample at a rate of 250KHz, and can monitor up to sixteen channels. Besides the DAQ, we also used an eight inch Advanced Technology eXtended (ATX) power cable that had one male and one female 24-pin connector. The 24-pin male connector was attached to the motherboard, and the 24-pin female connector was attached to the power supply unit (PSU). A PSU is an electronic device that supplies electric energy to an electric load. Specifically, a PSU converts alternate current (AC) to low-voltage regulated DC current for the internal components of a computer [18]. Power supplies are rated in terms of how many watts they generate. Most power supplies have overcurrent protection (OCP) to protect the circuit when the current reaches a value that will cause an excessive or dangerous temperature rise in conductors [160].

Each group of wires on the PSU are connected to a single OCP circuit that is called a rail. A PSU has three voltage rails: +3.3V, +5V, and +12V. The +3.3V rails and +5V rails are used by the digital electronic components and circuits in the system, such as adapter cards and disk drive logic boards [216]. The disk drive motors, CPU voltage regulators, and cooling fans are used by the +12V rails on the motherboard [216]. Table 4.1 provides a list of the devices that are typically powered by these voltage rails.

To ensure the power consumption data was collected adequately, three hardware configurations were tested. The first hardware configuration consisted of an ATX power extender cable and several minigrabbers. A minigrabber is a micro-hook test clip that allows analog discovery's signal wires to be connected to component leads, wires, and other circuit components [16, 17]. Before using the minigrabbers, we had to solder a wire on each micro clip manually. Figure 4.1 shows the DAQ used for the data collection, and Figure 4.2 shows how the minigrabbers looks like after the soldering process.

Using the first hardware configuration we monitored a total of eleven DC power channels (four pins had a signal of +3.3V, five pins had a signal of +5V, and two pins had a signal of +12V). Figure 4.3 shows the signal of each pin for version 2.0

of the ATX standard connectors and Figure 4.4 shows the first hardware configuration.

Table 4.1: Voltage rail usage for a general-purpose computer

Rail	**Devices Powered**
+3.3V	chipsets, some DIMMs, PCI/AGP/PCIe cards, miscellaneous chips
+5V	disk drive logic, low-voltage motors, SIMMs, PCI/AGP/ISA cards, and voltage regulators
+12V	motors, high-output voltage regulators, AGP/PCIe cards
+12V CPU	CPU

Acronyms:

- SIMM = Single Inline Memory Module
- DIMM = Dual Inline Memory Module
- PCI = Peripheral Component Interconnect
- PCIe = PCI Express
- AGP = Accelerated Graphics Port
- ISA = Industry Standard Architecture
- CPU = Central Processing Unit

Figure 4.1: USB-1608G DAQ

Figure 4.2: Minigrabbers

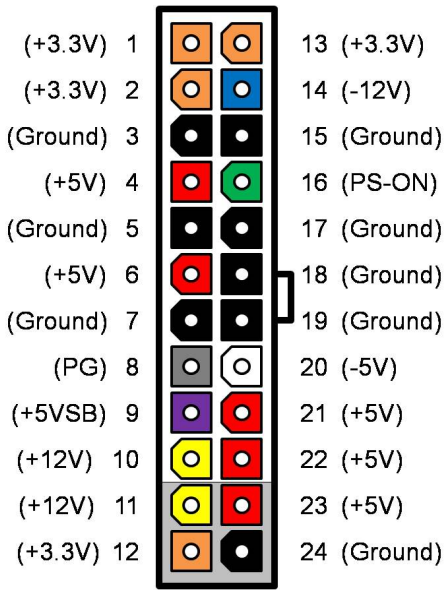

Figure 4.3: ATX connector

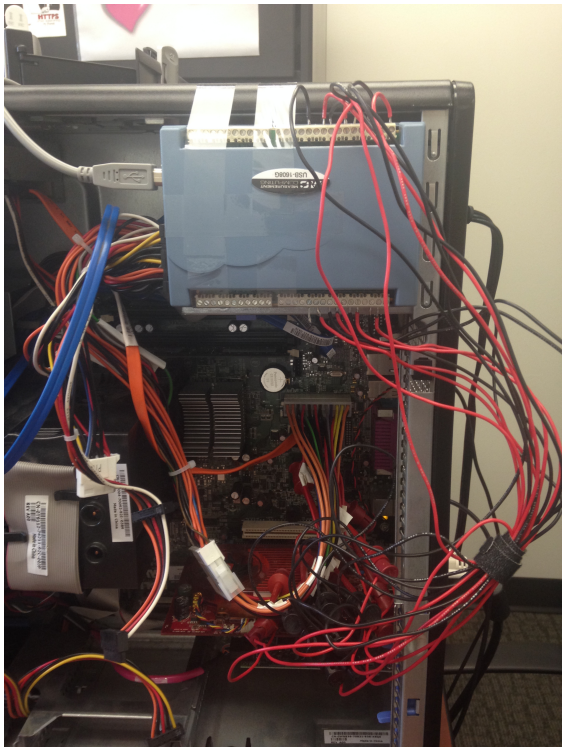

Figure 4.4: First configuration

The issue with the first hardware configuration was that the DAQ by itself collected only the voltage consumption, and since we were interested in power, both the voltage and current were required. To address this challenge, several options were studied in the second hardware configuration. The first option was to use a DC current switch and transducers, while a second option was to use a compact DC voltage and current sense print circuit board (PCB) with analog output.

The problem with the DC current switch and transducers was that the output for the samples were in milliampere (mA), while the DC voltage and current sense PCB provides the samples in amperes (A). Since we wanted to establish a difference between malicious and non-malicious behavior, having the samples in mA will be challenging because an mA is just a decimal fraction of an ampere. Also, the DC current switch and transducers were expensive in comparison with the compact DC voltage and current sense PCB with analog output. For these reasons, we decided to use the compact DC voltage and current sense PCB for collecting the

current consumption of the experimental machine.

The DC voltage and current sense PCB determines the DC current by measuring the voltage drop across a *shunt resistor*, and then converts that current to analog voltage output [6]. A shunt resistor is a device that allows electric current to pass around another point in the circuit by creating a low resistance path [19, 21]. For the second hardware configuration, the PCBs were welded to those wires on the ATX power extender cable that we were interested in monitoring (i.e., +3.3V, +5V, and +12V rails). Figure 4.5 shows the voltage and current sense PCB that was used for the second hardware configuration, while Figure 4.6 shows the second hardware configuration.

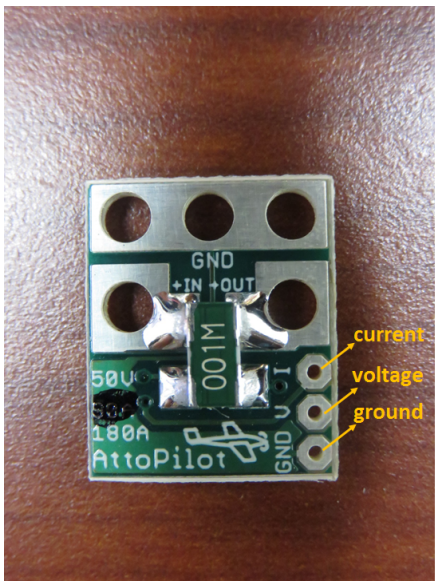

Figure 4.5: Voltage and current sense PCB

Figure 4.6: Second configuration

Using the second hardware configuration left a total of fourteen DC power channels to be monitored (eleven channels were used to measure the current and the other three channels were used to measure the voltage). That is, one voltage value for all the rails that were +3.3V, one value for all the rails that were +5V, and one value for all the rails that were +12V. While testing this configuration, we noticed that there were two +12V rails that were powering the CPU of the

experimental machine. These +12V rails were separate from the rails that we were already monitoring on the ATX power extender cable. Specifically, the +12V rails were connected from the PSU to a 4-pin ATX12V power connector on the motherboard. Including these rails, we ended up monitoring a total of sixteen channels. [1]

Monitoring sixteen channels at the same time was challenging because, when post-processing, we had to sum several measured currents together. To simplify the hardware configuration, we evaluated other options that could help us to reduce the number of channels to be monitored. After some exploring we found that all wires from the same voltage value were soldered together on the same contact point on the power supply. This means that all the +3.3V rails were connected to the same contact point, and the same was true for the +5V rails, and the +12V rails. Figure 4.7 shows the +12V rails soldered together on the same contact point in the power supply.

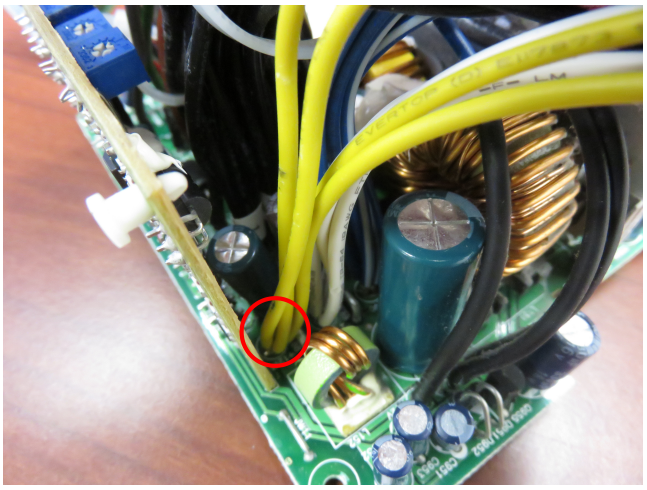

Figure 4.7: +12V rails soldered on the same contact point in the PSU

The third hardware configuration emerged from this observation. We grouped all the +3.3V rails on the same voltage and current sense PCB which was attached to the ATX power extender cable; the same was done for the +5V rails and the

[1] A survey of other machines was made to verify that general-purpose computers have the 4-pin ATX12V power connector. More than twenty computers were verified and all of them had the 4-pin ATX12V power connector.

+12V rails. Figure 4.8 shows the third hardware configuration used during the experiments and Figure 4.9 shows how the wires from the ATX power extender cable were attached to the DAQ.

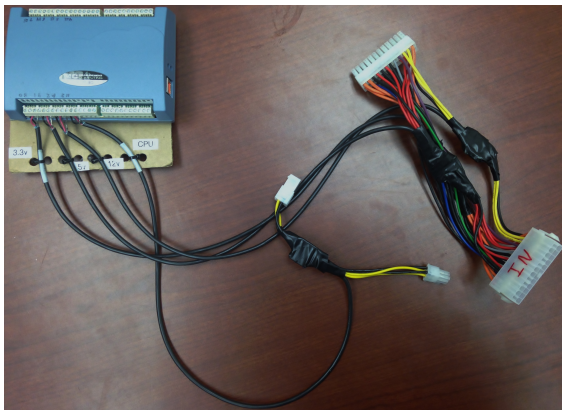

Figure 4.8: Third configuration

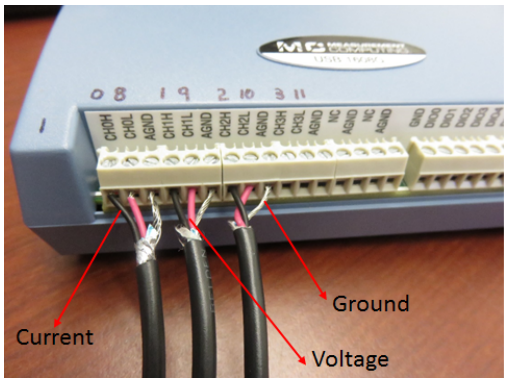

Figure 4.9: Wires attached to DAQ

Grouping the voltage rails reduced the numbers of channels to be monitored to six—three channels for measuring the current, and the other three channels for measuring the voltage. In addition, we also included the two +12V rails that power the CPU. Overall, instead of monitoring fifteen channels, we reduced the number to eight—four voltage channels and four corresponding current channels. This hardware configuration was the one used for the experiments and data collection described in this dissertation. It was chosen because it allowed us to obtain the same power consumption data as the second configuration, but with less monitoring channels.

4.1.2 Software Configuration

Initially, we used a tool called TracerDAQ Pro (version 2.3.1.0), which is an out-of-the box virtual instrument that acquires and displays power consumption data [20]. This tool ran on a different machine (data collection repository machine) in order to provide integrity during the data collection process. The acquired power consumption data from the experimental machine was stored as a comma-separated value (.csv) file on the data collection repository machine.

TracerDAQ Pro provides options such as strip chart, oscilloscope, function generator, and rate generator. When testing the first and second hardware configuration, we used only the strip chart option since we were interested in monitoring all eight channels simultaneously. For the first and second hardware configuration, power consumption data was collected using a sampling rate of 100Hz, a sampling interval of 0.01 seconds, and the data was collected for five minutes. At the end, 30,000 samples per channel were obtained.

As our experimental design evolved, we found that TracerDAQ Pro was not suitable for obtaining precise power consumption data. To address this issue we developed our own Visual Basic program. Our program was written in Visual Basic since the libraries (.dll files) from the DAQ were compatible with the Microsoft .NET framework. By using our software, the collected power consumption data was stored as a comma-separated value (.csv) file on a different machine (the data collection repository machine). The communication between the data collection repository machine and the experimental machine was established through the USB port. Figure 4.10 shows the graphical user interface (GUI) of our "DAQ Monitoring Tool" software.

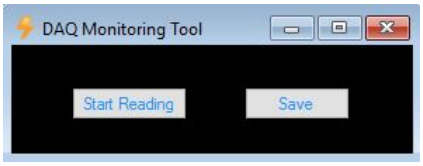

Figure 4.10: GUI for the "DAQ Monitoring Tool" software

To ensure the malware will not spread around the main network, we designed a segregated network, which consisted of the experimental machine, the data collection repository machine, a switch, and a cellular data connection. The data collection repository machine was connected to the personal hotspot, and then through a network switch the wireless connection was shared with the experimental machine. The advantages from the use of a segregated network are: (1) allowing the malware to behave normally, while avoiding the possibility of infect-

ing other machines on the network, and (2) allowing us to monitor and record the experimental machine's network traffic.

In addition to our "DAQ Monitoring Tool" software, Wireshark was used to collect the network traffic of the experimental machine with the objective of validating that the experimental machine was successfully infected with the rootkits being tested. As part of the network traffic analysis, we organized the protocols on the .pcap file by alphabetical order and then focused only on the column for the Domain Name System (DNS) protocol. From the domains that were captured, one of them got our attention (`term0l5ter12.com`). Interestingly, several websites [29, 8] had this domain registered as malicious. After all these analyses, we were certain that the experimental machine was successfully infected with the chosen rootkits.

4.2 Data Collection & Analysis

The power consumption of the experimental machine was collected in two different scenarios: non-malicious behavior (no rootkit running on the system) and malicious behavior (a rootkit was running on the system). For the data collection workflow, we assumed a clean installation of Windows, then power consumption data was collected and labeled as non-malicious. Subsequently, the experimental machine was infected and power consumption data was collected and labeled as malicious. For this case study we infected the experimental machine with two rootkits: Alureon and Pihar.

The first rootkit, Alureon, also known as TDL4 or TDSS, is a Trojan that allows an attacker to intercept incoming and outgoing Internet traffic in order to gather confidential information such as user names, passwords, and credit card data [10]. There are several generations of this type of malware, and for our experiments, we used the fourth generation [28]. Typically, it infects a computer via drive-by download through a questionable website, often a distributor of pornography or pirated media [246]. Once Alureon is installed on the machine, the software

searches the system for any competitor's malware and removes it. It also uses an encryption algorithm to hide its communications from traffic analysis tools that are sometimes used to detect suspicious transmissions [246]. Furthermore, this rootkit can manipulate the master boot record (MBR) of the computer to ensure that it is loaded early during the bootup process so that it can interfere with the loading of the OS [9]. The second rootkit, which is a variant of Alureon, is a Trojan called Purple Haze (also known as Pihar). Like Alureon, this rootkit can modify the MBR of the machine, as well as changing system settings and reconfiguring the Windows Registry. Its rootkit capabilities include disabling the anti-virus (AV) software to keep itself hidden [24].

To initiate the data collection process, we wrote two scripts: a Python script that executes a sequence of events, and a C++ program that inserted what we will called a marker. The objective of the Python script was to ensure repeatability, while the objective of the marker was to insert a signal into the measured power consumption data to mark the start and end points for each sequence of events (i.e., idle, opening IE, and booting/rebooting).

When the Python script is executed, it launches two markers before the experimental machine goes idle for a minute. Then, the Python script opens ten windows of Internet Explorer (IE) each with five seconds delay. IE was chosen because the Alureon and Pihar rootkits affect the performance of browsers [15, 245]. Figure 4.11 shows the sequence of events during the data collection process for the +12V CPU rails. The events (idle, opening IE, booting/rebooting) were recorded during three states: (1) prior to infection, (2) after infection, and (3) after infection plus reboot. In order to segment these sections of the power profile, we used the marker to stress the CPU of the experimental machine for five seconds. The Python script places markers in the power consumption data before and after the events were recorded. The advantage of using these markers is that they allow us to understand when a particular event occurs and how long it takes to complete its execution. This workflow was completed, for each rootkit, three times for the four monitored voltage rails.

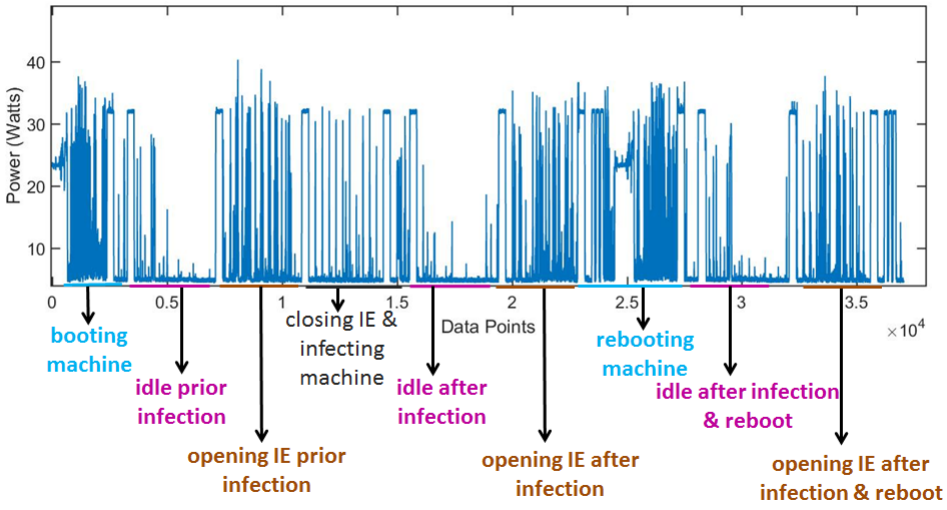

Figure 4.11: Sequence of events during the data collection process

After the data collection, the power consumption data was pre-processed by multiplying the voltage and current for the monitored rails. Then, we wrote a Matlab script to separate the events based on their start and end point for further analysis. The MATLAB script returns the start and end point of all the markers that appeared on the dataset. For this case study, there were a total of eighteen markers. Once we had the start and end point for each event, the next step was to compare those events that were related to each other. Specifically, we were interested in the following comparisons: (1) when the machine was booting prior to infection versus when the machine was rebooting after infection; (2) idle prior to infection versus idle after infection; (3) idle prior to infection versus idle after infection and reboot; (4) when opening IE windows prior to infection versus when opening IE windows after infection; and (5) when opening IE windows prior to infection versus when opening IE windows after infection and reboot.

4.3 Preliminary Results

The hypothesis under investigation in this proof of concept was if there is a difference in the power consumption of a general-purpose computer after malware

(i.e., rootkit) infection. To prove or disprove this hypothesis, several experiments were conducted and power profiles were collected for specific events (i.e., idle, opening IE, and booting/rebooting). This was done for the rootkits Alureon and Pihar. For each rootkit there were three datasets. Each dataset contains the power consumption obtained for each monitored voltage rail. The comparison between the non-malicious and malicious state was done for each of the events that were recorded on the four monitored voltage rails. Five graphs were generated for each monitored voltage rail. The x axis for each of these graphs shows "Data Points", which refers to the total of power readings that were sampled every 10 milliseconds. For example, if a graph shows $3,000$ data points that would be equivalent to thirty seconds.

4.3.1 +3.3V Rails

These rails are typically used by digital electronic components and circuits in the system, such as memory. When comparing the power profiles of booting prior to infection versus when it was rebooting after infection, we noticed that at the beginning the power consumption was lower and subsequently both events kept their power consumption similar to each other. Regarding the other events (i.e., idle and opening IE), results showed that the difference in the power consumption cannot be established by the naked eye. After analyzing all six datasets (i.e., three datasets per rootkit), we concluded that the +3.3V rails were not very useful for detecting different behaviors between the non-malicious and malicious power profiles because these voltage rails are used to power up memory, and that component does not consume as much power as the hard disk drive or CPU.

4.3.2 +5V Rails

For all datasets, when comparing booting prior to infection with the rebooting after infection for the +5V rails, we noticed the same behavior as the +3.3V rails, that is the power consumption after infection was lower at the beginning of the

initialization process, but later it kept the same pace as the non-malicious behavior. Hence, comparing booting prior to infection versus booting after infection for the +5V rails is not sufficient to distinguish between non-malicious and malicious behavior.

When we compared idle prior to infection versus idle after infection with Alureon we obtained an increment in the power consumption after the experimental machine was infected for two out of the three datasets (66.67% of the time), while for Pihar we noticed an increment in the power consumption for all datasets (100% of the time). However, when comparing idle prior to infection versus idle after infection and reboot for both rootkits, we noticed that the power profiles for both scenarios (malicious and non-malicious) were at the same level. In other words, a distinguishable difference cannot be made by the naked eye. Furthermore, when comparing all the graphs in which the experimental machine was idle we noticed a delay in the power consumption data after the experimental machine was infected. We believe this delay is because after the infection more processes are running and this extra work consumes more power. Figure 4.12 shows the power consumption after infecting the experimental machine with the Alureon rootkit. As can be seen from Figure 4.12, the power consumption in the idle state was higher after the infection than prior to infection. Hence, this comparison is a good criterion for detecting malware through the power consumption.

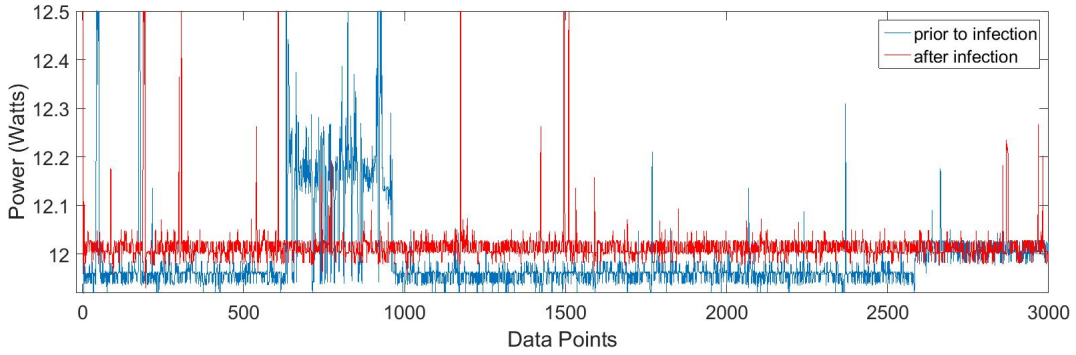

Figure 4.12: Power consumption for idle prior to infection vs. idle after infection with Alureon for the +5V rails

When IE was opened prior to infection versus after the infection with Alureon, we noticed an increment in the power consumption after infection for two out of three datasets (66.67% of the time). In the case of the Pihar rootkit, this behavior was seen only in one out of three datasets (33.33% of the time). Figure 4.13 shows the power consumption after opening IE prior to infection versus after infection for the Alureon rootkit.

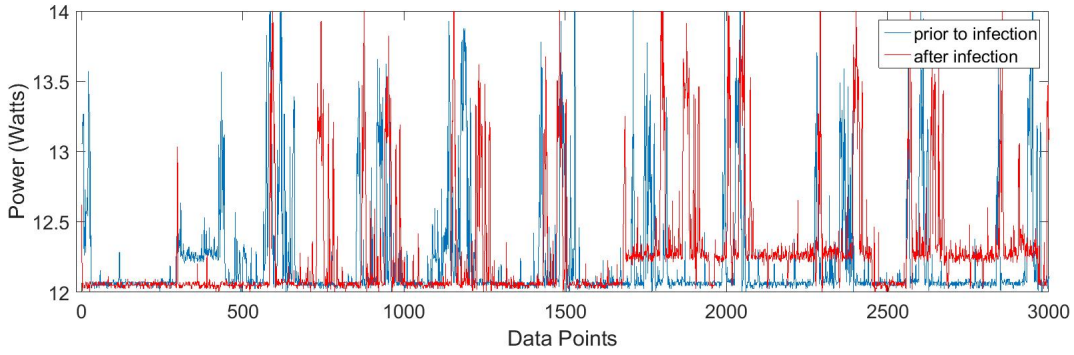

Figure 4.13: Power consumption for opening IE prior to infection vs. opening IE after infection with Alureon for the +5V rails

From Figure 4.13 we can see an increment in the power consumption when some IE windows were opened. Interestingly, this increment was seen when some windows of IE were jammed. This was consistent with the behavior we saw during the data collection process and later was confirmed when analyzing the .pcap file. Based on network traffic data, we noticed that Alureon was trying to redirect the search engine to advertisement websites. However, when IE was opened prior to infection versus after the infection and reboot for both rootkits, a difference by the naked eye could not be established.

4.3.3 +12V Rails on the Motherboard

The +12V rails on the motherboard are used to power up the hard disk drive motors and the fans. For one of the Alureon datasets results showed that the power consumption was higher after the infection compared to when it was booted prior to infection (33.33% of the time). However for the other two datasets, we

saw similar behavior as in the case of +3.3V and +5V rails. Figure 4.14 shows an increment in the power consumption after the experimental machine was infected with Alureon during the initialization process. In the case of Pihar, an increment in the power consumption was noticeable on two out of three datasets (66.67% of the time).

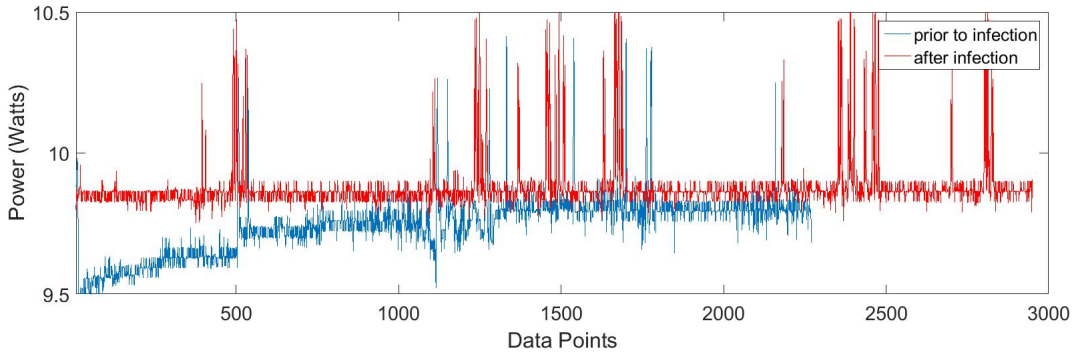

Figure 4.14: Power consumption for booting prior to infection vs. booting after infection with Alureon for the +12V rails on the motherboard

When comparing the idle state (idle prior to infection versus idle after infection and idle prior to infection versus idle after infection and reboot), results for Alureon showed an increment in the power consumption after infection for two out of the three datasets (66.67% of the time). Similar increment was seen in all three datasets of Pihar (100% of the time). Figure 4.15 shows an increment in the power consumption when comparing idle prior to infection versus idle after infection and reboot for the Alureon rootkit.

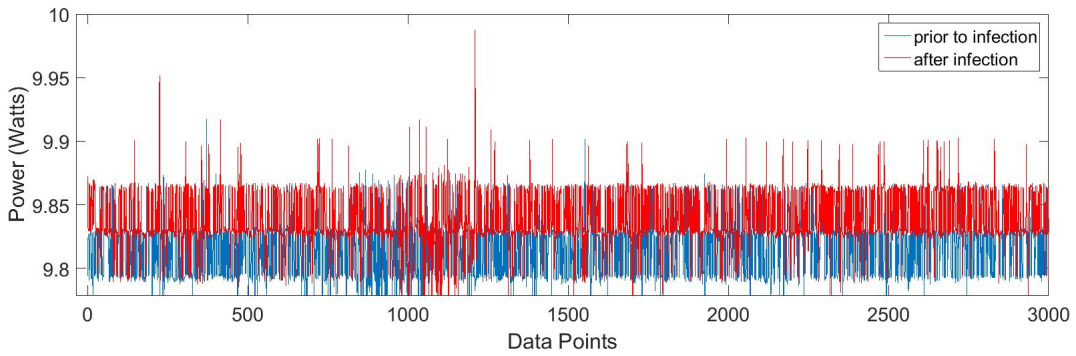

Figure 4.15: Power consumption for idle prior to infection vs. idle after infection and reboot with Alureon for the +12V rails on the motherboard

Nonetheless, when comparing IE (IE prior to infection versus after infection and IE prior to infection versus after infection and reboot), results for Alureon showed that an increment in the power consumption after infection can be seen in only one of the datasets (33.33% of the time). Figure 4.16 shows an increment in the power consumption when comparing IE prior to infection versus IE after the Alureon infection and reboot.

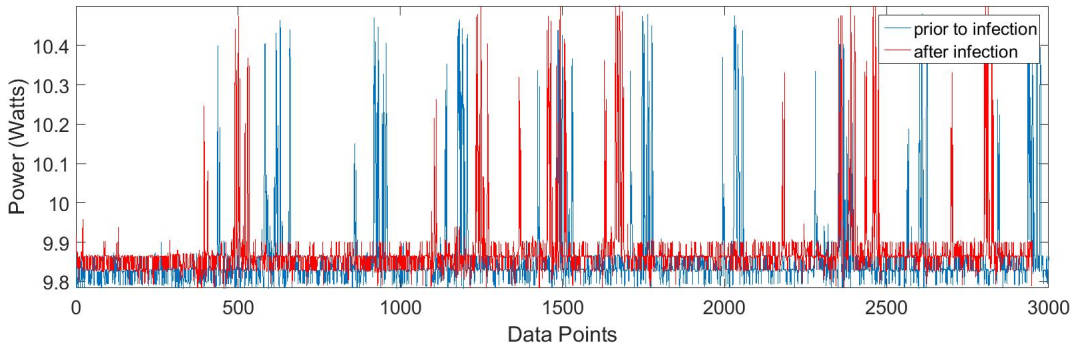

Figure 4.16: Power consumption for opening IE prior to infection vs. opening IE after infection and reboot with Alureon for the +12V rails on the motherboard

When comparing IE prior to infection versus IE after infection for Pihar, we noticed an increment in the power consumption after infection for one out of the three datasets (33.33% of the time). Interestingly, when comparing IE prior to infection versus IE after infection and reboot we noticed the power consumption of the experimental machine was higher after infection for all datasets (100% of

the time).

After analyzing the +12V rails on the motherboard, we concluded these rails are useful when distinguishing the non-malicious and malicious power profiles.

4.3.4 +12V CPU Rails

The +12V CPU rails are separate from the +12V rails on the motherboard (monitored in the PSU). They are used to power the CPU or graphics processing unit (GPU) of a general-purpose computer. The +12V rails on the motherboard are used to power hard disk drive motors and fans.

The comparison between the power consumption when the experimental machine was booting prior to infection versus when it was booting after infection showed that at the beginning of the initialization process the power consumption was higher prior to infection for both rootkits. However, at some point during the initialization, an increment in the power consumption after infection was noticeable. This comparison by itself does not provide information that can help us to distinguish between non-malicious and malicious behavior because of the presence of noise. Noise is expected during the booting and rebooting process because the system is executing several processes simultaneously, so even if the malware is present, its challenging to differentiate between non-malicious and malicious states.

In the case of idle (idle prior to infection versus after infection and idle prior to infection versus after infection and reboot), we noticed that the power consumption for both rootkits in the non-malicious and malicious scenarios were similar. However, there were some higher spikes after infection. We believe these spikes were generated when the system was executing non-malicious processes. Similarly, these spikes were also seen in the +5V rails.

A similar behavior was noticeable during IE execution (IE prior to infection versus after infection and IE prior to infection versus after infection and reboot). Results showed that for both non-malicious and malicious power profiles, the power consumption was similar. In addition, some delays were seen on the experimen-

tal machine after it was infected. Figure 4.17 shows the power consumption for opening IE prior to infection versus opening IE after infection for the Alureon rootkit.

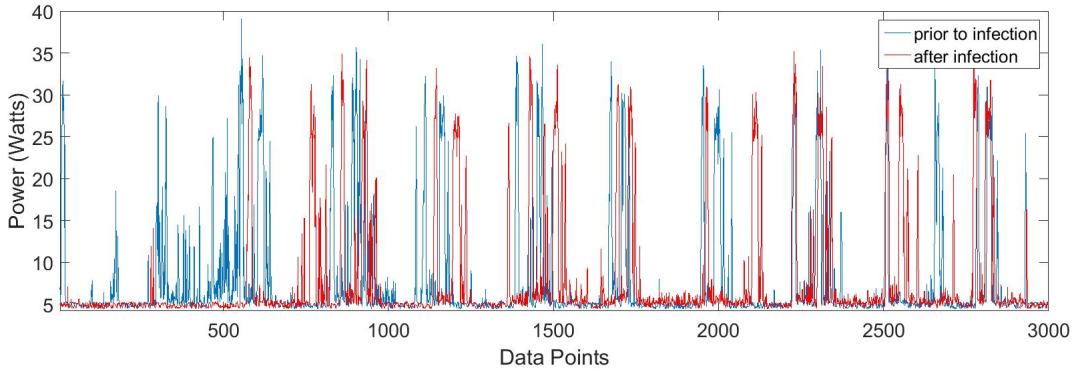

Figure 4.17: Power consumption for opening IE prior to infection vs. opening IE after infection with Alureon for the +12V CPU rails

After analyzing all six datasets (three datasets per rootkit), we concluded that a distinguishable difference cannot be made by the naked eye when analyzing the non-malicious and malicious power profiles for the +12V CPU rails. These results are not the ones we expected, because by monitoring the CPU of the general-purpose computer we thought these voltage rails would be more informative. However, we are aware that many processes are running and this extra work consumes more power making it difficult to establish a difference by the naked eye. Furthermore, using power consumption to distinguish malware from non-malicious software can be done by using machine learning techniques, which is explored in this dissertation.

Chapter 5

Experimental Set-up & Data Collection

Based on what we learned from experiments presented in Chapter 4, in this chapter we describe the software tools used in our testbed and explain the conducted experiments when collecting data from multiple modalities (i.e., power consumption, network traffic data, and system logs). Note that the data collected in this Chapter was used for the experiments described in Chapters 8, 9, and 10.

5.1 Testbed Design & Development

5.1.1 Hardware & Software Configuration

Although we used the same hardware configuration that is described in Chapter 4, the software configuration of our testbed was modified to collect the power consumption, the network traffic data and system logs simultaneously. We designed a segregated network (described in Subsection 4.1.2), which consisted of the experimental machine, the data collection repository machine, a switch, and a cellular data connection, as shown in Figure 5.1. The data collection repository machine was connected to the personal hotspot, and then through a network switch the wireless connection was shared with the experimental machine. The experimental machine had unfiltered Internet access, which is crucial for the malware samples

to perform their full functionality, as most malware initiates network traffic (e.g., contacts the command and control servers).

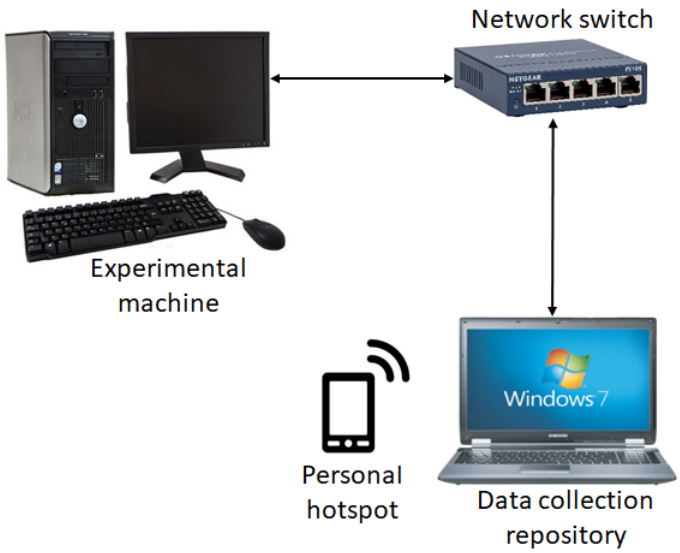

Figure 5.1: Experimental set-up

Several software tools were used for the experimental design. We used ClockSynchro [33] to synchronize the clocks of the experimental machine and the data collection repository machine, Wireshark [42] to collect the network traffic data, CaptureBAT [32] to collect the system logs, and the Clonezilla application [14] to ensure that the hard disk drive of the experimental machine contained a clean (i.e., uninfected) copy of the Windows OS.

5.2 Data Collection Set-up

5.2.1 Dynamic Behavioral Data Collection

To conduct the dynamic analysis, the executable files were launched manually in a Dell OptiPlex 755 computer with a clean installation of 32-bit Windows 7 Ultimate. The objective of the dynamic analysis was to collect behavioral-based features (i.e., power consumption, network traffic, and system logs), while running different malware and non-malicious applications on a general-purpose computer.

The instrumentation used in our experiments to collect the power consumption data was a DAQ, Model Number: USB-1608G Series [20]. The DAQ was attached to the experimental machine through a 24-pin ATX-extender cable. The 24-pin male connector from the ATX-extender cable was attached to the motherboard, while the 24-pin female connector was attached to the PSU. Eight DC power channels—four voltage channels and four corresponding current channels were monitored (+3.3V rails, the +5V rails, the +12V rails on the motherboard and the +12V rails on the CPU). To collect precise power measurements, we developed a program that directly accessed the DAQ to read the power consumption data and stored them on a separate machine, used as data repository. A detailed description about the hardware configuration used to collect the power consumption data can be found in Chapter 4 and in our technical report [172].

We also used Wireshark [42], which ran on the data collection repository machine, to collect the network traffic data. In the case of the system logs, several software applications (i.e., Event Viewer [208], RegFsNotify [112], Logstash [37], and CaptureBAT) were evaluated.

Event Viewer [208] is a tool that allows the user to monitor the events that occurred in the system, it maintains system logs about programs, security, and system events on the computer, can be used to view and manage the system logs and to gather specific information (e.g., hardware and software problems) [208]. Because the recorded events did not included the milliseconds on their timestamps, this tool was discarded. Note that precise timestamps are essential since we are interested in correlating the system logs with the power consumption and the network traffic data.

Similarly, RegFsNotify [112] detects the changes that occurred in the Windows Registry and file system in real time, but since no timestamp was recorded for the collected events it was discarded. On the other hand, Logstash [37] (an open source server-side tool for managing events and logs) was also discarded because by having a client-server architecture additional software is required to run on the background of the operating system (OS), which causes additional noisiness

in the power consumption data. Finally, we evaluated CaptureBAT [263, 32], a lightweight open source tool that logs the changes that occurs in the OS when an application is running. This tool allows to record changes that occur in the file system, Registry, and system processes. The file system monitor captures system details such as when an event occurs, the type of event (i.e., read and write), and the name of the process that triggered that particular event. The Registry monitor reports the time with a resolution in milliseconds, the process that triggered the registry event, the path to the key where the action occurred, and the type of action that was performed on the key (i.e., created and deleted). The process monitor pays attention to the creation and destruction of processes but does not report on the running processes. It captures the time, whether the process was created or terminated, and the file name that represents that particular process. This tool was chosen among the other tools, because it has an exclusion list mechanism that allows to omit noise that occurs naturally in the system. Moreover, this tool has also been recommended for conducting dynamic malware analysis [143] and has been used by previous works for rootkit detection [273] and for malware analysis in memory forensics [284].

To account for the randomness of different Windows OS background processes, each malware and non-malicious software application was executed three times. Each run lasted for thirty minutes. Other works that used behavioral characteristics for malware detection have executed the malware in a controlled sandbox environment for one minute and a half [178], two minutes [251], five minutes [95, 77, 303, 99] and twenty minutes [114]. We decided to run our malware samples for thirty minutes because for these specific malware examples this was sufficient time for them to perform malevolent actions. Note that for each thirty minutes run, we collected one .csv file with the power consumption data, one .pcap file with the network traffic data, and one text file (.txt) with the system log data.

5.2.2 Code-based Static Data Collection

Static data can be collected without executing the portable executable file. Specifically, the portable executable files has to be decompiled first. To decompile Windows executables, disassembler and memory dumper tools can be used. Some examples of tools commonly used to disassemble Windows executables are: IDAPro [35], OllyDbg [38], ExifTool [152], and PE Explorer [51]. From these tools, we used a disassembler tool called PE Explorer [51]. PE Explorer is an integrated collection of tools that provide a framework for working with several executable formats that run on Windows 32-bit platforms. After unpacking the executable file, information such as headers, data directories, and dynamic linked libraries (DLL) dependencies were collected from each malicious and non-malicious software application.

5.3 Malicious and Non-malicious Applications

5.3.1 Malicious Software Selection

Malware examples were obtained from two malware repositories: (1) Contagio (`http://contagiodump.blogspot.com/`) and (2) VirusShare (`https://virusshare.com/`). To download the malware examples from the former website (Contagio), no account was needed. However, in the case of the latter (virusShare), we had to contact the website administrator and request access to the malicious software. Table 5.1 lists the fifty one malware examples chosen for our experiments. Note that those malware examples shown in bold in Table 5.1 are variants from another malware that was also chosen for our experiments. For example, Carberp V1 is a variant of Carberp and so on.

Table 5.1: Malicious applications chosen for the experiments

#	***Malware Examples***
1	Alureon
2	Avatar
3	Bangat
4	Biscuit
5	Carberp
6	**Carberp V1**
7	Citadel-Atmos
8	Cookiebag
9	CryptoLocker
10	**CryptoLocker V1**
11	Dairy
12	DarkMegi
13	Dexter
14	Emotet-K
15	Emotet-Conficker trojan
16	Filecoder
17	Greencat
18	Kovter-Zcryptor
19	Locky
20	MaxRootkit
21	Miniasp
22	Necurs
23	Newsreel
24	Pihar
25	Crisis
26	Rustock

#	***Malware Examples***
27	Satan
28	**Satan V1**
29	**Satan V2**
30	Sirefef
31	Tabsgsql
32	Tarsip-Eclipse
33	Tarsip-Moon
34	Teerac-A
35	Tescrypt-A
36	**Tescrypt-A V1**
37	Tescrypt-D
38	Tescrypt-J
39	Warp
40	Web-C2-AUSOV
41	Web-C2-BOLID
42	Web-C2-CSON
43	Web-C2-DIV
44	Web-C2-HEAD
45	Web-C2-KT3
46	Web-C2-QBP
47	Web-C2-RAVE
48	Xpaj
49	Zbot
50	**Zbot V1**
51	**Zbot V2**

Each malicious software was executed in a virtual machine to ensure that it was not corrupted and that it was functional for the Windows OS. We also used a malware analysis service [26] to generate a behavioral report for each malicious file. These behavioral reports were used to ensure the malware chosen for our experiments was entirely removed from the hard disk drive after formatting the

experimental machine using the Clonezilla tool.

Malware was executed manually to infect the experimental machine. We used great care to allow malware to behave as intended. For each malware example, we ensured that it was active by monitoring the network traffic and by observing events such as files being encrypted, pop-ups with adult content, etc. The malware selected for our experiments have traits of of viruses (e.g., Dexter [23]), worms (e.g., Gamarue [22]), backdoors (e.g., Greencat [25]), rootkits (e.g., Alureon [294]), and ransomware (e.g., Locky [49]). Different types of malware were used in our experiments to have a representative, diverse malware sample. Distinguishing among different malware types and/or malware families are beyond the scope of this dissertation.

5.3.2 Non-malicious Software Selection

We used different types of non-malicious applications (some network intensive, other CPU and memory usage intensive). We launched each non-malicious software manually and ensured that it was provided with adequate inputs/workloads. For example, we used Firefox [46] to navigate the Internet which generated network traffic while IntelBurnTest [48] and HeavyLoad [47] were used to stress the CPU of the experimental machine.

During the data collection process, six out of the twenty eight non-malicious software applications (i.e., benchmark tools) were discarded because the duration of the benchmarking test was less than a minute. Therefore, the remaining twenty two non-malicious applications were chosen for the data analysis. A list of the twenty two non-malicious software applications is given in Table 5.2. Note that the discarded non-malicious software applications are shown in bold.

Table 5.2: Non-malicious applications chosen for the experiments

#	***Non-malicious Applications***
1	Adobe Reader
2	ATTO-Disk
3	**BlackHole-B1**
4	**BlackHole-B2**
5	**BlackHole-B3**
6	CPUID-CPU-Z
7	CPUStress
8	Firefox
9	FurMark-CPUBurner
10	GeekBench
11	HeavyLoad-StressCPU
12	HeavyLoad-StressMemory
13	HeavyLoad-TreeSize
14	HeavyLoad-WriteTempFile

#	***Non-malicious Applications***
15	**HyperPi**
16	IntelBurnTest
17	KLite
18	MaxMemm
19	Notepad
20	Opera
21	ParticleFury
22	Spotify
23	StressMyPC
24	**UserBenchmark**
25	VLC
26	**WebServerStress**
27	Windows Media Player
28	XtremeBenchmark

Chapter 6

Data Pre-processing & Feature Extraction

This chapter explains the data pre-processing and how the dynamic behavioral-based and code-based static features were extracted for the malicious and non-malicious software applications.

6.1 Dynamic Behavioral-based Features

Dynamic methods require the execution of a given malware example, typically in a sandbox environment [305, 275], and extract behavioral-based features that represent the actions performed by the malware. Although the usage of dynamic behavioral-based features is more costly, they are more resilient to obfuscation because they extract behavior actions performed by the malware, rather than binary code patterns. Therefore, using dynamic behavioral-based features is suitable for detecting new malware examples and variants of existing malware. In this dissertation, the extracted dynamic behavioral-based features are divided into three categories: (1) power-based features, (2) network traffic-based features, and (3) system logs-based features. A total of three hundred and forty five dynamic behavioral-based features (i.e., one hundred and thirty two power-based features, ten system logs-based features, and two hundred and three network traffic-based

features) were extracted.

After the data was collected, the power consumption, system logs, and the network traffic data were pre-processed. To extract the desired behavioral features from the power consumption, system logs, and network traffic data, we developed our own Python script. With respect to the system logs and network traffic data, each .txt file and .pcap file were converted to the .csv format for further analysis. In the case of the power consumption data, a data conversion was not necessary because the data was already in the .csv format. Furthermore, we developed another Python script to calculate the autocorrelation function (ACF) for the power consumption and network traffic data. In the case of the system logs, we could not calculate the autocorrelation function because CaptureBAT collects only the changes that occurs in the system after the execution of the malicious and non-malicious software. Meaning there was not a specific time interval when collecting the system logs, unlike with the power consumption and network traffic data.

The autocorrelation function is the coefficient of correlation between two values in a time series [43]. Informally, autocorrelation can be defined as the similarity between observations as a function of the time lag between them. The ACF for a time series y_t can be defined as:

$$Corr(y_t,\ y_{t-k}),\ k = 1,\ 2,\ \ldots\ n$$

where k refers to the time gap being considered as the lag. A lag whose $k = 1$ refers that the autocorrelation between the values is one time period apart. The analysis of autocorrelation is helpful for finding repeating patterns, such as the presence of a periodic signal obscured by noise. In our Python script we calculated the autocorrelation function for different lags (i.e., when $k = 5, 10, 15, 20, 25$, and 50).

6.1.1 Power Consumption

Power consumption data was pre-processed by multiplying the voltage and current for each of the monitored voltage rails to obtain the power consumption in Watts.

To this end, we used a bash script that multiplies the current and voltage for each of the monitored voltage rails. The extracted power-based features are given in Table 6.1. Note that for the autocorrelation-based features in Tables 6.1 and 6.2, if $k = n$, we would have a total of n values for k. For example, if $k = 5$, we have 5 values that were used as an autocorrelation-based feature. This is why we have a total of 125 autocorrelation-based features ($5 + 10 + 15 + 20 + 25 + 50 = 125$) for each modality (i.e., power consumption and network traffic data).

Table 6.1: List of extracted power-based features

Feature name	***Description***
PwrMinimum	Minimum power measurement
PwrMaximum	Largest power measurement
PwrMean	Average power measurement
PwrMedian	Median power measurement
PwrVariance	Variance power measurement
PwrSkewness	Skewness power measurement
PwrKurtosis	Kurtosis power measurement
AC_Pwr_5	Autocorrelation values for $k = 5$
AC_Pwr_10	Autocorrelation values for $k = 10$
AC_Pwr_15	Autocorrelation values for $k = 15$
AC_Pwr_20	Autocorrelation values for $k = 20$
AC_Pwr_25	Autocorrelation values for $k = 25$
AC_Pwr_50	Autocorrelation values for $k = 50$

6.1.2 Network Traffic

Analyzing network traffic data helps us to understand about what is happening on the network. Network traffic-based features are convenient for malware detection, since unusual amount of traffic in a network is a possible sign of a cyber-attack. In this dissertation, the extracted network traffic-based features can be divided into two categories: (1) commonly used network traffic-based features (e.g., number of received packets, number of unique source IP address, etc.); and (2) network flows-based features. The extracted commonly used network traffic-based features are listed in Table 6.2, while the extracted network flows-based features with their corresponding aggregation levels are given in Table 6.3.

Table 6.2: List of extracted commonly used network traffic-based features

Feature name	***Description***
Packets	# of received packets
PktsLength	Packets length in bytes
UniqueSourceIP	IP address of the device sending the packet
UniqueDestIP	IP address of the device receiving the packet
LLMNR	# of packets related to LLMNR protocol
UDP	# of UDP protocol packets
ARP	# of ARP protocol packets
BROWSER	# of BROWSER service packets
NBNS	# of NBNS service packets
DHCP	# of DHCP protocol packets
DHCPV6	# of DHCPV6 protocol packets
DNS	# of DNS protocol packets
HTTP	# of HTTP protocol packets
ICMP	# of ICMP protocol packets
ICMPV6	# of ICMPV6 protocol packets
IGMPV3	# of IGMPV3 protocol packets
SSDP	# of SSDP protocol packets
TCP	# of TCP protocol packets
AC_Ntwk_5	Autocorrelation values for $k = 5$
AC_Ntwk_10	Autocorrelation values for $k = 10$
AC_Ntwk_15	Autocorrelation values for $k = 15$
AC_Ntwk_20	Autocorrelation values for $k = 20$
AC_Ntwk_25	Autocorrelation values for $k = 25$
AC_Ntwk_50	Autocorrelation values for $k = 50$

Table 6.3: List of extracted network flows-based features

Feature name	***Description***	***Aggregation levels***
Flows	# of flows	None
Duration	Time communication lasted	Max, Avg
L4ProtoUDP	# of flows related to UDP	None
L4protoIGMP	# of flows related to IGMP	None
L4ProtoTCP	# of flows related to TCP	None
L4ProtoICMP	# of flows related to ICMP	None
PktsSent	# of transmitted packets	Sum, Max, Avg
PktsRcvd	# of received packets	Sum, Max, Avg
BytesSnt	# of transmitted bytes	Sum, Min, Max, Avg
BytesRcvd	# of received bytes	Sum, Max, Avg
MinPktSize	Minimum layer 3 packet size	Min
MaxPktSize	Maximum layer 3 packet size	Max
AvgPktSize	Average packet load ratio	Avg, Median
StdPktSz	Standard deviation packet load ratio	Std
Pktps	Packets sent per second	Max, Avg
Bytps	Bytes sent per second	Max, Avg
PktAsm	Packet stream asymmetry	Min, Avg
BytAsm	Byte stream asymmetry	Min, Avg
TcpPSeqCnt	TCP packet sequence count	Max, Avg
TcpSeqSntBytes	TCP sent sequence diff bytes	Max, Avg
TcpSeqFaultCnt	TCP sequence # fault count	Max, Avg
TcpPAckCnt	TCP packet ack count	Max, Avg
TcpFlLAcRcByt	TCP flawless ack received bytes	Max, Avg
TcpAckFaultCnt	TCP ack # fault count	Max, Avg
TcpInitWinSz	TCP initial window size	Max, Avg
TcpAveWinSz	TCP average window size	Avg, Median
TcpWinSzDwCn	TCP window size change down count	Max, Avg
TcpWiSzUpCnt	TCP window size change down count	Max, Avg
TcpWiSzChDiCn	TCP window size direction change count	Max, Avg
FlowDirA	Flows direction is clnt to srvr	None
FlowDirB	Flows direction is srvr to clnt	None
AvgIAT	Average of IAT	Avg, Median
StdIAT	Standard deviation of IAT	Std

Literature shows several definitions for network flows [104, 105, 198]. In this dissertation we follow the definition given by Claise [105], which describes a network flow as "a set of IP packets passing an observa-

tion point in the network during a certain time interval, such that all packets belonging to a particular flow have a set of common properties". Specifically, a network flow is defined using the following 5-tuple:

(ip_src, ip_dest, port_src, port_dst, and proto)

where *ip_src* refers to the source IP address, *ip_dest* refers to the destination IP address, *port_src* refers to the source port number, *port_dst* refers to the destination port number, and *proto* refers to the used protocol.

There exists a variety of tools that can be used to extract network flows from a .pcap file. Some examples of these tools are FlowScan [2, 232], NetFlow [104], NetViewer [183], Netpy [130], Softflowd [7], TCPflow [3], SpliCap [5], and Tranalyzer [52], which extends Cisco NetFlow's [104] functionality. In this dissertation we used Tranalyzer [52, 96], a lightweight unidirectional flow exporter that collects packet information with common characteristics, such as IP addresses and port numbers, to obtain the network flows. This tool was chosen over the others for three reasons: (1) It is an extension of NetFlow [104] which has been widely used by the research community as a flow exporter (aggregates packets into flows and export flow records) and as a flow collector (storage and pre-process flow data); (2) It supports features that can be categorized into groups (e.g., time, inter-arrival, packets, etc.); (3) It has been used by previous works [148, 149] for detection of botnets.

Since multiple flows (e.g., > 100) were generated for each individual .pcap file, we performed aggregation of the network flow-based features to achieve our final network flows-based features. Aggregation is commonly used to get additional information about particular groups based on specific characteristics. In addition, it also helps to achieve a coarser granularity in the data.

6.1.3 System logs

System logs or syslogs are files that contain events that are logged by components from the OS. Extracting system logs-based features is useful because they

contain information about the software, hardware, system processes and system components as well as information, such as error and warning events related to the computer OS. The extracted system logs-based features (given in Table 6.4) were based on the changes that occurred in the file system, Registry, and system processes while the malicious and non-malicious software were executed.

Table 6.4: List of extracted system logs-based features

Feature name	***Description***
Changes	# of changes that occurred in the system
FileChgs	# of changes in the file system
RegistryChgs	# of changes in the Registry
ProcessesChgs	# of changes in the process manager
FlsWrite	# of written files
FlsDelete	# of deleted files
CreatedPrcs	# of created processes
TerminatedPrcs	# of terminated processes
SetValueKeyChgs	# of times the value entry under the open key was replaced/created
DelValueKeyChgs	# of times a method deleted a value entry under the open key

6.2 Code-based Static Features

Common malware analysis techniques initiate by conducting static analysis. Static analysis describes the process of analyzing the code or structure of a program to determine what it does without the need of executing it. These methods can be applied to detect known malware with high accuracy and speed, but they are susceptible to code obfuscation which is a common practice of malware creators.

In order to conduct static malware analysis, the portable executable (PE) file format is used. A PE file consist of a number of headers and sections that tell the dynamic linker how to map the file into memory, while the PE file format is a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code [271]. Typically, every file with executable code that is loaded by the Windows OS is in the PE file format. The structure of these files begin with a header that includes information about the code, the type

of application and dynamic library references for linking, application programming interface (API), export and import tables, resource management data and thread-local storage (TLS) data.

In this dissertation, we extracted the code-based static features using the PE Explorer tool [51]. The extracted code-based static features were in the .txt format, so we converted them to the .csv format for further analysis. To pre-process the data, we used a bash script to remove punctuation marks (i.e., the apostrophes and semicolons) and redundant information (i.e., the name of the computer used to collect the data and the timestamp in which the file was created).

After the data pre-processing, we developed our own Python script to extract the code-based static features. The extracted code-based static features can be divided into three categories: (1) headers-based features, (2) data directories-based features, and (3) DLL dependencies-based features. Since some of our code-based static features were in hexadecimal, we developed another Python script to convert these hexadecimal values to decimal before the data analysis. A total of forty eight code-based static features (i.e., fourteen header-based features, eight data directories-based features, and twenty six DLL dependencies-based features) were extracted.

The extracted headers-based features contains information about the number of sections, the size of the stack and heap, and so on. This type of information is of great value to the malware analyst since it can be obtained easily without the need of executing the malicious file [271]. A list of the extracted headers-based features is given in Table 6.5.

Data directories indicates how a specific section body's data is structured on the executable file. The data directories contain references to various tables (e.g., import, export, resource, etc.). These references (if appropriately analyzed) can provide valuable insight to malware analysts, since they provide a summary of the contents of the PE file [265]. Specifically, each data directory structure specifies the size and relative virtual address of the directory. The data directories contain information such as the resources, debugging information, base relocations and

other OS specific data. Table 6.6 lists the extracted data directories-based features.

Table 6.5: List of extracted headers-based features

Feature name	***Description***
Sections	Size of the section table
Magic	Integer identifying the sate of the image file
CodeSz	Size of the executable code
InitializedDataSz	Size of the initialized data
EntryPointAddr	Address of entry point when an executable file is loaded into memory
BaseOfCode	Offset of the executable code
BaseOfData	Offset of the initialized data
ImageSz	Amount of memory the image file will need
HeaderSz	Length of all headers including the data directories and the section headers
StackReserveSz	Stack commit size
StackCommitSz	Size of initially committed stack
HeapReserveSz	Size of the local heap space reserve
HeapCommitSz	Size of the committed heap
DataDirectories	# of valid entries in the data directories

Table 6.6: List of extracted data directories-based features

Feature name	***Description***
ExportTblSz	Directory of the exported symbols
ImportTblSz	Directory of the imported symbols
ResourceTblSz	Directory of the resources
RelocationTblSz	Directory of the base relocation table
TLSTblSz	Directory of the Thread Local Storage (TLS)
LoadConfigTblSz	Directory of the load configuration
IAT-TblSz	Directory of the Import Address Table (IAT)
DelayImportDescriptors	Address and size of the delay import descriptor

A program's DLL contain valuable information about its functionality [271]. Using the Dependency Scanner option from the PE Explorer tool, we extracted the DLL dependencies for each malicious and non-malicious software application. Software dependencies are modules or pieces of code that are required by an application to load and run correctly. Specifically, the DLL dependencies-based features consists from the information extracted from the program's libraries and functions. The extracted DLL dependencies-based features are given in Table 6.7.

Table 6.7: List of extracted DLL dependencies-based features

Feature name	***Description***
Parameters	# of unique function call parameters
kernel32	# of kernel32.dll
advapi32	# of advapi32.dll
gdi32	# of gdi32.dll
ole32	# of ole32.dll
user32	# of user32.dll
glu32	# of glu32.dll
opengl32	# of opengl32.dll
shell32	# of shell32.dll
comctl32	# of comctl32.dll
comdlg32	# of comdlg32.dll
oleaut32	# of oleaut32.dll
version	# of version.dll
winspool	# of winspool.drv
wininet	# of wininet.dll
wintrust	# of wintrust.dll
mpr	# of mpr.dll
urlmon	# of urlmon.dll
rasapi32	# of rasapi32.dll
msimg32	# of msimg32.dll
imm32	# of imm32.dll
winmm	# of winmm.dll
lz32	# of lz32.dll
indexMax	Maximum # of the function in the ordinal export table
indexMin	Minimum # of the function in the ordinal export table
indexAvg	Average # of the function in the ordinal export table

Some of these DLL dependencies-based features are known to be useful for distinguishing malware among non-malicious software. For example, the work by Kolosnjaji et al. [190] stated that functions imported from kernel32.dll entail that malware opens and manipulates processes. Similarly, functions imported from the shell32.dll suggests that malware launches other programs. In addition, the work by Salehi et al. [251] mentioned the user32.dll, kernel32.dll, advapi32.dll, and wininet.dll as some of the most important DLLs to consider for malware detection.

Chapter 7

Machine Learning Algorithms & Performance Metrics

This chapter explains the conducted experiments to classify between malicious and non-malicious software. Furthermore, it provides a description of the supervised machine learning algorithms and performance metrics used for the experiments in Chapters 8, 9, and 10.

7.1 Background on Standard Machine Learning Algorithms

Through our work we used standard supervised machine learning algorithms for malware and non-malicious software classification. We used ten supervised machine learning algorithms (i.e., J48, Random Forest, Random Tree, OneR, Naive Bayes, JRip, PART, Multilayer Perceptron, SMO, and Decision Table) of different types, with a goal to identify the best performing learner(s). Table 7.1 lists the names and types of the ten learners used in this work.

Table 7.1: Name and type of each learner used for this work

Learner	**Type**
J48 [238]	Tree
Random Forest (RF) [90]	Ensemble Tree
Random Tree [69]	Tree
One R [161]	Rule
Naive Bayes (NB) [173]	Bayes Theorem
JRip [110]	Rule
PART [138]	Rule + Tree
Multilayer Perceptron (MLP) [218]	Artificial Neural Network
SMO [231]	Support Vector Machine
Decision Table [187]	Rule

The J48 learner is an open source Java implementation of the C4.5 decision tree algorithm developed by Ross Quinlan [238]. Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees and outputs the average prediction of the individual trees [90]. Random trees is a collection (ensemble) of tree predictors that is called forest. The classification works as follows: the random trees classifier takes the input feature vector, classifies it with every tree in the forest, and outputs the class label that received the majority of "votes" [174]. One R (1R) is a simple classification algorithm that ranks attributes according to error rate (on the training set) by treating all numerically-valued attributes as continuous and using a straightforward method to divide the range of values into several disjoint intervals [161]. Naive Bayes is an algorithm based on the Bayesian theorem in which numeric estimator precision values are chosen based on analysis of the training data [173]. JRip is a direct rule learner that outputs learned knowledge as rules by using a separate and-conquer technique to identify rules covering instances from a specific class, separate them out, and continue on the remaining instances [110]. PART is a hybrid rule-and-tree algorithm that builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule [138]. MLP is a class of feed forward artificial neural network (ANN) that identify non-linear decision boundaries of data by including many perceptrons

(i.e., nodes) that are organized into multiple layers in the network [218]. SMO is a type of support vector machine (SVM) that implements the sequential minimal optimization algorithm for the training of the support vector classifier [231]. Decision Table builds a decision table majority classifier by evaluating feature subsets using best-first search and can also use cross-validation for evaluation [187]. We used the implementations of these ten learners provided in Weka [150].

Besides the classification algorithms, we also used a feature selection method on all features (i.e., the combined set of power-based, network traffic-based, system logs-based, and code-based static features). Specifically, we used a feature selection method called information gain [180], which ranks the features from the most descriptive to the least descriptive using information gain as a measure. All features by their ranking order can be found in Appendix A.

7.2 Performance Metrics

To evaluate the supervised machine learning algorithms performance, we used several metrics computed from the confusion matrix:

	Actual: *Non-malicious*	**Actual:** *Malware*
Predicted: *Non-malicious*	TN	FN
Predicted: *Malware*	FP	TP

where TN, FN, FP, and TP refer to the numbers of true negatives, false negatives, false positives, and true positives, respectively. We computed the following performance metrics that assess different aspects of the classification:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7.2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{7.3}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN} \tag{7.4}$$

$$\text{F-score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{7.5}$$

$$\text{G-score} = \frac{2 \cdot Recall \cdot (1 - FPR)}{Recall + (1 - FPR)} \tag{7.6}$$

The accuracy (see Equation (7.1)) provides the percentage of instances that were detected correctly. The Recall, defined by Equation (7.2), is the ratio of detected malware to all malware instances. Precision (see Equation (7.3)) determines the fraction of instances correctly classified as malware out of all instances classified as malware. False Positive Rate (FPR), defined by Equation (7.4), is the ratio of non-malicious software applications misclassified as malware to the number of all non-malicious applications. Values of all metrics are in the interval $[0, 1]$. Ideally, a good classifier would have Accuracy, Recall, and Precision of 1 and FPR of 0.

In addition to these metrics, we used two composite metrics: F-score and G-score. The F-score, defined by Equation (7.5), is the harmonic mean of the Recall and Precision. Similarly, G-score, given by Equation (7.6), is the harmonic mean of Recall and $(1 - FPR)$. Larger values of F-score and G-score correspond to better learner performance. An ideal learner would have both F-score and G-score of 1.

Chapter 8

Malware Detection Using Power & Network Traffic Data

In this Chapter we discuss using power consumption and network traffic data for malware detection. The results in this Chapter will be presented at the International Conference on Data Intelligence and Security (ICDIS) in June 2019 [156].

8.1 Approach & Contributions

We used the data collected from our experimental set-up (See Chapter 5) to extract the features. First, both the power consumption and network traffic data of malware and non-malicious software were pre-processed. For the power data, we multiplied the voltage and current for each of the four monitored DC rails to obtain the power consumption in Watts. In the case of the network traffic data, each .pcap file was exported as a .csv file.

Using these features, we conducted a series of machine learning experiments for malware detection, that is, used classification to attribute each run to a malware or non-malicious software. Specifically, we experimented with ten supervised machine learning algorithms (i.e., J48, Random Forest, Random Tree, OneR, Naive Bayes, JRip, PART, Multilayer Perceptron, SMO, and Decision Table) by using their implementations provided in Weka [150]. For each learner, we used ten-fold cross

validation, using nine folds of the labeled malware and non-malicious software instances for training and the tenth fold (of unseen) malware and non-malicious instances for testing. This was repeated ten times, each time using a different fold for testing. The learners performance was evaluated using the performance metrics described in Chapter 7.

Specifically, we explore the following research questions:

RQ1: Do some learners perform consistently better than other using power-based and/or network traffic-based features?

RQ2: Which of the monitored voltage rails is the best predictor for malware detection on a general-purpose computer?

RQ3: Does the combination of power-based features and network traffic-based features provide better malware detection performance than each set of features individually?

RQ4: What is the smallest number of features that can be used for malware detection without performance degradation?

The contributions of the research work presented in this Chapter are as follows:

- Using our testbed [171], we conducted experiments to collect power consumption and network traffic data when running samples of malware and non-malicious software applications. Power data was collected by using a Data Acquisition System (DAQ) which measured the power consumption from four different voltage rails, while the network traffic data was collected using Wireshark [42].

- The study is focused on detecting real malware using the extracted dynamic behavioral features (both power-based and network traffic-based) and supervised machine learning algorithms, which has not been done by any of the previous works.

- We ran a large number of machine learning experiments, which allowed us to identify the best performing learner, DC voltage rails that led to the best malware detection performance, and the subset of features that are the best predictors for malware detection. Even more, the comparison of malware detection performance was done using a comprehensive set of metrics that reflect different aspects of the quality of malware detection.

8.2 Results

8.2.1 RQ1: Learners Analysis Performance

To answer RQ1, we explored which learners perform consistently better than others for power-based and/or commonly used network traffic-based features. We used ten supervised machine learning algorithms (i.e., J48, Random Forest, Random Tree, OneR, Naive Bayes, JRip, PART, Multilayer Perceptron, SMO, and Decision Table). Learners performance were evaluated in terms of the mean F-score and G-score over the ten folds, using only the power-based features (given in Table 6.1) extracted from the +12V CPU rails and for using only commonly used network traffic-based features (given in Table 6.2). Note that for this machine learning experiment we did not used the autocorrelation-based features nor the network flows-based features. Figure 8.1 shows the learners performance for the power-based features, while Figure 8.2 shows the learners performance for the commonly used network traffic-based features. Also, note that in Figure 8.2 the mean values for F-score and G-score of Naive Bayes were very close (i.e., 0.641 and 0.642, respectively), which explains why only the G-score is shown.

Random Forest had the highest F-score for both the power-based features and commonly used network traffic-based features (0.971 and 0.946, respectively). Random Tree, led to the same F-score as Random Forest and had the highest G-score (0.949) when using only power-based features, while J48 had the highest G-score when using only the commonly used network traffic-based features (0.890). Naive

Bayes and SMO performed significantly worse than the other learners, both when using only power-based features and when using only commonly used network traffic-based features. Since Random Forest had the best F-scores and close to the best G-scores for both power-based and commonly used network traffic-based features, we use it as a learner of choice in the rest of Chapter 8.

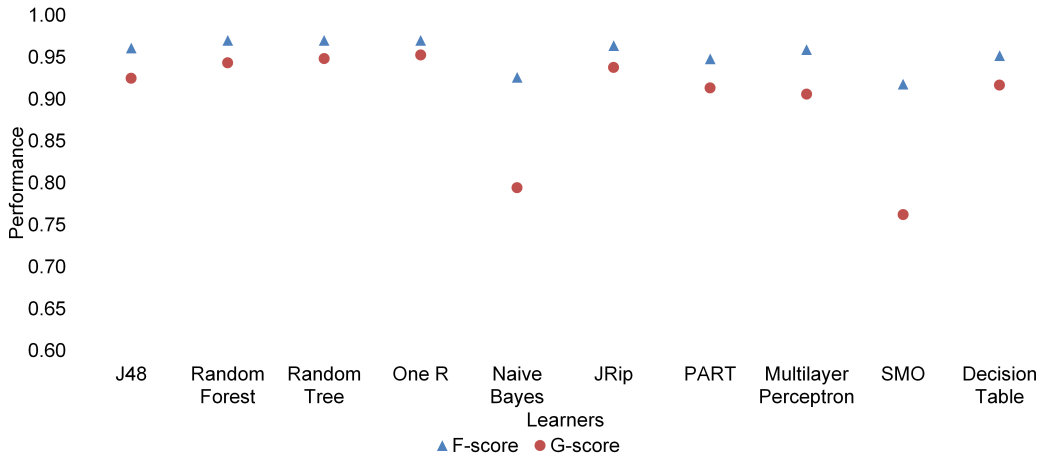

Figure 8.1: Mean F-score & mean G-score for each learner using only power-based features for the +12V CPU rails

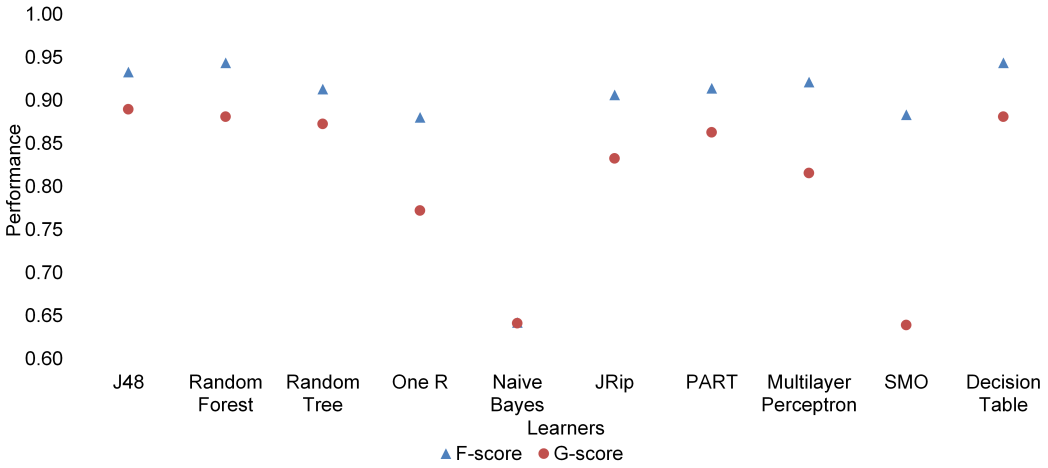

Figure 8.2: Mean F-score & mean G-score for each learner using only commonly used network traffic-based features

8.2.2 RQ2: Voltage Rail Analysis

As described in Chapter 5, four DC voltage channels and four corresponding current channels were monitored. To address RQ2, we evaluated the performance using only the power-based features extracted from power data collected on each of the monitored voltage rails, using the Random Forest classifier. The box plots of the performance metrics, for each of the voltage rails, are shown in Figure 8.3. Note that in Figure 8.3 the performance range is from 0.70 to 1 and instead of FPR we show (1-FPR). Therefore, the results for all performance metrics shown in Figure 8.3 are better when they are closer to 1.

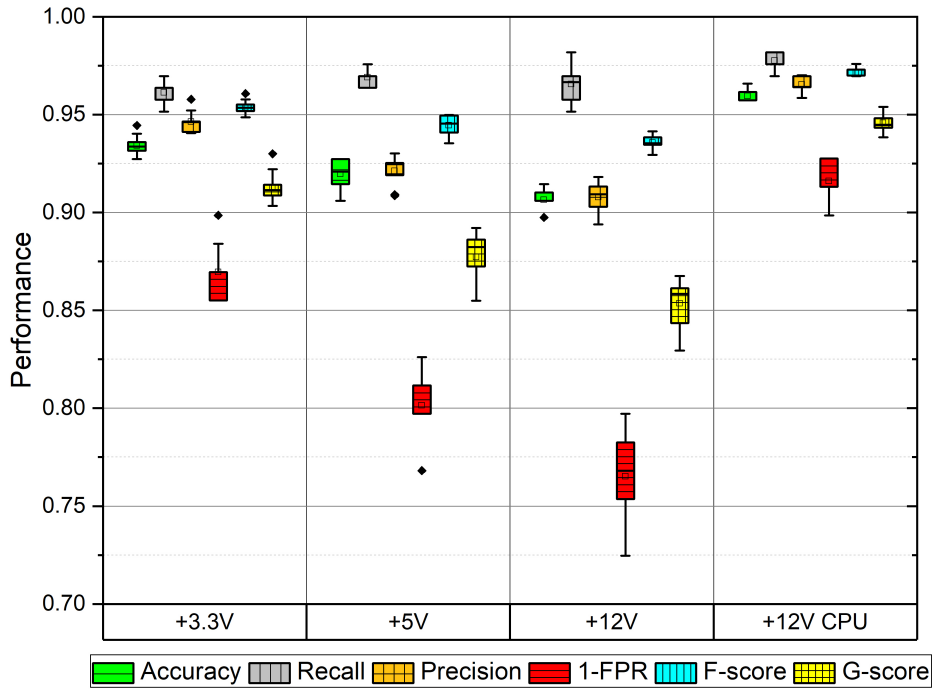

Figure 8.3: Box plots of the Random Forest performance metrics for each of the monitored voltage rails

With respect to all performance metrics, as can be seen in Figure 8.3, the features extracted from the +12V CPU rails led to the best classification results, followed by the +3.3V rails, +5V rails, and +12V rails of the motherboard. The G-scores were 0.945, 0.912, 0.873, and 0.857, respectively. Based on these results

we use the power-based features extracted from the +12V CPU rails in the rest of Chapter 8.

8.2.3 RQ3: Feature Level Fusion Using Power & Network Traffic Data

To answer RQ3, we explored if the combination of power-based features and network traffic-based features (i.e., commonly used network traffic-based features) provide better malware detection performance than each set of features individually. We first used the power-based features and commonly used network traffic-based features separately, then combined into one feature vector. Results presented in Figure 8.4 show that using only power-based features provided significantly better performance than using only commonly used network traffic-based features, with respect to all performance metrics.

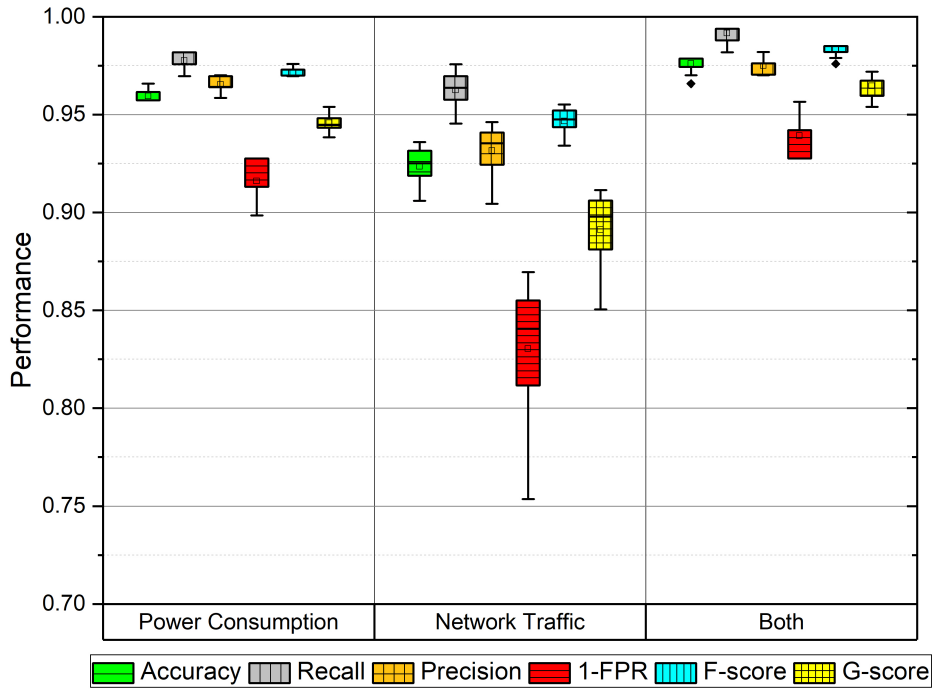

Figure 8.4: Box plots of Random Forest performance using only power-based features from +12V CPU rails, only commonly used network traffic-based features, and combined set of features

Specifically, using only power-based features achieved mean G-score of 0.945 and mean F-score of 0.971, while when using only commonly used network traffic-based features led to mean G-score of 0.891 and mean F-score of 0.946. Note that when using only commonly used network traffic-based features the FPR was considerably higher than when using only power-based features (i.e., 0.169 compared to 0.084).

The performance was the best when the combined set of power-based and commonly used network traffic-based features were used. Adding the commonly used network traffic-based features to the power-based features noticeably improved the malware detection performance (i.e., the mean G-score improved from 0.945 to 0.964 and the mean F-score improved from 0.971 to 0.983). Furthermore, the combined set of features (as in the case of using only power-based features) exhibited much smaller variability of the performance metrics over the 10 fold cross-validation runs than when only commonly used network traffic-based features were used.

We also analyzed the extracted features separately for the two groups (i.e., malware and non-malicious applications). The basic statistics for the power-based features and network traffic-based features are given in Tables 8.1 and 8.2, respectively. We used the non-parametric Mann-Whitney test to explore if the features of the two groups (i.e., malware and non-malicious applications) differ. We used 5% level of significance (i.e., $\alpha = 0.05$). If for a specific feature the p-value is less than α, it follows that there is a statistically significant difference of the values of that feature for malware and non-malicious applications. Statistically significant refers to the likelihood that the differences between the feature values for malware and non-malicious applications are caused by something more than a random chance. Results presented in Table 8.1 show that all power-based features are significantly different for the malware and non-malicious software groups. As shown in Table 8.2, most of the network-based features are also significantly different between malware and non-malicious software groups. The only exceptions are the number of packets related to the LLMNR, UDP, DHCP, and DHCPv6 protocols.

Table 8.1: Basic statistics for power-based features

	Non-malicious					Malware					Mann-Whitney
Power-based features	*Min*	*Max*	*Mean*	*Median*	*SD*	*Min*	*Max*	*Mean*	*Median*	*SD*	*p-value*
PwrMinimum	5.013	42.15	13.22	5.78	12.17	4.95	25.58	5.31	5.11	1.72	$p < 2.2e^{-16}$
PwrMaximum	34.010	57.61	41.02	39.45	4.71	33.13	43.99	38.35	37.96	2.56	$p < 0.001$
PwrMean	6.11	51.13	23.76	24.73	14.25	5.63	28.96	6.69	5.83	3.16	$p < 2.2e^{-16}$
PwrMedian	5.65	54.89	23.58	24.87	15.12	5.53	28.92	6.11	5.66	3.11	$p < 2.2e^{-16}$
PwrVariance	0.07	97.83	23.66	13.01	27.28	0.30	66.56	11.94	1.76	18.67	0.0019
PwrSkewness	−28.41	8.35	0.76	0.61	5.00	1.38	21.90	11.83	12.84	5.94	$p < 2.2e^{-16}$
PwrKurtosis	−1.58	841.33	32.20	7.65	102.67	−0.12	564.60	204.26	195.48	158.50	$p < 2.2e^{-16}$

Table 8.2: Basic statistics for network traffic-based features

	Non-malicious					Malware					Mann-Whitney
Network traffic-based features	*Min*	*Max*	*Mean*	*Median*	*SD*	*Min*	*Max*	*Mean*	*Median*	*SD*	*p-value*
Packets	$5.91e^2$	$2.97e^5$	$2.52e^4$	$3.22e^3$	$5.75e^4$	$1.51e^3$	$2.22e^5$	$7.97e^3$	$3.46e^3$	$2.65e^4$	$p < 0.001$
PktsLength	$1.91e^5$	$2.63e^8$	$2.16e^7$	$4.8e^5$	$5.41e^7$	$3.34e^5$	$1.91e^8$	$3.86e^6$	$5.06e^5$	$2.30e^7$	$p < 0.052$
UniqueSourceIP	6.00	35.00	9.07	6.00	5.91	6.00	53.00	9.75	7.00	8.55	0.0185
UniqueDestIP	14.00	49.00	17.55	14.00	7.10	14.00	571.00	32.39	15.00	65.58	$p < 0.001$
LLMNR	0.00	376.00	55.10	48.00	46.28	28.00	526.00	100.03	48.00	121.88	0.6195
UDP	0.00	142.00	110.49	120.00	32.73	94.00	1921.00	175.86	120.00	294.01	0.1821
ARP	56.00	707.00	538.23	581.00	154.63	131.00	659.00	489.39	535.00	136.37	$p < 0.001$
BROWSER	1.00	15.00	8.26	8.00	2.89	6.00	19.00	10.08	9.00	2.52	$p < 0.001$
NBNS	$4.6e^1$	$1.53e^3$	$1.17e^3$	$1.22e^3$	$3.31e^2$	0.00	$6.40e^3$	$1.40e^3$	$1.42e^3$	$1.16e^3$	$3.54e^{-2}$
DHCP	0.00	54.00	13.51	12.00	8.87	10.00	24.00	12.33	12.00	1.15	0.3299
DHCPV6	0.00	102.00	58.29	56.00	14.28	49.00	70.00	58.92	58.00	3.18	0.45
DNS	19.00	282.00	50.83	30.00	55.05	0.00	$1.08e^4$	189.30	36.00	$1.11e^3$	$p < 0.001$
HTTP	0.00	$1.73e^3$	89.41	0.00	358.12	0.00	$1.73e^4$	350.99	0.00	2246.15	0.04988
ICMP	0.00	21.00	16.58	18.00	5.07	13.00	656.00	41.75	18.00	79.22	$p < 0.001$
ICMPV6	0.00	61.00	36.12	40.00	11.71	30.00	56.00	43.87	41.00	5.25	$p < 0.001$
IGMPV3	0.00	61.00	36.01	39.00	11.70	30.00	56.00	43.86	41.00	5.33	$p < 0.001$
SSDP	12.00	$1.13e^3$	887.36	969.00	260.35	360.00	$1.5e^3$	909.18	913.00	90.32	$p < 0.001$
TCP	0.00	$2.38e^5$	$1.95e^4$	0.00	$4.87e^4$	0.00	$2.01e^5$	$3.81e^3$	53.00	$2.41e^4$	0.0019

- Note that in Table 8.2 some of the network traffic-based features (i.e., Packets, PktsLength, NBNS, HTTP, SSDP, and TCP) were converted to scientific notation.

8.2.4 RQ4: Smallest Feature Set Without Performance Degradation

To answer RQ4, we analyzed the smallest number of features that can be used for malware detection using the information gain feature selection method [180]. To determine the smallest number of features that can be used for classification without performance degradation we started building the model with the highest ranked feature and included one feature at a time until reaching less than or equal to 1% difference of the Recall compared to when all 25 features were used. Table 8.3 shows the features by their ranking order. Note that the features in gray are the power-based features.

The top eleven features ranked by information gain provided similar performance as using all 25 features (i.e., Recall of 0.981). Five out of seven power-based features were among the top eleven features.

Table 8.3: Power-based and commonly used network traffic-based features ranked using information gain

Ranking	***Feature***
1	PwrMedian
2	PwrSkewness
3	PwrAverage
4	PwrMinimum
5	PwrKurtosis
6	Packets
7	SSDP
8	NBNS
9	ARP
10	UDP
11	IGMPV3
12	PwrVariance
13	ICMPV6
14	TCP
15	PwrMaximum
16	PktsLength
17	LLMNR
18	ICMP
19	UniqueSourceIP
20	BROWSER
21	UniqueDestIP
22	DNS
23	DHCP
24	DHCPV6
25	HTTP

8.3 Summary of Findings

The main findings of the work presented in this Chapter are as follows: (1) Random Forest had the highest F-scores and close to the highest G-scores. (2) The power data extracted from the +12V CPU rails led to significantly better performance than power data from the other three voltage rails. (3) Using only power-based features provided better performance than using only network traffic-based features; using combined features led to the best malware detection performance. (4) The top eleven features ranked by information gain provided same performance as using all 25 features. Note that the proposed solution and the used features are only repeatable in this type of environment, since the machine learning algorithms may perform differently depending on the type of malware and used features.

Chapter 9

Malware Detection using Network Traffic & System Logs

This Chapter presents our findings when using network traffic data and system logs to classify malware from non-malicious software. The results presented in this Chapter has been published in the 18^{th} IEEE International Symposium on Network Computing and Applications (NCA) [171].

9.1 Approach & Contributions

In this Chapter, we conducted a series of machine learning experiments for malware detection. The baseline feature vector was created by combining the system logs-based features and the commonly used network traffic-based features. Then, we added the network flows-based features to the baseline feature vector to study their effect on the malware detection. Note that any feature (regardless of the type) which had all instances equal to 0 was removed from the learning process.

For classification, we used four supervised machine learning algorithms (i.e., J48, Random Forest, Naive Bayes, and PART) of different types, with a goal to identify the best performing learner(s). With respect to the malware detection experiments, we used ten-fold cross validation, which consists of using nine folds of the labeled malware and non-malicious software instances for training (i.e., 90%

of the data) and the tenth fold (of unseen) malware and non-malicious instances for testing. We also experimented with smaller training set sizes (i.e., 75%, 50%, and 25% of the data). The learners performance was evaluated using the performance metrics described in Chapter 7.

Specifically, we explore the following research questions:

RQ1: Does the network flows-based features improve the performance of malware detection? What is (are) the best performing learner(s)?

RQ2: What is the smallest number of features sufficient to successfully distinguish malware from non-malicious software? What are the types of the best predictor features?

RQ3: How much data must be set aside for training in order to attain acceptable detection results?

The contributions of the research work presented in this Chapter are as follow:

- Most of the previous works that have used network flows-based features [124, 120, 98, 61, 316, 73, 289, 117, 125, 84, 308, 204, 148, 149] have done classification of the network traffic, while our study is focused on classifying the software running in a machine as malware and non-malicious software using the extracted dynamic behavioral features (i.e., network traffic-based features and system logs-based features).

- Feature selection methods were not commonly used by previous works on malware detection, with an exception of [84]. Determining a small subset of features that can provide predictions as good as when all features are used has a practical usefulness and importance because it allows building more efficient models.

- We experimented with different sizes of the training set (i.e., 90%, 75%, 50%, and 25% of the data) and found that smaller training sets produced very good classification results. Specifically, using 75% of the data for training

has only slightly worse performance compared to when using 90% of the data for training (which is the standard ten-fold cross validation approach). Somewhat worse, but still very good classification performance was achieved with as little as 25% of the data used for training. This aspect of our work has a practical value because the manual labeling of the training set is a tedious and time consuming process.

9.2 Results

9.2.1 RQ1: Network Flows-based Features Performance

To answer RQ1, we compared the learners performance when the baseline features were used (see Figure 9.1) and when the network flows-based features were used (see Figure 9.2). The baseline feature vector was created by combining the system logs-based features (given in Table 6.4) and the commonly used network traffic-based features (given in Table 6.2). Then, we added the network flows-based features (given in Table 6.3) to the baseline feature vector to study their effect on the malware detection. Note that the autocorrelation-based features were not included in this experiment.

For classification, we used four supervised machine learning algorithms (i.e., J48, Random Forest, Naive Bayes, and PART) of different types, with a goal to identify the best performing learner(s). Because low FPR indicates better performance, $1 - FPR$ is shown in Figures 9.1 and 9.2. The range of performance metrics for both Figures 9.1 and 9.2 is from 0.5 to 1. Note that since $1 - FPR$ for Naive Bayes was below 0.5, it is not shown in Figure 9.2.

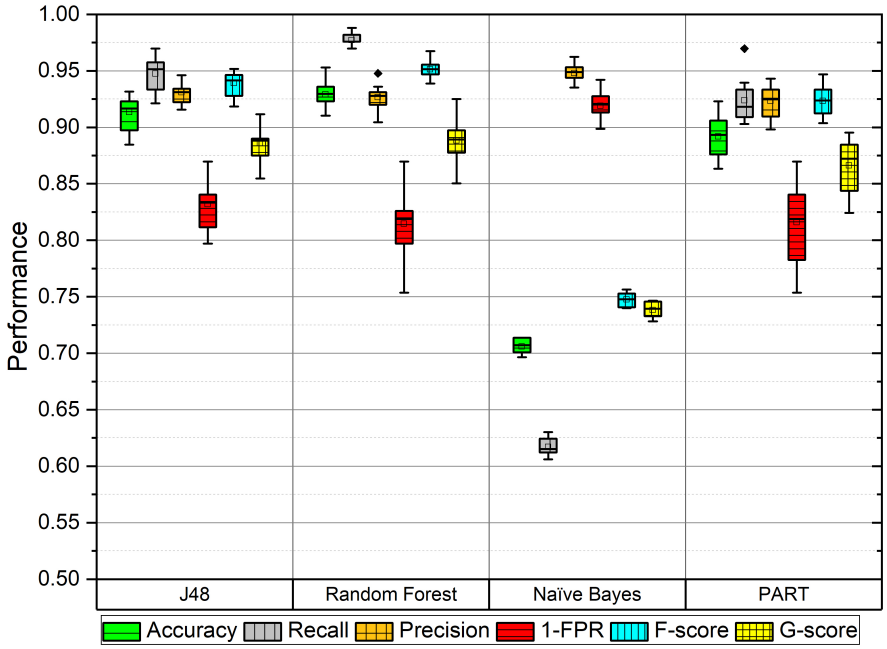

Figure 9.1: Box plots of the learners performance metrics for the baseline feature vector

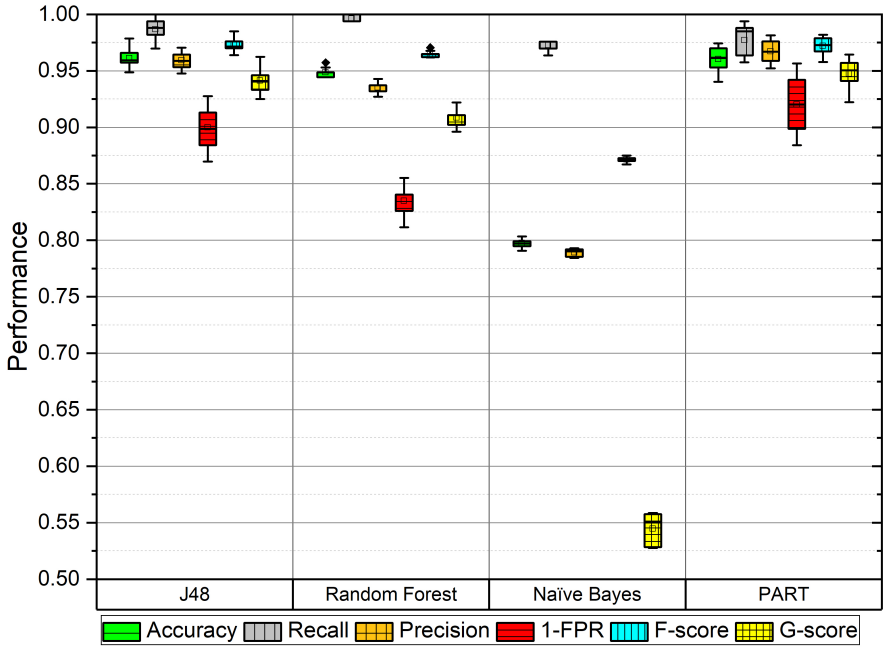

Figure 9.2: Box plots of the learners performance metrics for all features

In addition to box plots shown in Figures 9.1 and 9.2, we used the basic statistics (i.e, mean, median, variance, and interquartile range (IQR)) for the G-score (given in Table 9.1) and F-score (shown in Table 9.2). Note that IQR is a mea-

sure of statistical dispersion, being equal to the difference between 75^{th} and 25^{th} percentiles.

Table 9.1: Basics Statistics of G-score

	Baseline				**All Features**			
Learners	*Mean*	*Median*	*Variance*	*IQR*	*Mean*	*Median*	*Variance*	*IQR*
J48	0.886	0.888	$3.12 \cdot 10^{-4}$	0.014	0.941	0.941	$1.21 \cdot 10^{-4}$	0.013
RF	0.888	0.889	$3.94 \cdot 10^{-4}$	0.019	0.908	0.911	$5.25 \cdot 10^{-5}$	0.008
NB	0.738	0.740	$5.23 \cdot 10^{-5}$	0.019	0.545	0.550	$2.17 \cdot 10^{-4}$	0.029
PART	0.866	0.872	$5.70 \cdot 10^{-4}$	0.037	0.947	0.950	$1.83 \cdot 10^{-4}$	0.016

Table 9.2: Basic Statistics of F-score

	Baseline				**All Features**			
Learners	*Mean*	*Median*	*Variance*	*IQR*	*Mean*	*Median*	*Variance*	*IQR*
J48	0.939	0.942	$1.25 \cdot 10^{-4}$	0.015	0.973	0.972	$3.69 \cdot 10^{-5}$	0.005
RF	0.951	0.951	$6.10 \cdot 10^{-5}$	0.008	0.965	0.965	$7.40 \cdot 10^{-6}$	0.002
NB	0.747	0.748	$4.11 \cdot 10^{-5}$	0.011	0.871	0.871	$6.21 \cdot 10^{-6}$	0.002
PART	0.923	0.924	$1.97 \cdot 10^{-4}$	0.018	0.972	0.973	$5.78 \cdot 10^{-5}$	0.011

In case of G-score, when network flows-based features were added to the baseline feature vector, J48, Random Forest, and PART showed a significant improvement of the mean and median G-score, as well as smaller variance and IQR. On the other side, the Naive Bayes algorithm experienced degradation of the G-score when all features were used. This was due to the increased FPR, which likely was a result of the fact that this learner assumes that features are conditionally independent from one another.

In the case of F-score, the performance of all learners was improved when using all features compared to when the baseline feature vector was used, that is, they had significantly higher mean and median F-scores and smaller variance and IQR. Note that the F-score of the Naive Bayes algorithm had significantly smaller mean and median values than the other three algorithms.

In summary, when all features were used for classification, J48 and PART were the best performing learners. PART had sightly higher median G-score than J48

(0.950 compared to 0.941), while they had similar median F-score values (0.973 and 0.972, respectively). Since J48 and PART were the best performing learners, we used them as learners of choice in the rest of Chapter 9.

9.2.2 RQ2: Smallest Feature Set Without Performance Degradation

To address RQ2, we used feature selection method on all features (i.e., the combined set of baseline features and network flows-based features). Specifically, we used a feature selection method called information gain [180], which ranks the features from the most descriptive to the least descriptive using the information gain as a measure.

To determine the smallest number of features that can be used for malware detection without performance degradation we used the following approach. We started building the model with the highest ranked feature and included one feature at a time until reaching less than or equal to 1% difference of the Recall compared to when all 88 features were used. Table 9.3 shows all features by their ranking order. Note that in Table 9.3 the network flows-based features are shown in gray, while the system logs-based features are shown in bold.

For J48, the top five features ranked by information gain provided similar performance as when using all 88 features. Four out of the five features were network flows-based features. In the case of PART, the first fourteen features ranked by information gain led to similar performance as when using all 88 features. Six out of the fourteen features were network flows-based features.

Table 9.3: Network traffic-based and system logs-based features ranked using information gain

Rank	***Feature***
1	BytesSntMax
2	PktsSentSum
3	PktsSentMax
4	Packets
5	BytesSntSum
6	L4ProtoIGMP
7	SSDP
8	NBNS
9	ARP
10	UDP
11	BytesSntAvg
12	IGMPV3
13	ICMPV6
14	TCP
15	PktsRcvdSum
16	DurationAvg
17	BytesTransferred
18	AvgPktSizeAvg
19	TcpAckFaultCntMax
20	PktsSentAvg
21	LLMNR
22	BytesRcvdSum
23	ICMP
24	UniqueSourceIP
25	TcpAckFaultCntAvg
26	PktsRcvdMax
27	FlowDirA
28	BytesRcvdMax
29	BROWSER
30	TcpInitWinSzAvg
31	L4ProtoTCP
32	Flows
33	AvgPktSizeMedian
34	TcpInitWinSzMax
35	BytpsAvg
36	BytesRcvdAvg
37	BytesSntMin
38	PktsRcvdAvg
39	StdPktSzStd
40	TcpFlLAcRcBytAvg
41	**RegistryChgs**
42	**SetValueKeyChgs**
43	UniqueDestIP
44	TcpPSeqCntMax

Rank	***Feature***
45	TcpWinSzDwnCntMax
46	TcpPAckCntMax
47	FlowDirB
48	TcpFlLAcRcBytMax
49	TcpAveWinSzAvg
50	MinPktSizeMin
51	DNS
52	PktpsMax
53	BytAsmAvg
54	TcpWinSzUpCntMax
55	**FileChgs**
56	**FlsWrite**
57	TcpSeqFaultCntMax
58	PktpsAvg
59	L4ProtoUDP
60	TcpPSeqCntAvg
61	TcpWnSzChgDiCnMax
62	TcpPAckCntAvg
63	**Changes**
64	L4ProtoICMP
65	AvgIATAvg
66	AvgIATMedian
67	**TerminatedPrcs**
68	**ProcessesChgs**
69	**CreatedPrcs**
70	DHCPV6
71	DHCP
72	StdIATStd
73	**FlsDelete**
74	HTTP
75	MaxPktSizeMax
76	TcpWnSzChDiCnAvg
77	TcpSeqSntBytesMax
78	PktAsmAvg
79	PktAsmMin
80	BytpsMax
81	BytAsmMin
82	TcpSeqSntBytesAvg
83	TcpWinSzUpCntAvg
84	TcpSeqFaultCntAvg
85	TcpWnSzDwCnAvg
86	DurationMax
87	TcpAveWinSzMedian
88	**DelValueKeyChgs**

9.2.3 RQ3: Training Sets with Different Sizes

To address RQ3, we explored how much data must be set aside for training in order to attain acceptable detection results. For this part of our study, we restricted the experiments to the best performing learners J48 and PART, using all features. Table 9.4 shows the performance of J48 and PART using training sets with different sizes (i.e., 90%, 75%, 50%, and 25% of the data).

Table 9.4: J48 and PART performance on training sets with different sizes

Learner	***Performance Metrics***	***% of data used for training***			
		90%	*75%*	*50%*	*25%*
J48	*Accuracy*	96.11%	94.74%	92.56%	92.13%
	Precision	95.94%	94.91%	93.82%	91.25%
	Recall	98.67%	97.82%	95.88%	90.15%
	FPR	10.00%	12.61%	15.36%	10.23%
	F-score	97.28%	96.32%	94.80%	90.63%
	G-score	94.13%	92.26%	89.69%	89.43%
PART	*Accuracy*	96.03%	94.15%	92.74%	91.90%
	Precision	96.72%	94.35%	93.98%	91.85%
	Recall	97.70%	97.58%	95.88%	89.55%
	FPR	8.00%	14.06%	14.78%	9.93%
	F-score	97.20%	95.92%	94.90%	90.60%
	G-score	94.75%	91.33%	90.15%	89.20%

The results showed that the learners were able to produce similar performance with 75% of the data used for training as in the case when 90% of data were used for training, which is the commonly used 10-fold cross validation machine learning approach. The performance of the learners was more significantly affected when 50% of the data were used for training, with less than 3% degradation of the F-score and 5% degradation of the G-score compared to when 90% of the data were used for training.

Even when only 25% of the data were used for training the malware detection performance was still satisfactory, with Accuracy, Precision, Recall, F-score, and G-score all around or above 90% and less than 7% degradation of the F-score and

6% degradation of the G-score compared to when 90% of the data were used for training. It should be noted that the FPR was significantly more affected by the smaller sizes of the training set than any other performance metric.

It appears that the amount of data used for training is a trade-off between somewhat better results at an expense of significantly more effort invested in labeling more data. The fact that smaller training sizes led to successful malware detection is an important result of our study, with a significant practical value because the manual labeling of the training set is a tedious and time consuming process. In addition, to the best of our knowledge, none of related works have experimented with different sizes of training sets.

9.3 Summary of Findings

The main findings of the work presented in this Chapter are as follows: (1) Adding network flows-based features improved significantly the performance of malware detection. (2) J48 and PART were the best performing learners, with the highest F-score and G-score values. (3) Using J48, the first five features ranked by information gain attained the same performance as when using the all 88 features, while in the case of PART the first fourteen features ranked by information led to same performance as when the all 88 features were used. None of the system logs-based features were included in these two models. (4) The classification performance when training on 75% of the data was comparable to training on 90% of the data. Using as little as 25% of the data for training led to somewhat worse, but still very good classification performance, with Accuracy, Precision, Recall, F-score, and G-score all around or above 90% and less than 7% degradation of the F-score and and 6% degradation of the G-score compared to when 90% of the data were used for training.

Chapter 10

Malware Detection Using All Modalities

In Chapters 8 and 9 we evaluated the performance of malware detection using power consumption with network traffic data and network traffic data with system logs, respectively. In this Chapter we are using four modalities, that is, all the extracted features. Specifically, in this Chapter we present a multimodal deep learning neural network that integrates different modalities (i.e., power consumption, system logs, network traffic, and code-based static data) at decision level for malware detection. We evaluate the performance of malware detection for each of the modalities, and when using both feature level and decision level fusion.

10.1 Background on Artificial Neural Network

An artificial neural network (ANN) is an information processing system which is inspired by the models of biological neural networks [272]. The basic unit of computation in a neural network is the neuron, often called a node or unit. A neural network consists of an input layer, one or more hidden layers, and an output layer. The input layer receives various forms of information from the outside world that the network will attempt to learn about, recognize, or otherwise process. The goal of the hidden layer is to transform the inputs into something that the output

layer can use, while the output layer signals how the model responds to the learned information.

Artificial neural networks are widely used in many areas because of its capacity of nonlinear mapping, high accuracy for learning, and good robustness [268]. Six commonly used artificial neural networks in machine learning are: feed forward neural network [145], radial basis function (RBF) neural network [224], Kohonen self organizing neural network [188], recurrent neural network (RNN) [153], convolutional neural network (CNN) [153], and modular neural network [76]. From these types of ANN, we implemented a deep feed forward neural network.

A deep feed forward neural network (also called feed forward network or multilayer perceptron) is an artificial neural network in which the connections between the nodes do not form a cycle. In this network the information moves in one direction, forward, starting from the input nodes and moving onward the hidden layers and output layer [139]. The goal of a feed forward neural network is to approximate some function f^*. For example, for a classifier, $y = f^*(x)$ maps an input x to a category y. A feed forward neural network defines a mapping $y = f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation [196].

Feed forward neural networks are called networks because they are typically represented by composing together many distinct functions. For example let us assume we have three functions ($f^{(1)}$, $f^{(2)}$, and $f^{(3)}$) connected in a chain to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. These chain structures are the most commonly used structure for neural networks, in this case $f^{(1)}$ is the first layer of the network, $f^{(2)}$ is the second layer, and so on. The overall length of the chain provide us with an idea of the depth of the model. During the training process, the idea is to lead $f(x)$ to match $f^*(x)$. The training data provides us with approximate examples of $f^*(x)$ evaluated at different training points. Each example x is followed by a label $y \approx f^*(x)$. Since the training data does not show the desired output for these layers, they are called hidden layers. The last layer of the feed forward neural network is the output layer. Using hidden layers require the usage of activation functions. The activation function of a neuron defines the output of that neuron

given an input or set of inputs. This output is then used as input for the next neuron and so on, until a desired solution to the original problem is found [54]. This function maps the resulting values into the desired range, such as between 0 to 1.

The process in which a feed forward neural networks learns is called back-propagation (also known as BackProp). BackProp is a supervised training scheme, which means it learns from labeled training data. The learning process in deep neural networks is achievable through the usage of optimization algorithms. The objective of optimization algorithms is to search for a parameter vector w^*, in which the loss function f takes a minimum value. A loss function is a measure of how good a prediction model does in terms of being able to predict the expected result. It uses two parameters: bias and weights. The bias are constants attached to neurons and added to the weights input before the activation function is applied, while the weights represents the strength of the connection between the neurons. For example, if the weight from neuron a to b has greater magnitude, it means that neuron a has greater influence over neuron b. In other words, weights decide how much influence the input will have on the output.

Optimization is done through the calculation of gradients. A gradient measures how much the output of a function changes after modifying the inputs gradually [55]. After computing the gradients, the optimization algorithm goes back to adjust the weights and biases in the input and hidden layers to reduce the error. This process is repeated until the difference between the desired and expected output is below some threshold value.

10.2 Background on Multimodal Learning

10.2.1 What is Multimodal Learning?

Multimodal learning involves relating information from multiple sources [221]. Each of these sources is known as a *modality* [79]. The objective of multimodal

learning is to build models that can process and relate information from multiple modalities. However, the research field of multimodal learning brings some unique challenges given the heterogeneity of the data.

Baltrušaitis et al. [79] identified and explored five challenges related to multimodal learning: (1) representation; (2) translation; (3) alignment; (4) fusion; and (5) co-learning. Representation refers to how represent and summarize multimodal data in a way that exploits the complementarity and redundancy of multiple modalities. Translation answers how to map data from one modality to another. Alignment is to identify the direct relations between sub-elements from two or more different modalities. Fusion refers to combine information from two or more modalities to perform a prediction. Co-learning explores how knowledge learning from one modality can help a computational model trained on a different modality.

From these multimodal challenges, in this dissertation we explore multimodal fusion for malware detection.

10.2.2 Multimodal Fusion

Multimodal data fusion is the process of integrating information from multiple modalities with the goal of predicting an outcome (e.g., malware versus non-malicious software) through classification or regression [79, 74]. The interest in multimodal fusion arises due to several advantages: (1) Having access to multiple modalities that observe the same event may allow robust predictions and might also allows us to capture complementary information; (2) A multimodal approach can still operate when one of the modalities is missing or has been compromised (e.g., an attacker modifies a modality). Next, we describe two multimodal fusion levels: feature level and decision level fusion.

10.2.3 Levels of Multimodal Fusion

There exists two levels of multimodal fusion: *feature level* and *decision level*. Feature level (also known as early fusion) is the most widely used approach as it fuses

all the extracted features into one feature vector. Feature level fusion is accomplished by simply combining the feature sets from different modalities. Let us suppose that $\mathbf{X} = \{x_1, x_2, x_3, ..., x_n\}$ and $\mathbf{Y} = \{y_1, y_2, y_3, ..., y_n\}$ are feature vectors ($\mathbf{X} \in R^m$ and $\mathbf{Y} \in R^m$) representing the information extracted from two different modalities. The objective is to combine these two feature sets in order to obtain a new feature vector $\mathbf{Z}$ that would be used for classification. An advantage of feature level fusion is that it uses the correlation between multiple features from different modalities at an early stage, which helps to perform tasks better. However, when doing feature level fusion is hard to represent the time synchronization between the multimodal features [296]. To avoid this issue, features should be represented in the same format before performing the fusion.

Decision level fusion (also known as late fusion) fuses multiple modalities in the semantic space [74]. Here, decisions are combined using a decision fusion unit to make a fused decision vector that is analyzed further to obtain a final decision D about the task or hypothesis. Unlike feature level fusion, the decisions usually have the same format representation. Furthermore, decision level fusion offers scalability in terms of the modalities used during the fusion process, which is difficult to achieve in the feature level fusion [75]. Another advantage of decision level fusion, is that it allows us to use the most suitable methods for analyzing each modality. However, a disadvantage of decision level fusion is that as different learners are used to obtain the local decisions, the learning process for them becomes time consuming. Besides feature level and decision level fusion, some prior works have also used a hybrid approach by performing fusion on both feature level and decision level. The idea behind the hybrid fusion approach is to use the advantages of both early and late fusion strategies. Some prior works that used hybrid fusion focused on multimedia analysis [85, 223, 298].

The fusion of multiple modalities provides complementary information and it is recognized by prior works in multimedia analysis [74] and pattern recognition [79] to increase the classification performance. We used the findings from previous multimodal approaches on multimedia analysis and pattern recognition as motivation

to explore the effectiveness of multimodal data fusion for malware detection.

10.2.4 Data Fusion Techniques

Some previous works have categorized distinct multimodal data fusion techniques [79, 127, 74]. Baltrušaitis et al. [79] divided the multimodal fusion techniques into two categories: model-agnostic approaches and model-based approaches. Model agnostic approaches include fusion techniques, such as averaging, voting schemes, weighting-based, or a learned model; while model-based approaches include fusion techniques like kernel-based methods, graphical models, and neural networks.

Durrant-Whyte et al. [127] described some probabilistic methods that are commonly employed for data fusion in robotics. Most of these methods were based on the Bayes rule for combining prior and observation information. Typically, the Bayes rule can be implemented by using the Kalman and extended Kalman filters through sequential Monte Carlo methods or through the use of functional density estimates. However, there are several limitations when using probabilistic techniques: (1) complexity; (2) inconsistency; and (3) precision of models. To address these limitations in multimodal data fusion, they recommended using techniques like interval calculus, fuzzy logic and/or Dempster-Shafer methods.

Atrey et al. [74] presented three categories for multimodal data fusion techniques: rule-based methods, classification-based methods, and estimation-based methods. Rule-based fusion methods includes a variety of statistical rule-based methods like linear weighted fusion and majority voting. Linear weighted fusion is one of the simplest and widely used method, in which the information is combined in a linear fashion. Some previous works used the linear fusion strategy at the feature level (i.e., for video surveillance and traffic monitoring [175, 288]) and decision level (i.e., for speaker recognition and speech event detection [100]) to perform multimedia analysis tasks. In the case of majority voting, the final decision is the one where the majority of the learners reach a similar decision [228]. Some fusion

methods under the classification-based category are the support vector machine, Bayesian inference, Dempster Shafer theory, dynamic Bayesian networks, neural networks, and the maximum entropy model. While the estimation category includes fusion techniques, such as the Kalman filter, extended Kalman filter, and particle filter fusion methods.

From these data fusion techniques, we decided to used a deep neural network because they are a commonly used in the multimodal domain. For instance, multimodal neural networks have been explored for multimedia analysis tasks [142].

10.3 Malware Detection Using Deep Neural Network with Decision Level Fusion

10.3.1 Approach & Contributions

Only a few works on malware detection used multimodal fusion [193, 184]. However, these works monitored mobile devices and none of them compared the performance of feature level with decision level fusion, which is explored in this dissertation. In this Section we present a multimodal deep learning neural network that integrates different modalities (i.e., power consumption, network traffic data, system logs, and code-based static data). We evaluated the performance of each modality individually, when doing feature level fusion, and when doing decision level fusion.

To construct a multimodal representation using neural networks each modality starts with several individual neural layers followed by a hidden layer that projects the modalities into a joint space [71, 217, 225, 297]. The joint multimodal representation is then passed through multiple hidden layers [79].

Our multimodal deep learning method is a feed forward network and was implemented using Keras [58]. Each modality is inputted individually to the initial networks, which are not connected to each other. The last layers of these networks are connected to the merging layer. The merging layer, which is the first layer

of the final network, concatenates the last hidden layers of the initial networks and outputs the classification results. Figure 10.1 shows the deep neural network architecture for decision level fusion.

Each modality on the initial network consists of an input layer and two hidden layers. The number of used neurons (nodes) for the input layer varies per modality, in particular the static data uses 48 neurons, the power consumption uses 132 neurons, the system logs 10 neurons, and the network traffic data 203 neurons, which it is equal to the number of features extracted for each modality. Note that the number of neurons comprising the input layer must be equal to the number of features in the data [57].

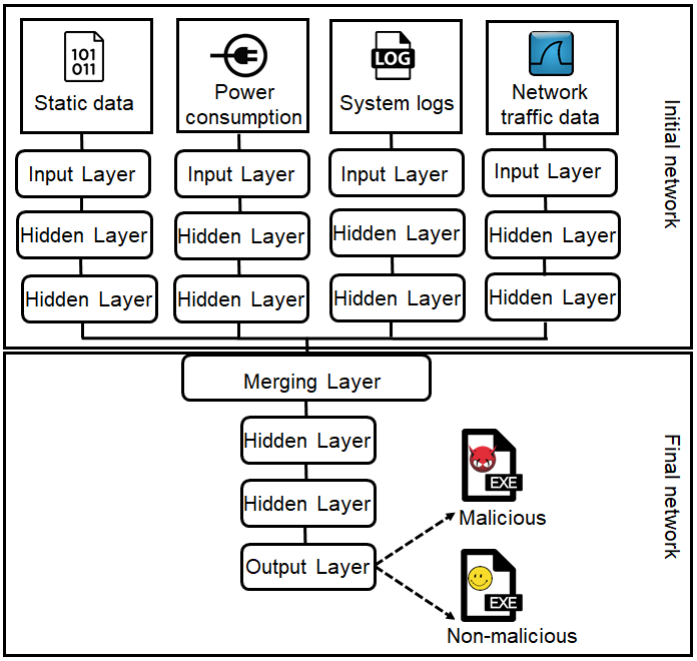

Figure 10.1: Deep neural network architecture for decision level fusion

While ReLU is the most used activation function [196], it has the problem that turns all negative numbers to zeros, which decreases the ability of the model to fit or train the data properly. To avoid this problem we used the exponential linear unit function (ELU) [109] as activation function. Unlike ReLU, ELU allows negative values to push mean unit activations closer to zero speeding up the learning. Furthermore, to avoid overfitting in our multimodal approach we used

dropout regularization [278]. Dropout is a technique that helps to prevent overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently [278]. For our multimodal fusion approach we used a dropout rate of 0.20. This rate is commonly used among deep neural network-based models to prevent overfitting [141].

The structure of the final network is similar to the initial network. It consists of the merging layer (which is the input layer of the final network), two hidden layers, and the output layer which produces the classification results (i.e., classify between malware and non-malicious software). The classification is done using the Sigmoid activation function [151] and the Adam optimization algorithm [185] with 100 epochs, a batch size of 25, and a learning rate of 0.001. The reason why the Sigmoid activation function was chosen is because we are solving a two-class problem and the output for this function ranges between 0 and 1. In the case of the optimization algorithm, Adam was chosen because it is an extension of the stochastic gradient descent, which has been popular among prior works [248]. Furthermore, Adam combines the advantages of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) [185]. We used the default Keras parameters for the Adam optimization algorithm which are: α (learning rate) = 0.001, β_1 =0.9, and β_2 =0.999. The learning rate refers to the proportion in which the weights are updated, β_1 is the exponential decay rate for the first moment estimates, and β_2 is the exponential decay rate for the second moment estimates. In addition, using a batch size of 25 with epochs equal to 100 is a common practice among smaller datasets [56].

To evaluate the performance of our multimodal approach we conducted several experiments. We evaluated the performance of each modality individually, when doing feature level fusion, and when doing decision level fusion. We compared its performance to standard supervised machine learning algorithms (i.e., Random Forest, J48, JRip, PART, Naive Bayes, and SMO). For these six learners we used the implementation provided in Weka [150]. For the malware detection

experiments, we used five-fold cross validation, which consists of using four folds of the labeled malware and non-malicious software instances for training (i.e., 80% of the data) and the fifth fold of unseen malware and non-malicious instances for testing. In the case of our deep learning neural network we also used five fold cross validation, but we used 10% of the data for the validation set, 70% of the data for training, and the remaining 20% of the data for testing. The validation set provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters (e.g., the number of hidden units in a neural network) [243]. We used the validation set as well as the training set to tune the neural network model. While the training set is used to fit the model, the classification accuracy using the validation set is also measured together. It is important to emphasize that the validation set does not update the weights and the biases of the model, but by monitoring the trends of the accuracies of both the training set and the validation set, we can verify whether the model fitting was done correctly by avoiding the overffiting problem. We used the same performance metrics described in Chapter 7.

Specifically, we explore the following research questions:

RQ1: Is each modality a good malware predictor when it is used individually?

RQ2: Does the malware detection performance improve when using multimodal feature level fusion?

RQ3: Does multimodal decision level fusion performs better than using multimodal feature level fusion?

The contributions of the research work presented in this Chapter are as follow:

- Collecting data from different sources allows us to develop a multimodal approach to malware detection, which has not been widely explored by the prior works. Exceptions are [193, 184]. Kumar et al. [193] used two modalities and Kim et al. [184] used only code-based static features, while we are combining behavioral-based with code-based static features. Both of the

prior works [193, 184] monitored mobile devices, while here we monitored a general-purpose computer. None of these works compared the performance of feature level with decision level fusion, which is explored in this dissertation.

- We proposed a multimodal decision level fusion malware detection approach using a deep neural network. We compared its performance with the performance of feature level fusion approaches based on deep neural network and standard supervised machine learning algorithms (i.e., Random Forest, J48, JRip, PART, Naive Bayes, and SMO). Kim et al. [184] used only code-based static features, while we are combining dynamic behavioral-based with code-based static features.

10.4 Results

10.4.1 RQ1: Results Using Each Modality Individually

Each modality was evaluated using our deep learning neural network and six standard supervised machine learning algorithms (i.e., Random Forest, J48, JRip, PART, Naive Bayes, and SMO). For the feature level fusion and for each modality, our deep neural network consisted of the input layer, two hidden layers and the output layer. We used ELU as activation function for the hidden layers, the Sigmoid activation function for the output layer, and the Adam optimization algorithm with 100 epochs, a batch size of 25, and a learning rate of 0.001. Figures 10.2, 10.3, 10.4, and 10.5 show the box plots of the learners performance for each modality individually (i.e., power consumption, network traffic, system logs, and code-based static data, respectively). Because low FPR indicates better performance, $1 - FPR$ is shown in these Figures. The range of performance metrics for all these Figures is from 0.0 to 1.0. Values of all metrics are in the interval $[0, 1]$. Ideally, a good classifier would have Accuracy, Recall, Precision, $1 - FPR$, F-score, and G-score of 1.

In addition to box plots we also used the basic statistics (i.e, mean, me-

dian, variance, and interquartile range (IQR)) based on the F-scores (given in Tables B.1, B.2, B.3, and B.4) for each modality individually. Note that IQR is a measure of statistical dispersion, being equal to the difference between 75^{th} and 25^{th} percentiles. Note that in Tables 10.1, B.1, B.2, B.3, and B.4, highest values are inside a box and lowest values are shown in bold.

Table 10.1: Mean learners performance for each modality individually

Modality	***Learners***	***Performance Metrics***					
		Accuracy	*Recall*	*Precision*	*1-FPR*	*G-score*	*F-score*
Power consumption	RF	0.968	0.994	0.962	0.909	0.950	0.978
	J48	0.953	0.971	0.962	0.912	0.940	0.966
	JRip	0.961	0.982	0.963	0.912	0.946	0.973
	PART	0.950	0.968	0.961	0.909	0.938	0.964
	NB	0.911	**0.942**	0.931	0.838	0.887	**0.936**
	SMO	0.932	1.00	**0.911**	**0.773**	**0.872**	0.953
	Deep NN	0.934	0.970	0.936	0.857	0.910	0.952
Network traffic	RF	0.919	0.990	0.904	0.755	0.856	0.945
	J48	0.944	0.964	0.957	0.898	0.930	0.960
	JRip	0.937	0.971	0.941	0.859	0.911	0.956
	PART	0.935	0.959	0.949	0.879	0.917	0.954
	NB	0.715	0.918	**0.738**	**0.244**	**0.385**	**0.818**
	SMO	0.898	0.966	0.896	0.741	0.839	0.930
	Deep NN	0.791	**0.887**	0.822	0.586	0.701	0.853
System logs	RF	0.756	0.890	0.788	0.445	0.593	0.836
	J48	0.771	0.961	0.769	0.330	0.491	0.854
	JRip	0.747	0.940	0.757	0.300	0.453	0.838
	PART	0.699	1.00	**0.674**	0.189	0.312	0.805
	NB	0.541	**0.429**	0.832	0.798	0.558	**0.566**
	SMO	0.726	1.00	0.718	**0.091**	**0.167**	0.836
	Deep NN	0.805	0.993	0.781	0.400	0.559	0.874
Code-based Static	RF	0.728	1.00	0.720	0.098	0.179	0.837
	J48	0.706	0.996	0.705	**0.033**	**0.064**	0.826
	JRip	0.719	0.948	0.730	0.188	0.311	0.825
	PART	0.711	0.992	0.710	0.061	0.113	0.828
	NB	0.727	0.948	0.737	0.217	0.352	0.829
	SMO	0.738	0.980	0.734	0.177	0.300	0.839
	Deep NN	0.655	**0.937**	**0.679**	0.050	0.093	**0.787**

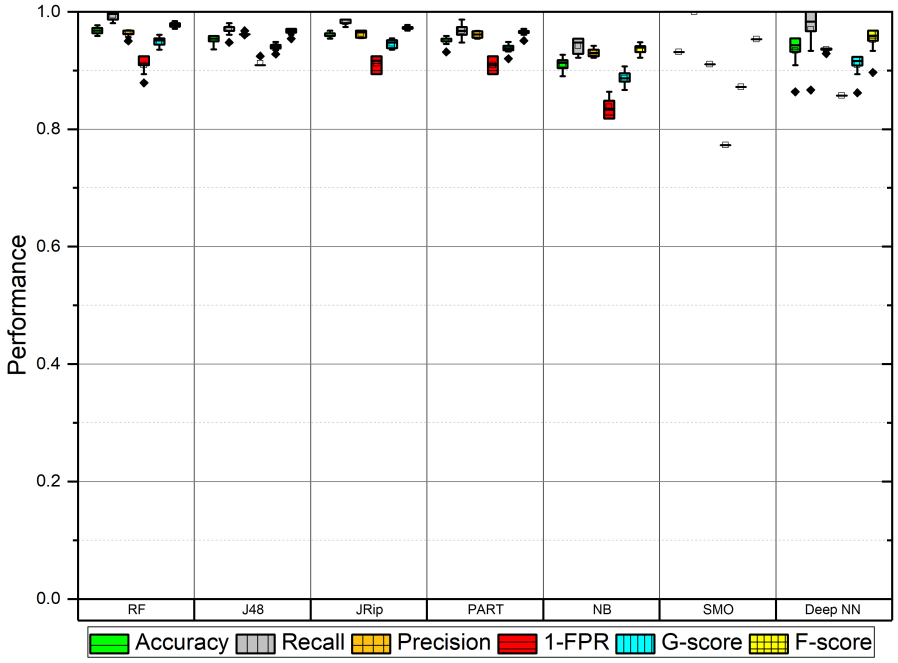

Figure 10.2: Box plots of the learners performance metrics for the power-based features

Results in Figure 10.2 show that Random Forest provides better performance than the other learners with respect to most of the performance metrics, except for $1 - FPR$ (with mean G-score of 0.950 and mean F-score of 0.978), followed by JRip (with mean G-score of 0.946 and mean F-score of 0.973) and the deep learning neural network (with mean G-score of 0.910 and mean F-score of 0.952). Lowest FPR was achieved by J48 and JRip (both with mean FPR of 0.088), followed by Random Forest and PART (both with mean FPR of 0.091). Random Forest and JRip had highest G-score and F-score because both Recall were highest (with mean Recall of 0.994 vs. 0.982), while both learners had similar mean Precision values (0.962 vs. 0.963). Naive Bayes and SMO performed significantly worse than the other learners. G-score for Naive Bayes was affected due to higher FPR (mean FPR of 0.162), while F-score (mean F-score of 0.936) was fairly high because of both Recall (mean Recall of 0.942) and Precision (mean Precision of 0.931). With

respect to SMO, we concluded that our results are unreliable because for each repetition in the 5-folds, the results were the same for all performance metrics.

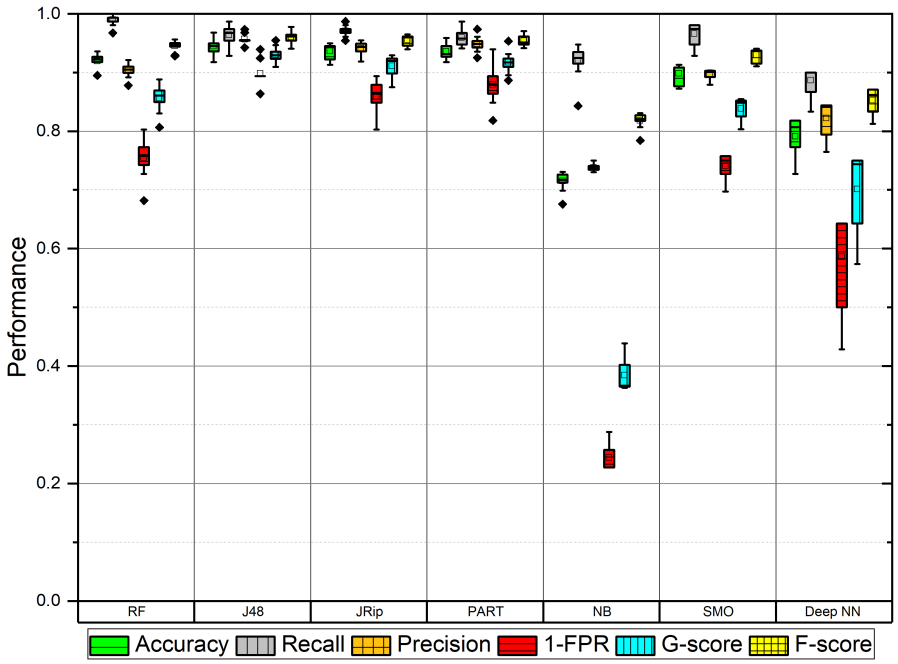

Figure 10.3: Box plots of the learners performance metrics for the network traffic-based features

Results in Figure 10.3 show that lowest FPR was achieved by both J48 and JRip (both with mean FPR of 0.088), followed by Random Forest and PART (both with mean FPR of 0.091). Random Forest and JRip had highest G-score and F-score because both Recall were highest (with mean Recall of 0.994 vs. 0.982), while both learners had similar mean Precision values (0.962 vs. 0.963). Naive Bayes and SMO performed significantly worse than the other learners. G-score for Naive Bayes was affected due to higher FPR (mean FPR of 0.162), while F-score (mean F-score of 0.936) was fairly high because of both Recall (mean Recall of 0.942) and Precision (mean Precision of 0.931).

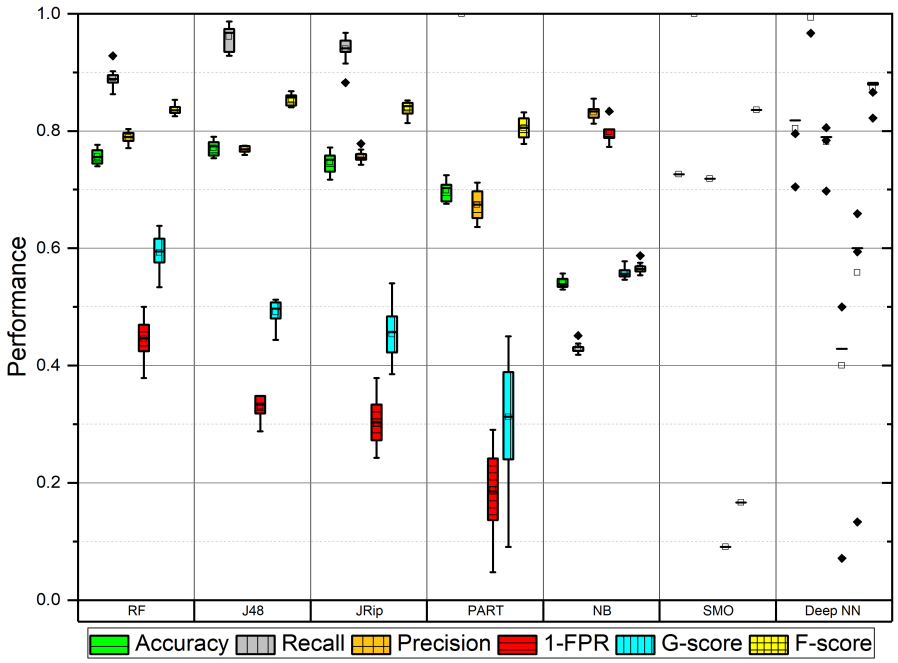

Figure 10.4: Box plots of the learners performance metrics for the system logs-based features

In comparison to power-based and network traffic-based features, system logs-based features did not provide a good performance on their own (see Figure 10.4). However, when comparing the learners performance the deep learning neural network was the best with mean G-score of 0.559 and mean F-score of 0.874. Worse performance was attained for SMO (with mean G-score of 0.167 and mean F-score of 0.836), Naive Bayes (with mean G-score of 0.558 and mean F-score of 0.566), and PART (with mean G-score of 0.312 and mean F-score of 0.805). The performance of SMO was affected by the high FPR (mean FPR of 0.909), while Naive Bayes had lowest Recall (mean Recall of 0.449), and PART was affected due to a lowest Precision (mean Precision of 0.674).

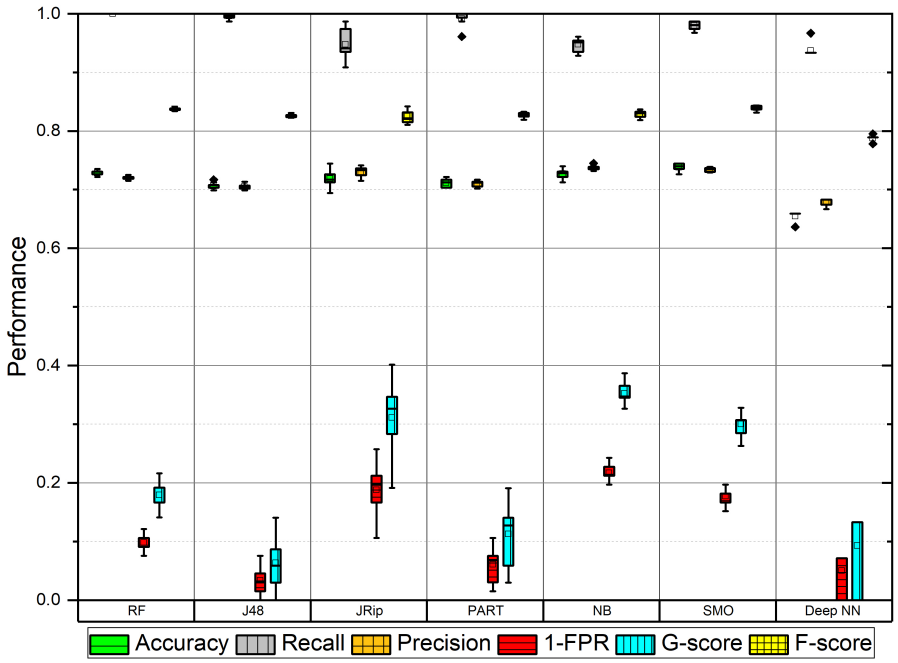

Figure 10.5: Box plots of the learners performance metrics for the code-based static features

As seen in Figure 10.5, the code-based static features did not perform well on their own. While J48 had fairly high F-score, its G-score was lowest among all learners (mean G-score of 0.064). Highest F-score was achieved with Random Forest (mean F-score of 0.837), while the lowest F-score was attained by the deep learning neural network (mean F-score of 0.787). The performance of J48 and the deep learning neural network were affected mostly because of a higher FPR (mean FPR of 0.967 and 0.950) and a lower Precision (mean Precision of 0.705 and 0.679, respectively).

Using different performance metrics is of great importance because it reflects the quality of malware detection. For instance, in Figure 10.5 the Recall is very good (close to 1), and Precision fairly good (higher than 0.70) for all learners, which lead to good F-score. However, the FPR is bad, which led to a bad G-score.

With respect to RQ1 (Is each modality a good malware predictor when it is used individually?), we conclude that power-based features did very good on their own, followed by network traffic-based features. System logs-based and code-based

static features had lowest performance when evaluated individually. Specifically, the statistical means in Table 10.1 shows that Random Forest was the best learner with respect to Recall, F-score and G-score. In the case of the network traffic-based features, J48 (in terms of F-score and G-score) and Random Forest (in terms of Recall) were the best learners. When using system logs-based features and code-based static features, the performance was significantly worse for all learners. Furthermore, the performance of the deep neural network was worse when using power-based, network traffic-based, and code-based static features, that is, it had significantly lowest mean Precision, Recall, F-score, and G-score. Additional basic statistics with respect to F-score for each of the modalities can be found in Appendix B.

10.4.2 RQ2: Results for Multimodal Feature Level Fusion

To evaluate the feature level fusion, we combined all features into one feature vector for all learners. In the case of decision level fusion, all modalities were evaluated individually and their results were fused before making the classification. See Figure 10.6 for the learners performance of both feature level and decision level fusion. The range of performance metrics for Figure 10.6 is from 0.50 to 1.0. In addition to box plots, we also used Table 10.2 to evaluate the mean learners performance for feature level and decision level fusion. Note that in Table 10.2, the results for the standard supervised algorithms are for feature level fusion, Deep NN-FL refers to the deep neural network for feature level fusion, and Deep NN-DL is the deep neural network for decision level fusion.

We used Figure 10.6 and Table 10.2 to answer RQ2 (Does the malware detection performance improve when using multimodal feature level fusion?) and RQ3 (Does multimodal decision level fusion performs better than using multimodal feature level fusion?).

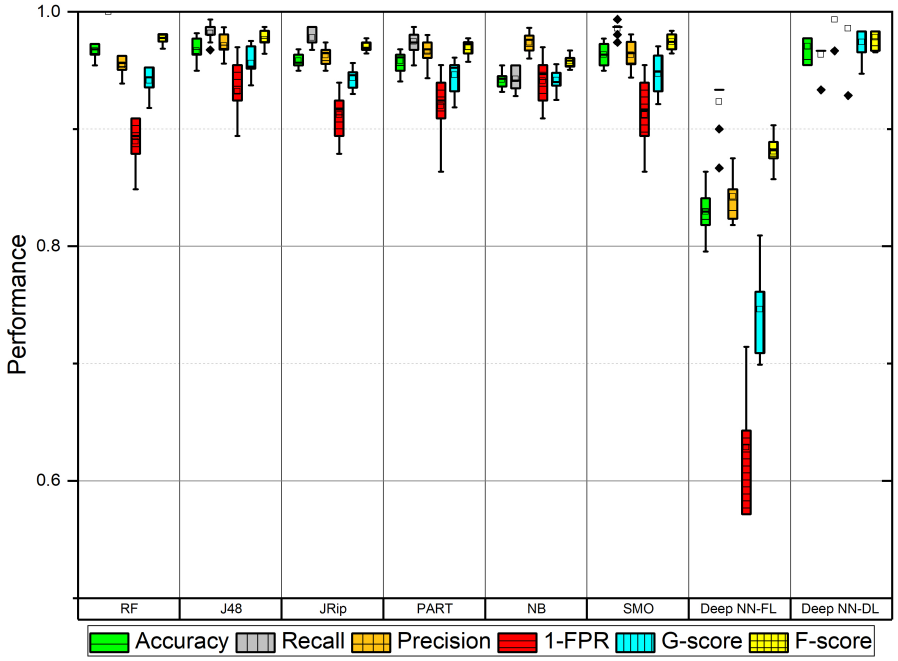

Figure 10.6: Box plots of the learners performance when doing feature level fusion

Table 10.2: Mean learners performance for feature level and decision level fusion

Learners	***Performance Metrics***					
	Accuracy	*Recall*	*Precision*	*1-FPR*	*G-score*	*F-score*
RF-FL	0.967	1.00	0.955	0.889	0.941	0.977
J48-FL	0.967	0.982	0.971	0.932	0.956	0.976
JRip-FL	0.958	0.978	0.963	0.912	0.944	0.970
PART-FL	0.958	0.975	0.966	0.920	0.946	0.970
NB-FL	0.942	0.942	0.974	0.941	0.942	0.958
SMO-FL	0.963	0.986	0.963	0.912	0.947	0.974
Deep NN-FL	0.830	**0.923**	**0.842**	**0.629**	**0.746**	**0.881**
Deep NN-DL	0.970	0.963	0.993	0.986	0.974	0.978

When doing feature level fusion the best performance was obtained for J48 with respect to G-score (mean G-score of 0.956), while the F-score values for both J48 and Random Forest were very close (mean F-score of 0.976 and 0.977, respectively). The second best learners were PART and JRip (with mean G-scores of 0.946 and 0.944 and mean F-scores of 0.970 for both learners). The deep learning neural network had the worst performance for feature level fusion, with respect to all

metrics (mean G-score of 0.746 and mean F-score of 0.881). Its performance was affected by relatively high FPR (with mean FPR of 0.371) and lowest Recall and Precision (mean Recall of 0.923 and mean Precision of 0.842).

To answer RQ2 (Does the malware detection performance improve when using multimodal feature level fusion?), the performance of all learners was compared after we did feature level fusion. Results from Figure 10.6 and Table 10.2 showed that feature level fusion improves the performance of the learners compared to when each modality was evaluated individually, with the exception of the power-based features, which did very good on its own. Note that the malware detection based only on power-based features had the closest performance (See Table 10.1) to the performance of the feature level fusion.

Interestingly, the deep learning neural network had the worse performance for feature level fusion when compared to standard supervised algorithms. This behavior could be explained due to several reasons. First, deep neural networks are proven to work best for bigger datasets [226, 266]. Also, most of the used supervised algorithms are either tree-based or rule-based algorithms, which tend to perform relatively well without considering the dataset size. Second, the deep neural network architecture for the feature level fusion has only the input layer, two hidden layers, and the output layer. Hence, when compared to the architecture for the decision level fusion it has two hidden layers less which might be affecting the ability of the model to learn. Third, the performance of the deep neural network could be affected when dealing with unbalanced data. While we do not consider our data as unbalanced, this possibility is contemplated since we have 153 instances labeled as malware and 66 instances labeled as non-malicious. Meaning we have twice the amount of malware instances when compared to the number of non-malicious instances.

10.4.3 RQ3: Results for Multimodal Decision Level Fusion

Here we are comparing the performance of the deep learning neural network feature level fusion with the deep learning neural network decision level fusion. To address RQ3 (Does multimodal decision level fusion performs better than using multimodal feature level fusion?) we used Figure 10.6 and Tables 10.2.

The performance of the deep learning neural network decision level fusion works best when compared to the deep learning neural network fusion level for all performance metrics. We believe the behavior of the feature level fusion in the deep neural network is due to the size of our dataset. While we are using the same dataset, multimodal decision level fusion offers scalability in terms of the modalities used in the fusion process, which is difficult to achieve in the feature level fusion [74, 75].

When the deep learning neural network decision level fusion was compared with the feature level fusion of standard supervised algorithms, the deep learning neural network decision level fusion works best for most of the performance metrics, with the exception of Recall and F-score (see Tables 10.2). In the case of Recall, both Random Forest and J48 feature level fusion had highest values in comparison to deep learning neural network decision level fusion, which justifies why their mean F-scores were very close to the deep learning neural network decision level fusion. While Recall for the deep learning neural network decision level fusion was not among the highest, its mean F-score was similar to the mean F-scores of both Random Forest and J48 feature level fusion due to higher Precision.

10.5 Summary of Findings

The main findings of the work presented in this Chapter are as follows:

- From the four evaluated modalities, malware detection based on power consumption was the only one that performed well on its own, that is, power-based features are good as malware predictors.

- Some of the learners (e.g., J48) did well for network traffic-based features.
- When using system logs-based or code-based static features, the performance of malware detection was significantly worse for all learners.
- When doing feature level fusion, the performance of Random Forest, J48, JRip, PART, Naive Bayes, and SMO was better when compared to the deep neural network feature level fusion. In terms of F-score, the best performance for feature level fusion was attained by Random Forest.
- When compared to Random Forest, J48, JRip, PART, Naive Bayes, and SMO, the multimodal decision level fusion for the deep neural network works best for most of the performance metrics, with the exception of Recall and F-score.
- The performance of the deep learning neural network decision level fusion works better when compared to the deep learning neural network feature level fusion.

Chapter 11

Threats to Validity

While we took the necessary precautions to prevent threats to validity, these cannot be avoid completely. A threat to the construct validity is the fact that power consumption may vary depending on what type of application is used. For example, there are applications that affect a particular component (e.g., CPU or memory) of the general-purpose computer. To address this issue, we used different types of malware and non-malicious applications.

Another threat to construct of validity with respect to our experimental set-up is that we did not consider evaluating different sampling rates on the hardware configuration when collecting the power consumption data. Nevertheless, the work by [121] stated that a high sampling rate is not always necessary to understand the behavior of the power consumption during the execution of an application software. Furthermore, to prevent construct factors, different hardware configurations were evaluated during the testbed design and development. Evaluating different hardware configurations is of great importance because each configurations may yield different results.

An internal threat to validity is with respect to the number of malware examples that belong to a specific malware type (e.g., trojans, viruses, rootkit, ransomware). Since we used a small representation of each malware type, their behavior may not be representative enough to conclude that all malware types will behave the same way. Furthermore, we also took into consideration the fact that Windows

OS executes actions in the background that could increase the false positive rate when using only power-based features. Thus, to prevent an internal threat to validity we integrated the power-based features with other types of behavioral-based features (network traffic-based and system logs-based features) and code-based static features.

With respect to the conclusion validity we used multiple performance metrics, because doing so helps to reflect different aspects of the quality of malware detection.

A threat to external validity is that while our dataset is largest in comparison to prior works that have used power-based features, it is still small when compared to other works that have done malware detection. For the external validity, it is important to mention that machine learning algorithms may perform differently on different datasets, and when different features are used.

Chapter 12

Conclusions & Future Work

Although malware detection is a very active area of research, few works were focused on using physical properties, such as power consumption. In this dissertation we presented a malware detection approach based on using dynamic features extracted from the power consumption, network traffic data and system logs, which were collected while running malware samples and non-malicious software applications on our experimental testbed. In addition, we also collected code-based static features. We used the extracted features for multimodal malware detection, using both feature level fusion and decision level fusion. Specifically, our malware detection approaches are based on: (1) feature level fusion using power consumption and network traffic data; (2) feature level fusion using network traffic data (i.e., commonly used network traffic-based and network flow-based features) and system logs; and (3) multimodal feature level and decision level fusion.

For the feature level fusion using power consumption and network traffic data, we conducted several machine learning experiments for malware detection. Seven power-based and eighteen network traffic-based features were extracted and ten supervised machine learning algorithms were used for classification. The main findings include: (1) Among the best performing learners, Random Forest had the highest F-score and close to the highest G-score. (2) Power data extracted from the +12V CPU rails led to better performance than power data from the other three voltage rails. (3) Using only power-based features provided better performance

than using only network traffic-based features; using both types of features had the best performance. (4) Feature selection based on information gain was used to identify the smallest numbers of features sufficient to successfully distinguish malware from non-malicious software. The top eleven features provided the same performance as using all 25 features. Five out of seven power-based features were among the top eleven features.

For feature level fusion using network traffic data and system logs, the baseline feature vector was created by combining the system logs-based features and the commonly used network traffic-based features. Then, we added the network flows-based features to the baseline feature vector to study their effect on the malware detection. We evaluated the performance of four supervised machine learning algorithms (i.e., J48, Random Forest, Naive Bayes, and PART) for malware detection and identified the best learner. Furthermore, we used feature selection based on information gain to identify the smallest number of features needed for classification. In addition, we experimented with training sets of different sizes. The main findings include: (1) Adding network flows-based features improved significantly the performance of malware detection. (2) J48 and PART were the best performing learners, with the highest F-score and G-score values. (3) Using J48, the top five features ranked by information gain attained the same performance as when using all 88 features. In the case of PART, the top fourteen features ranked by information gain led to the same performance as when all 88 features were used. None of the system logs-based features were included in these two models. (4) The classification performance when training on 75% of the data was comparable to training on 90% of the data. As little as 25% of the data can be used for training at an expense of somewhat higher, but not very significant performance degradation (i.e., less than 7% for F-score and 6% for G-score compared to when 90% of the data were used for training).

We also presented a multimodal deep learning neural network that integrates different modalities (i.e., power consumption, system logs, network traffic, and code-based static data). We evaluated the performance of each modality indi-

vidually, when doing feature level fusion, and when doing decision level fusion. We compared its performance to standard supervised machine learning algorithms (i.e., Random Forest, J48, JRip, PART, Naive Bayes, and SMO). The main findings for the multimodal approach include: (1) Power-based features did very well on their own for all learners, while some of the learners did well for network traffic-based features (e.g., J48), and system logs-based and code-based static features performed significantly worse for all learners. (2) Deep neural network feature level fusion performs worst compared to feature level fusion for standard supervised algorithms. (3) Deep neural network decision level fusion performs slightly better compared to feature level fusion for standard supervised algorithms.

As part of our future work we would like to increase the sample size for the experiments. Furthermore, instead of hand picking the features given to the multimodal approach, we plan as future work to let the algorithm extract features for both malware and non-malicious software to explore if doing so improves the performance of the model. In addition, we would like to explore the performance of malware detection per malware type and/or malware families and to evaluate other methods for multimodal fusion.

List of Publications

1. **Hernández Jiménez, J.**, Nichols, J. A., Goseva-Popstojanova, K., and Prowell, S. (2016, March). *A malware detection framework based on power consumption monitoring.* In 3rd Women in Cybersecurity Conference (WiCys). [Poster]

2. **Hernández Jiménez, J**., Bridges, R., Nichols, J., Goseva-Popstojanova, K., and Prowell, S. (2016, May). *Towards malware detection framework based on power consumption monitoring.* In 37th IEEE Symposium in Security and Privacy (IEEE SSP). [Poster]

3. **Hernández Jiménez**, J., Chen, Q., Nichols, J., Calhoun, C., and Sykes, S. (2017, January). *Towards a cyber defense framework for SCADA systems based on power consumption monitoring.* In 50th Hawaii International Conference on System Sciences (HICSS), pp. 2915-2921. [**47% acceptance rate**]

4. **Hernández Jiménez, J.**, Nichols, J. A., Goseva-Popstojanova, K., Prowell, S., and Bridges, R. A. (2017, May). *Malware detection on general-purpose computers using power consumption monitoring: A proof of concept and case study.* arXiv preprint arXiv:1705.01977. [Technical Report]

5. Prowell, S., Nichols, J., **Hernández Jiménez, J.** (2018, May). *System and method for monitoring power consumption to detect malware.* Provisional Patent Application No. 62/506,114. [Patent]

6. Bridges, R., **Hernández Jiménez, J.**, Nichols, J., Goseva-Popstojanova, K., and Prowell, S. (2018, August). *Towards malware detection via CPU power consumption: Data collection design and analytics.* In 17th IEEE International Conference On Trust, Security and Privacy in Computing and Communications (IEEE TrustCom), pp. 1680-1684.

7. **Hernández Jiménez, J.**, and Goseva-Popstojanova, K. (2018, November). *The effect on network flows-based features and training set sizes on malware detection.* In 17th IEEE International Symposium on Network Computing and Applications (IEEE NCA), pp. 1-9. [**26% acceptance rate**]

8. **Hernández Jiménez, J.**, and Goseva-Popstojanova, K. (2019, June). *Malware detection using power consumption and network traffic data.* In 2nd IEEE International Conference on Data Intelligence and Security (IEEE ICDIS).

References

[1] (1995) Frequently Asked Questions on virus-L/comp.virus. [Online]. Available: http://www.faqs.org/faqs/computer-virus/faq/

[2] (2001) FlowScan - Network traffic flow visualization and reporting tool. [Online]. Available: https://www.caida.org/tools/utilities/flowscan/

[3] (2003) TCPflow – A TCP flow recorder. [Online]. Available: https://bit.ly/2FWrBgv

[4] (2010) Iran confirms Stuxnet worm halted centrifuges. [Online]. Available: https://cbsn.ws/2U2V5OK

[5] (2010) SplitCap. [Online]. Available: https://bit.ly/2D0RbjH

[6] (2011) Compact DC voltage and current sense PCB with analog output. [Online]. Available: https://bit.ly/2IjBkAy

[7] (2011) Softflowd. [Online]. Available: http://www.mindrot.org/projects/softflowd/

[8] (2012) Malware Domains. [Online]. Available: https://bit.ly/2U2pfBU

[9] (2013) Backdoor.Tidserv. [Online]. Available: https://www.symantec.com/security_response/writeup.jsp?docid=2008-091809-0911-99

[10] (2013) Trojan:Win32/Alureon.FE. [Online]. Available: https://bit.ly/2KdXrLi

[11] (2014) Alert (ta14-150a) gameover Zeus P2P malware. [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA14-150A

[12] (2014) Cyber security event. [Online]. Available: https://definedterm.com/cyber_security_event

[13] (2014) How Dragonfly hackers and RAT malware threaten ICS security. [Online]. Available: https://bit.ly/2uRrjlS

[14] (2015) Clonezilla. [Online]. Available: http://clonezilla.org/

[15] (2015) How to remove TDL4 (Tidserv. [Online]. Available: https://www.solvusoft.com/en/malware/rootkits/tdl4-tidserv/

[16] (2015) Minigrabber test clips. [Online]. Available: https://shop.trenz-electronic.de/en/24645-Mini-Grabber-Test-Clips-6-pack

[17] (2015) Model 3780 minigrabber test clip, one end only. [Online]. Available: http://www.pomonaelectronics.com/pdf/d3780_1_01.pdf

[18] (2015) Power supply unit (PSU). [Online]. Available: https://en.wikipedia.org/wiki/Power_supply_unit_(computer)

[19] (2015) Shunt Resistor. [Online]. Available: http://www.resistorguide.com/shunt-resistor/

[20] (2015) TracerDAQPro. [Online]. Available: https://www.mccdaq.com/daq-software/tracerdaq-series.aspx

[21] (2015) What is a Shunt? [Online]. Available: http://www.reuk.co.uk/wordpress/electric-circuit/what-is-a-shunt/

[22] (2015) Win32/Gamarue. [Online]. Available: https://malwarefixes.com/threats/win32gamarue/

[23] (2016) Dexter. [Online]. Available: https://www.cyber.nj.gov/threat-profiles/pos-malware-variants/dexter

[24] (2016) Get rid of rootkit.Boot.Pihar.B trojan. [Online]. Available: https://bit.ly/2OWiFvz

[25] (2016) How to remove GREENCAT-2. [Online]. Available: https://www.solvusoft.com/en/malware/trojans/greencat-2/

[26] (2016) Payload Security. [Online]. Available: https://www.hybrid-analysis.com/

[27] (2016) Rootkits. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/threat-protection/intelligence/rootkits-malware

[28] (2016) TDL4 Carrier to Glupteba. [Online]. Available: https://bit.ly/2FUnP7m

[29] (2016) Term0l5ter12.com. [Online]. Available: http://www.urlvoid.com/scan/term0l5ter12.com/

[30] (2017) 10 evil user tricks for bypassing anti-virus. [Online]. Available: https://cuckoosandbox.org/

[31] (2017) Backdoor attacks: What is a backdoor? [Online]. Available: https://bit.ly/2TYnBB3

[32] (2017) Capture-BAT. [Online]. Available: https://www.honeynet.org/node/315

[33] (2017) ClockSynchro. [Online]. Available: http://clocksynchro.com/

[34] (2017) Cryptolocker. [Online]. Available: https://en.wikipedia.org/wiki/CryptoLocker

[35] (2017) IDA: About. [Online]. Available: https://www.hex-rays.com/products/ida/index.shtml

[36] (2017) The list of malware types. [Online]. Available: http://www.malwaretruth.com/the-list-of-malware-types/

[37] (2017) Logstash. [Online]. Available: https://www.elastic.co/products/logstash

[38] (2017) OllyDbg. [Online]. Available: http://www.ollydbg.de/download.htm

[39] (2017) Payload (computing). [Online]. Available: https://en.wikipedia.org/wiki/Payload_(computing)

[40] (2017) What is ransomware? [Online]. Available: https://virutec.com/what-is-ransomware/

[41] (2017) What is the difference: viruses, worms, trojans, and bots? [Online]. Available: https://www.cisco.com/c/en/us/about/security-center/virus-differences.html

[42] (2017) Wireshark. [Online]. Available: https://www.wireshark.org/

[43] (2018) Autocorrelation and time series methods. [Online]. Available: https://onlinecourses.science.psu.edu/stat462/node/188/

[44] (2018) Backdoor. [Online]. Available: https://en.wikipedia.org/wiki/Backdoor_(computing)

[45] (2018) Computer worm. [Online]. Available: https://en.wikipedia.org/wiki/Computer_worm

[46] (2018) Firefox. [Online]. Available: https://www.mozilla.org/en-US/firefox/new/

[47] (2018) HeavyLoad. [Online]. Available: https://www.jam-software.com/heavyload/

[48] (2018) IntelBurnTest. [Online]. Available: https://www.majorgeeks.com/files/details/intelburntest.html

[49] (2018) Locky. [Online]. Available: https://nakedsecurity.sophos.com/2016/02/17/locky-ransomware-what-you-need-to-know/

[50] (2018) Malware. [Online]. Available: https://en.wikipedia.org/wiki/Malware

[51] (2018) PE Explorer. [Online]. Available: http://www.heaventools.com/overview.htm

[52] (2018) Tranalyzer. [Online]. Available: https://tranalyzer.com/

[53] (2018) Trojan horse (computing). [Online]. Available: https://en.wikipedia.org/wiki/Trojan_horse_(computing)

[54] (2019) Activation function. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/activation-function

[55] (2019) Gradient descent in a nutshell. [Online]. Available: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

[56] (2019) How big should batch size and number of epochs be when fitting a model in keras? [Online]. Available: https://bit.ly/2VIgkqX

[57] (2019) How to choose the number of hidden layers and nodes in a feedforward neural network? [Online]. Available: https://bit.ly/2jrD2l5

[58] (2019) Keras: The Python deep learning library. [Online]. Available: https://keras.io/

[59] G. K. A. and P. V. K. D., "Survey on ransomware: A new era of cyber attack," *International Journal of Computer Applications*, vol. 168, no. 3, pp. 38–41, 2017.

[60] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "N-gram-based detection of new malicious code," in *28th IEEE Annual International Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2004, pp. 41–42.

[61] D. Acarali, M. Rajarajan, N. Komninos, and I. Herwono, "Event graphs for the observation of botnet traffic," in *8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2017, pp. 628–634.

[62] C. Aguayo González and A. Hinton, "Detecting malicious software execution in Programmable Logic Controllers using power fingerprinting," in *International Conference on Critical Infrastructure Protection*. Springer, 2014, pp. 15–27.

[63] C. Aguayo González and J. H. Reed, "Power fingerprinting in SDR & CR integrity assessment," in *IEEE Military Communications Conference (MILCOM)*, 2009, pp. 1–7.

[64] C. Aguayo González and J. H. Reed, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance," *Analog Integrated Circuits and Signal Processing Journal*, vol. 69, no. 2-3, p. 307, 2011.

[65] M. Ahmadi, A. Sami, H. Rahimi, and B. Yadegari, "Malware detection by behavioural sequential patterns," *Computer Fraud & Security Journal*, vol. 2013, no. 8, pp. 11–19, 2013.

[66] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *6th ACM Conference on Data and Application Security and Privacy*. ACM, 2016, pp. 183–194.

[67] B. A. AlAhmadi and I. Martinovic, "MalClassifier: Malware family classification using network flow sequence behaviour," in *APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2018, pp. 1–13.

[68] S. Alam, I. Traore, and I. Sogukpinar, "Annotated control flow graph for metamorphic malware detection," *The Computer Journal*, vol. 58, no. 10, pp. 2608–2621, 2015.

[69] D. Aldous, "The continuum random tree. I," *The Annals of Probability*, pp. 1–28, 1991.

[70] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: Bridging the static/dynamic gap," in *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, 2012, pp. 3–14.

[71] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh, "VQA: Visual question answering," in *IEEE International Conference on Computer Vision*, 2015, pp. 2425–2433.

[72] F. Apap, A. Honig, S. Hershkop, E. Eskin, and S. Stolfo, "Detecting malicious software by monitoring anomalous windows registry accesses," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2002, pp. 36–53.

[73] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in Android based mobile devices," in *8th IEEE Conference on Next Generation Mobile Applications, Security and Technologies (NGMAST)*, 2014, pp. 66–71.

[74] M. A. Atrey, Pradeep K.and Hossain, A. El Saddik, and M. S. Kankanhalli, "Multimodal fusion for multimedia analysis: A survey," *Multimedia Systems*, vol. 16, no. 6, pp. 345–379, 2010.

[75] P. K. Atrey, M. S. Kankanhalli, and J. B. Oommen, "Goal-oriented optimal subset selection of correlated multimedia streams," *ACM Transactions on*

Multimedia Computing, Communications, and Applications (TOMM), vol. 3, no. 1, p. 2, 2007.

[76] F. Azam, "Biologically inspired modular neural networks," Ph.D. dissertation, Virginia Tech, 2000.

[77] A. Azmoodeh, A. Dehghantanha, M. Conti, and K.-K. R. Choo, "Detecting crypto-ransomware in IoT networks based on energy consumption footprint," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–12, 2017.

[78] J. Bai and J. Wang, "Improving malware detection using multi-view ensemble learning," *Security and Communication Networks*, vol. 9, no. 17, pp. 4227–4241, 2016.

[79] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 2, pp. 423–443, 2019.

[80] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," in *Internet Measurement Workshop*. Citeseer, 2001, pp. 69–73.

[81] M. Bat-Erdene, H. Park, H. Li, H. Lee, and M.-S. Choi, "Entropy analysis to classify unknown packing algorithms for malware detection," *International Journal of Information Security*, vol. 16, no. 3, pp. 227–248, 2017.

[82] U. Bayer, C. Kruegel, and E. Kirda, *TTAnalyze: A tool for analyzing malware*, 2006.

[83] C. Beaumont. (2010) Stuxnet virus: Worm "could be aimed at high-profile Iranian targets". [Online]. Available: https://bit.ly/2WNzw6E

[84] D. Bekerman, B. Shapira, L. Rokach, and A. Bar, "Unknown malware detection using network traffic classification," in *IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 134–142.

[85] A. Bendjebbour, Y. Delignon, L. Fouque, V. Samson, and W. Pieczynski, "Multisensor image segmentation using Dempster-Shafer fusion in Markov fields context," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 8, pp. 1789–1798, 2001.

[86] D. Bilar, "Opcodes as predictor for malware," *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156–168, 2007.

[87] M. Bishop, *Introduction to Computer Security*. Pearson Addison-Wesley Professional, 2004.

[88] G. Bonfante, M. Kaczmarek, and J.-Y. Marion, "Control Flow Graphs as Malware Signatures," in *International Workshop on the Theory of Computer Viruses*, ser. TCV'07, E. Filiol, J.-Y. Marion, and G. Bonfante, Eds. Matthieu Kaczmarek; Guillaume Bonfante, 2007.

[89] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[90] ——, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[91] R. Bridges, J. H. Jiménez, J. Nichols, K. Goseva-Popstojanova, and S. Prowell, "Towards malware detection via CPU power consumption: Data collection design and analytics," in *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom)*, 2018, pp. 1680–1684.

[92] D. Bruschi, L. Martignoni, and M. Monga, "Detecting self-mutating malware using control-flow graph matching," in *Detection of Intrusions and Malware & Vulnerability Assessment*, R. Büschkes and P. Laskov, Eds. Springer Berlin Heidelberg, 2006, pp. 129–143.

[93] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and Intrusion Detection using smart batteries," in *41st IEEE Hawaii International Conference on System Sciences (HICSS)*, 2008, pp. 296–296.

[94] J. Buntinx. (2017) Malware vs. ransomware. [Online]. Available: https://themerkle.com/malware-vs-ransomware/

[95] P. Burnap, R. French, F. Turner, and K. Jones, "Malware classification using self organising feature maps and machine activity data," *Computers & Security Journal*, vol. 73, pp. 399–410, 2018.

[96] S. Burschka and B. Dupasquier, "Tranalyzer: Versatile high performance network traffic analyser," in *IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–8.

[97] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting Android malware using sequences of system calls," in *3rd International Workshop on Software Development Lifecycle for Mobile*, ser. DeMobile 2015. ACM, 2015, pp. 13–20.

[98] Z. B. Celik, J. Raghuram, G. Kesidis, and D. J. Miller, "Salting public traces with attack traffic to test flow classifiers," in *4th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2011, pp. 3–3.

[99] S. Chaba, R. Kumar, R. Pant, and M. Dave, "Malware detection approach for Android systems using system call logs," *arXiv preprint arXiv:1709.08805*, 2017.

[100] L. S.-H. Chen, "Joint processing of audio-visual information for the recognition of emotional expressions in human-computer interaction," Ph.D. dissertation, Champaign, IL, USA, 2000, aAI9971046.

[101] L. Chen, S. Hou, and Y. Ye, "SecureDroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *33rd Annual Computer Security Applications Conference*. ACM, 2017, pp. 362–372.

[102] S. Cherry. (2010) How Stuxnet is rewriting the cyberterrorism playbook. [Online]. Available: https://spectrum.ieee.org/podcast/telecom/security/how-stuxnet-is-rewriting-the-cyberterrorism-playbook

[103] D.-H. Choi and L. Xie, "Malicious ramp-induced temporal data attack in power market with look-ahead dispatch," in *3rd IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2012, pp. 330–335.

[104] B. Claise, "Cisco systems NetFlow services export Version 9," Tech. Rep., 2004.

[105] ——, "Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information," Tech. Rep., 2008.

[106] S. S. Clark, H. Mustafa, B. Ransford, J. Sorber, K. Fu, and W. Xu, "Current events: Identifying webpages by tapping the electrical outlet," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 700–717.

[107] S. S. Clark, B. Ransford, and K. Fu, "Potentia est scientia: Security and privacy implications of energy-proportional computing," in *7th USENIX Conference on Hot Topics in Security (HotSec)*, 2012, pp. 3–3.

[108] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, K. Fu, and W. Xu, "WattsUpDoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *USENIX Conference on Safety, Security, Privacy and Interoperability of Health Information Technologies (HealthTech)*, 2013, pp. 9–9.

[109] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," *arXiv preprint arXiv:1511.07289*, 2015.

[110] W. W. Cohen, "Fast effective rule induction," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 115–123.

[111] L. D. Coronado-De-Alba, A. Rodríguez-Mota, and P. J. Escamilla-Ambrosio, "Feature selection and ensemble of classifiers for Android malware detection," in *8th IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE, 2016, pp. 1–6.

[112] CyberHades. (2017) Malware analyst's DVD. [Online]. Available: https://www.cyberhades.com/2011/03/31/malware-analysts-dvd/

[113] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3422–3426.

[114] J. A. Dawson, J. T. McDonald, J. Shropshire, T. R. Andel, P. Luckett, and L. Hively, "Rootkit detection through phase-space analysis of power voltage measurements," in *12th International Conference on Malicious and Unwanted Software (MALWARE)*, 2017, pp. 19–27.

[115] Z. Dehlawi and N. Abokhodair, "Saudi Arabia's response to cyber conflict: A case study of the Shamoon malware incident," in *IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2013, pp. 73–75.

[116] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! A case study on Android malware detection," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[117] T. Diibendorfer and B. Plattner, "Host behaviour based early detection of worm outbreaks in Internet backbones," in *14th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2005, pp. 166–171.

[118] M. Dillinger, K. Madani, and N. Alonistioti, *Software defined radio: Architectures, systems and functions*. John Wiley & Sons, 2005.

[119] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of Android malware detection based on system calls," in *ACM International Workshop on Security And Privacy Analytics*, 2016, pp. 1–8.

[120] S. Ding, "Machine learning for cybersecurity: Network-based botnet detection using time-limited flows," 2017, unpublished.

[121] M. E. Diouri, M. F. Dolz, O. Glück, L. Lefèvre, P. Alonso, S. Catalán, R. Mayo, and E. S. Quintana-Ortí, "Solving some mysteries in power monitoring of servers: Take care of your wattmeters!" in *European Conference on Energy Efficiency in Large Scale Distributed Systems*, ser. EE-LSDS 2013. Springer-Verlag, 2013, pp. 3–18.

[122] B. Dixon, Y. Jiang, A. Jaiantilal, and S. Mishra, "Location based power analysis to detect malicious code in smartphones," in *1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011, pp. 27–32.

[123] B. Dixon, S. Mishra, and J. Pepin, "Time and location power based malicious code detection techniques for smartphones," in *13th IEEE International Symposium on Network Computing and Applications (NCA)*, 2014, pp. 261–268.

[124] R. F. M. Dollah, M. Faizal, F. Arif, M. Z. Mas'ud, and L. K. Xin, "Machine learning for HTTP botnet detection using classifier algorithms," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 10, no. 1-7, pp. 27–30, 2018.

[125] F. Dressler, W. Jaegers, and R. German, "Flow-based worm detection using correlated honeypot logs," in *ITG-GI Conference in Communication in Distributed Systems*, 2007, pp. 1–6.

[126] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 1285–1298.

[127] H. Durrant-Whyte and T. C. Henderson, "Multisensor data fusion," *Springer handbook of Robotics*, pp. 585–610, 2008.

[128] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, p. 6, 2012.

[129] Y. Elovici, A. Shabtai, R. Moskovitch, G. Tahan, and C. Glezer, "Applying machine learning techniques for detection of malicious code in network traffic," in *Annual Conference on Artificial Intelligence*. Springer, 2007, pp. 44–50.

[130] C. Estan and G. Magin, "Interactive traffic analysis and visualization with Wisconsin Netpy," in *Large Installation System Administration Conference (LISA)*, vol. 5, 2005, pp. 17–17.

[131] Y. Fang, B. Yu, Y. Tang, L. Liu, Z. Lu, Y. Wang, and Q. Yang, "A new malware classification approach based on malware dynamic analysis," in *Australasian Conference on Information Security and Privacy*. Springer, 2017, pp. 173–189.

[132] A. M. Fawaz, M. Noureddine, and W. H. Sanders, "PowerAlert: An integrity checker using power measurement," *arXiv preprint arXiv:1702.02907*, 2017.

[133] A. M. Fawaz and W. H. Sanders, "Learning process behavioral baselines for anomaly detection," in *22nd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2017, pp. 145–154.

[134] X. Feng, R. Ge, and K. W. Cameron, "Power and energy profiling of scientific applications on distributed systems," in *19th International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 2005, pp. 34–44.

[135] J. Fildes. (2010) Stuxnet worm "targeted high-value Iranian assets". [Online]. Available: http://www.bbc.com/news/technology-11388018

[136] FireEye. (2017) Looking ahead: Cybersecurity in 2018. [Online]. Available: https://www.fireeye.com/current-threats/annual-threat-report.html

[137] J. B. Fraley and M. Figueroa, "Polymorphic malware detection using topological feature extraction with data mining," in *SoutheastCon*. IEEE, 2016, pp. 1–7.

[138] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *15th International Conference on Machine Learning*, 1998, pp. 144–151.

[139] J. A. Freeman, *Simulating neural networks with Mathematica*. Addison-Wesley Reading (Mass.) etc, 1994.

[140] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[141] A. Gajbhiye, S. Jaf, N. Al Moubayed, A. S. McGough, and S. Bradley, "An exploration of dropout with rnns for natural language inference," in *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 157–167.

[142] M. Gandetto, L. Marchesooti, S. Sciutto, D. Negroni, and C. S. Regazzoni, "From multi-sensor surveillance towards smart interactive spaces," in *International Conference on Multimedia and Expo. (ICME)*, vol. 1. IEEE, 2003, pp. I–641.

[143] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 5, no. 02, p. 56, 2014.

[144] R. Ge, X. Feng, and S. Shuaiwen, "Powerpack: Energy profiling and analysis of high-performance systems and applications," in *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 2010, pp. 658–671.

[145] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[146] S. Guo, Q. Yuan, F. Lin, F. Wang, and T. Ban, "A malware detection algorithm based on multi-view fusion," in *International Conference on Neural Information Processing*. Springer, 2010, pp. 259–266.

[147] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 194–204, 2013.

[148] F. Haddadi, D. Le Cong, L. Porter, and A. N. Zincir-Heywood, "On the effectiveness of different botnet detection approaches," in *Information Security Practice and Experience*. Springer, 2015, pp. 121–135.

[149] F. Haddadi and A. N. Zincir-Heywood, "Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification," *IEEE Systems Journal*, vol. 10, no. 4, pp. 1390–1401, 2016.

[150] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *ACM (SIGKDD) Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[151] J. Han and C. Moraga, "The influence of the Sigmoid function parameters on the speed of backpropagation learning," in *International Workshop on Artificial Neural Networks*. Springer, 1995, pp. 195–201.

[152] P. Harvey. (2017) ExifTool. [Online]. Available: https://sno.phy.queensu.ca/~phil/exiftool/

[153] S. S. Haykin *et al.*, *Neural networks and learning machines*. New York: Prentice Hall,, 2009.

[154] J. M. Hernández, A. Ferber, S. Prowell, and L. Hively, "Phase-space detection of cyber events," in *10th ACM Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW)*, 2015, p. 13.

[155] J. Hernández Jiménez, Q. Chen, J. Nichols, C. Calhoun, and S. Sykes, "Towards a cyber defense framework for SCADA systems based on power consumption monitoring," in *50th Hawaii International Conference on System Sciences (HICSS)*, 2017, pp. 2915–2921.

[156] J. Hernández Jiménez and K. Goseva-Popstojanova, "Malware detection using power consumption and network traffic data," in *2nd International Conference on Data Intelligence and Security*, 2019.

[157] L. M. Hively and J. T. McDonald, "Theorem-based, data-driven, cyber event detection," in *8th ACM Annual Cyber Security and Information Intelligence Research Workshop*, 2013, pp. 58:1–58:4.

[158] J. Hoffmann, S. Neumann, and T. Holz, "Mobile malware detection based on energy fingerprints—A dead end?" in *Research in Attacks, Intrusions, and Defenses (RAID)*. Springer, 2013, pp. 348–368.

[159] G. Hoglund and J. Butler, *Rootkits Subverting the Windows Kernel*. Addison Wesley, 2008.

[160] M. Holt. (2002) Back to basics-overcurrent protection-incomplete. [Online]. Available: https://www.mikeholt.com/mojonewsarchive/ET-HTML/HTML/Back2BasicsOvercurrentProtection~20020510.htm

[161] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine Learning*, vol. 11, no. 1, pp. 63–90, Apr 1993.

[162] G. Hunt and D. Brubacher, "Detours: Binary interception of Win3 2 functions," in *3rd USENIX Windows NT Symposium*, 1999.

[163] N. A. Huynh, W. K. Ng, and K. Ariyapala, "A new adaptive learning algorithm and its application to online malware detection," in *International Conference on Discovery Science*. Springer, 2017, pp. 18–32.

[164] A. Hylick, R. Sohan, A. C. Rice, and B. Jones, "An analysis of hard drive energy consumption," *IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, pp. 1–10, 2008.

[165] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University*, vol. 48, 2007.

[166] F. Iglesias and T. Zseby, "Analysis of network traffic features for anomaly detection," *Machine Learning*, vol. 101, no. 1, pp. 59–84, Oct 2015.

[167] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, 2013.

[168] G. A. Jacoby, R. Marchany, and N. J. Davis, "Battery-based intrusion detection a first line of defense," in *5th IEEE Annual SMC Information Assurance Workshop*, 2004, pp. 272–279.

[169] R. J. P. S. R. James, A. Albasir, K. Naik, M.-Y. Dabbagh, P. Dash, M. Zaman, and N. Goel, "Detection of unknown applications in smartphones: A signal processing perspective," in *IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017, pp. 1–6.

[170] J.-w. Jang, J. Yun, A. Mohaisen, J. Woo, and H. K. Kim, "Detecting and classifying method based on similarity matching of Android malware behavior with profile," *SpringerPlus*, vol. 5, no. 1, p. 273, 2016.

[171] J. M. H. Jiménez and K. Goseva-Popstojanova, "The effect on network flows-based features and training set size on malware detection," in *17th IEEE International Symposium on Network Computing and Applications (NCA)*, 2018, pp. 1–9.

[172] J. M. H. Jiménez, J. A. Nichols, K. Goseva-Popstojanova, S. Prowell, and R. A. Bridges, "Malware detection on general-purpose computers using power consumption monitoring: A proof of concept and case study," *arXiv preprint arXiv:1705.01977*, 2017.

[173] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *11th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1995, pp. 338–345.

[174] S. R. Kalmegh, "Comparative analysis of weka data mining algorithm randomforest, randomtree and ladtree for classification of indigenous news data," *International Journal of Emerging Technology and Advanced Engineering*, vol. 5, no. 1, pp. 507–517, 2015.

[175] M. S. Kankanhalli, J. Wang, and R. Jain, "Experiential sampling in multimedia systems," *Transactions on Multimedia*, vol. 8, no. 5, pp. 937–946, 2006.

[176] A. Kapoor and S. Dhavale, "Control flow graph based multiclass malware detection using bi-normal separation," *Defence Science Journal*, vol. 66, no. 2, pp. 138–145, 2016.

[177] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," *Journal in Computer Virology*, vol. 1, no. 1-2, pp. 13–23, 2005.

[178] N. Kawaguchi and K. Omote, "Malware function classification using APIs in initial behavior," in *10th Asia Joint Conference on Information Security*. IEEE, 2015, pp. 138–144.

[179] M. Kazmeyer. (2007) Rootkit vs. viruses or worms. [Online]. Available: https://bit.ly/2uVr753

[180] J. T. Kent, "Information gain and a general measure of correlation," *Biometrika*, vol. 70, no. 1, pp. 163–173, 1983.

[181] R. U. Khan, X. Zhang, and R. Kumar, "Analysis of ResNet and GoogleNet models for malware detection," *Journal of Computer Virology and Hacking Techniques*, pp. 1–9, 2018.

[182] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *6th ACM International Conference on Mobile systems, Applications, and Services (MobiSys)*, 2008, pp. 239–252.

[183] S. S. Kim and A. L. N. Reddy, "NetViewer: A network traffic visualization and analysis tool," in *19th Conference on Large Installation System Administration Conference (LISA)*, 2005, pp. 18–18.

[184] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019.

[185] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[186] M. Kjaersgaard. (2014) Everything you need to know about the notorious zeus gameover malware. [Online]. Available: https://bit.ly/2emBaKU

[187] R. Kohavi, "The power of decision tables," in *8th European Conference on Machine Learning*. Springer, 1995, pp. 174–189.

[188] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[189] R. Koller, A. Verma, and A. Neogi, "Wattapp: An application aware power meter for shared data centers," in *7th ACM International Conference on Autonomic Computing (ICAC)*, 2010, pp. 31–40.

[190] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, "Empowering convolutional networks for malware classification and analysis," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 3838–3845.

[191] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2004, pp. 470–478.

[192] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *19th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2013, pp. 1357–1365.

[193] A. Kumar, K. P. Sagar, K. Kuppusamy, and G. Aghila, "Machine learning based malware classification for Android applications using multimodal image representations," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*. IEEE, 2016, pp. 1–6.

[194] J. Kuriakose and P. Vinod, "Ranked linear discriminant analysis features for metamorphic malware detection," in *IEEE International Advance Computing Conference (IACC)*, 2014, pp. 112–117.

[195] D. Kushner. (2014) The real story of Stuxnet. [Online]. Available: https://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet/

[196] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[197] M. Leeds, M. Keffeler, and T. Atkison, "A comparison of features for Android malware detection," in *ACM SouthEast Conference*, 2017, pp. 63–68.

[198] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A survey of network flow applications," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 567–581, 2013.

[199] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *39th Annual Computer Software and Applications Conference*, vol. 2. IEEE, 2015, pp. 422–433.

[200] M. Loukides and A. Oram, "Getting to know GDB," *Linux Journal*, vol. 29, 1996.

[201] P. Luckett, J. T. McDonald, W. B. Glisson, R. Benton, J. Dawson, and B. A. Doyle, "Identifying stealth malware using CPU power consumption and learning algorithms," *Journal of Computer Security*, no. Preprint, pp. 1–25.

[202] F. Maggi, M. Matteucci, and S. Zanero, "Detecting intrusions through system call sequence and argument analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 381–395, 2010.

[203] J. Markoof and D. Sanger. (2010) In a computer worm, a possible biblical clue. [Online]. Available: https://www.nytimes.com/2010/09/30/world/middleeast/30worm.html?_r=3&pagewanted=2&hpw

[204] M. M. Masud, T. Al-Khateeb, L. Khan, B. Thuraisingham, and K. W. Hamlen, "Flow-based identification of botnet traffic by mining multiple log files," in *IEEE International Distributed Framework and Applications Conference*, 2008, pp. 200–206.

[205] M. M. Masud, L. Khan, and B. Thuraisingham, "A scalable multi-level feature extraction technique to detect malicious executables," *Information Systems Frontiers*, vol. 10, no. 1, pp. 33–45, 2008.

[206] McAfee. (2006) Rootkits, part 1 of 3: The growing threat. [Online]. Available: http://web.archive.org/web/20060823090948/http://www.mcafee.com/us/local_content/white_papers/threat_center/wp_akapoor_rootkits1_en.pdf

[207] S. McLaughlin, B. Holbert, S. Zonouz, and R. Berthier, "AMIDS: A multi-sensor energy theft detection framework for advanced metering infrastructures," in *3rd International Conference on Smart Grid Communications (SmartGridComm)*, 2012, pp. 354–359.

[208] Microsoft-TechNet. (2017) Event Viewer. [Online]. Available: https://technet.microsoft.com/en-us/library/cc938674.aspx

[209] R. Mirzazadeh, M. H. Moattar, and M. V. Jahan, "Metamorphic malware detection using linear discriminant analysis and graph similarity," in *5th*

IEEE International Conference on Computer and Knowledge Engineering (ICCKE), 2015, pp. 61–66.

[210] A. Mohaisen, A. G. West, A. Mankin, and O. Alrawi, "Chatter: Classifying malware families using system event ordering," in *Conference on Communications and Network Security*. IEEE, 2014, pp. 283–291.

[211] D. Moon, H. Im, I. Kim, and J. H. Park, "DTB-IDS: An intrusion detection system based on decision tree using behavior analysis for preventing apt attacks," *The Journal of Supercomputing*, vol. 73, no. 7, pp. 2881–2895, 2017.

[212] S. Moore, M. Yampolskiy, J. Gatlin, J. T. McDonald, and T. R. Andel, "Buffer overflow attack's power consumption signatures," in *6th ACM Workshop on Software Security, Protection, and Reverse Engineering*, 2016, p. 6.

[213] S. S. More and P. P. Gaikwad, "Trust-based voting method for efficient malware detection," *Procedia Computer Science*, vol. 79, pp. 657–667, 2016.

[214] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *23rd IEEE Computer security Applications Conference (ACSAC)*, 2007, pp. 421–430.

[215] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, and Y. Elovici, "Unknown malcode detection via text categorization and the imbalance problem," in *IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2008, pp. 156–161.

[216] A. Mpitziopoulos. (2015) PSU 101: A detailed look into power supplies. [Online]. Available: https://www.tomshardware.com/reviews/power-supplies-101,4193.html

[217] Y. Mroueh, E. Marcheret, and V. Goel, "Deep multimodal learning for audio-visual speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 2130–2134.

[218] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[219] P. Nandy, "Hidden Markov model based non-intrusive load monitoring using active and reactive power consumption," Tech. Rep., 2016.

[220] M. Narouei, M. Ahmadi, G. Giacinto, H. Takabi, and A. Sami, "DLLMiner: structural mining for malware detection," *Security and Communication Networks*, vol. 8, no. 18, pp. 3311–3322, 2015.

[221] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *28th International Conference on Machine Learning (ICML)*, 2011, pp. 689–696.

[222] G. Nguyen, B. M. Nguyen, D. Tran, and L. Hluchy, "A heuristics approach to mine behavioural data logs in mobile malware detection system," *Data & Knowledge Engineering*, vol. 115, pp. 129–151, 2018.

[223] J. Ni, X. Ma, L. Xu, and J. Wang, "An image recognition method based on multiple bp neural networks fusion," in *International Conference on Information Acquisition*. IEEE, 2004, pp. 323–326.

[224] M. J. Orr *et al.*, "Introduction to radial basis function networks," 1996.

[225] W. Ouyang, X. Chu, and X. Wang, "Multi-source deep learning for human pose estimation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2329–2336.

[226] A. Pasini, "Artificial neural networks for small dataset analysis," *Journal of Thoracic Disease*, vol. 7, no. 5, p. 953, 2015.

[227] A. Pektaş and T. Acarman, "Effective feature selection for botnet detection based on network flow analysis," 2017.

[228] L. S. Penrose, "The elementary statistics of majority voting," *Journal of the Royal Statistical Society*, vol. 109, no. 1, pp. 53–57, 1946.

[229] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2010, pp. 26–26.

[230] R. S. Pirscoveanu, S. S. Hansen, T. M. Larsen, M. Stevanovic, J. M. Pedersen, and A. Czech, "Analysis of malware behavior: Type classification using machine learning," in *International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*. IEEE, 2015, pp. 1–7.

[231] J. C. Platt, "Advances in kernel methods," B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds., 1999, ch. Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pp. 185–208.

[232] D. Plonka, "FlowScan: A network traffic flow reporting and visualization tool," in *14th USENIX Conference on System Administration (LISA)*, 2000, pp. 305–318.

[233] I. Polakis, M. Diamantaris, T. Petsas, F. Maggi, and S. Ioannidis, "Powerslave: Analyzing the energy consumption of mobile antivirus software," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 2015, pp. 165–184.

[234] R. Polikar, "Ensemble based systems in decision making," *IEEE Circuits and systems magazine*, vol. 6, no. 3, pp. 21–45, 2006.

[235] Ponemon-Institue, "Cost of cyber crime study: Insights on the security investments that make a difference," Tech. Rep., 2017. [Online]. Available: https://accntu.re/2wZ25Rh

[236] P. Prasse, L. Machlica, T. Pevnỳ, J. Havelka, and T. Scheffer, "Malware detection by analysing encrypted network traffic with neural networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 73–88.

[237] S. Qi, M. Xu, and N. Zheng, "A malware variant detection method based on byte randomness test," *Journal of Computers*, vol. 8, no. 10, pp. 2469–2477, 2013.

[238] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[239] R. M. Rad, X. Wang, M. Tehranipoor, and J. Plusquellic, "Power supply signal calibration techniques for improving detection resolution to hardware trojans," in *International Conference on Computer-Aided Design (ICCAD)*. IEEE Press, 2008, pp. 632–639.

[240] R. Rahul, T. Anjali, V. K. Menon, and K. Soman, "Deep learning for network flow analysis and malware classification," in *International Symposium on Security in Computing and Communication*. Springer, 2017, pp. 226–235.

[241] S. Ranveer and S. Hiray, "Comparative analysis of feature extraction methods of malware detection," *International Journal of Computer Applications*, vol. 120, no. 5, 2015.

[242] J. Reed and C. Aguayo González, "Enhancing smart grid cyber security using power fingerprinting: Integrity assessment and intrusion detection," in *Future of Instrumentation International Workshop (FIIW)*. IEEE, 2012, pp. 1–3.

[243] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge university press, 2007.

[244] J. Robbins, "Debugging windows based applications using WinDbg," *Miscrosoft Systems Journal*, 1999.

[245] E. Rodionov and A. Matrosov. The evolution of TDL: Conquering x64. [Online]. Available: http://go.eset.com/resources/white-papers/The_Evolution_of_TDL.pdf

[246] M. Rouse. TDL-4 (TDSS or Alureon) definition. [Online]. Available: https://bit.ly/2YUxofh

[247] ——. (2011) Metamorphic Malware. [Online]. Available: https://searchsecurity.techtarget.com/definition/Metamorphic-virus

[248] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[249] D. Ruta and B. Gabrys, "An overview of classifier fusion methods," *Computing and Information systems*, vol. 7, no. 1, pp. 1–10, 2000.

[250] A. M. Sainju and T. Atkison, "An experimental analysis of Windows log events triggered by malware," in *ACM SouthEast Conference*, 2017, pp. 195–198.

[251] Z. Salehi, M. Ghiasi, and A. Sami, "A miner for malware detection based on API function calls and their arguments," in *16th International Symposium on Artificial Intelligence and Signal Processing (AISP)*. IEEE, 2012, pp. 563–568.

[252] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," in *Proceedings of the ACM Symposium on Applied Computing*. ACM, 2010, pp. 1020–1025.

[253] P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Practical real time intrusion detection using machine learning approaches," *Computer Communications*, vol. 34, pp. 2227–2235, 2011.

[254] SANS. Ouch! What is malware. [Online]. Available: https://bit.ly/2N1EAUG

[255] ——, "Case study: Critical controls that could have prevented target breach," Available: https://www.sans.org/reading-room/whitepapers/casestudies/case-study-critical-controls-prevented-target-breach-35412, Tech. Rep., 2014.

[256] ——, "Case study: The Home Depot data breach," Available: https://bit.ly/1N9y7zk, Tech. Rep., 2015.

[257] ——, "The impact of Dragonfly malware on Industrial Control Systems," Available: https://www.sans.org/reading-room/whitepapers/ICS/impact-dragonfly-malware-industrial-control-systems-36672, Tech. Rep., 2016.

[258] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences*, vol. 231, pp. 64–82, 2013.

[259] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "OPEM: A static-dynamic approach for machine-learning-based malware detection," in *International Joint Conference Special Sessions*. Springer, 2013, pp. 271–280.

[260] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised learning for unknown malware detection," in *International Symposium on Distributed Computing and Artificial Intelligence*. Springer, 2011, pp. 415–422.

[261] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *10th IEEE International Conference on Malicious and Unwanted Software (MALWARE)*, 2015, pp. 11–20.

[262] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *IEEE Symposium on Security and Privacy (S&P)*, 2001, pp. 38–49.

[263] C. Seifert, R. Steenson, I. Welch, P. Komisarczuk, and B. Endicott-Popovsky, "Capture: A behavioral analysis tool for applications and documents," *digital investigation*, vol. 4, pp. 23–30, 2007.

[264] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics Journal*, vol. 1, no. 1, p. 1, 2012.

[265] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "PE-Miner: Mining structural information to detect malicious executables in realtime," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2009, pp. 121–141.

[266] T. Shaikhina and N. A. Khovanova, "Handling limited datasets with neural networks in medical applications: A small-data approach," *Artificial Intelligence in Medicine*, vol. 75, pp. 51–63, 2017.

[267] D. Shamah. (2013) Stuxnet, gone rogue, hit Russian nuke plant, space station. [Online]. Available: http://www.timesofisrael.com/stuxnet-gone-rogue-hit-russian-nuke-plant-space-station/

[268] K. G. Sheela and S. N. Deepa, "Review on methods to fix number of hidden neurons in neural networks," *Mathematical Problems in Engineering*, vol. 2013, 2013.

[269] S. Sheen, R. Anitha, and V. Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach," *Neurocomputing*, vol. 151, pp. 905–912, 2015.

[270] P. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.

[271] M. Sikorski and A. Honig, *Practical malware analysis: The hands-on guide to dissecting malicious software*. No Starch Press, 2012.

[272] S. Sivanandam and S. Deepa, *Introduction to neural networks using Matlab 6.0*. Tata McGraw-Hill Education, 2006.

[273] M. Skrzewski, "Monitoring system's network activity for rootkit malware detection," in *International Conference on Computer Networks*. Springer, 2013, pp. 157–165.

[274] S. Smith and S. Harrison, "Rookits," Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/rootkits.pdf, Tech. Rep., 2012.

[275] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences (HCIS)*, vol. 8, no. 1, p. 3, 2018.

[276] L. Sporck. (2017) 11 of the largest data breaches of all time. [Online]. Available: https://bit.ly/2LyjWWx

[277] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, and J. Hoffmann, "Mobile-Sandbox: Combining static and dynamic analysis with machine-learning techniques," *International Journal of Information Security*, vol. 14, no. 2, pp. 141–153, 2015.

[278] N. Srivastava *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[279] J. Stiborek, T. Pevnỳ, and M. Rehák, "Multiple instance learning for malware classification," *Expert Systems with Applications*, vol. 93, pp. 346–357, 2018.

[280] Techopedia. (2017) Ransomware. [Online]. Available: https://www.techopedia.com/definition/4337/ransomware

[281] TechTerms. (2017) Ransomware Definition. [Online]. Available: https://techterms.com/definition/ransomware

[282] A. Thabet, "Stuxnet malware analysis paper," *Freelancer Malware Researcher Report*, 2011.

[283] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," in *5th IEEE International Conference on Malicious and Unwanted Software (MALWARE)*, 2010, pp. 23–30.

[284] C.-W. Tien, J.-W. Liao, S.-C. Chang, and S.-Y. Kuo, "Memory forensics using virtual machine introspection for malware analysis," in *IEEE Conference on Dependable and Secure Computing*, 2017, pp. 518–519.

[285] D. Veluz. (2010) STUXNET malware targets SCADA systems. [Online]. Available: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/54/stuxnet-malware-targets-scada-systems

[286] C. Wang, J. Pang, R. Zhao, W. Fu, and X. Liu, "Malware detection based on suspicious behavior identification," in *1st IEEE International Workshop on Education Technology and Computer Science*, 2009, pp. 198–202.

[287] C. Wang, J. Ding, T. Guo, and B. Cui, "A malware detection method based on sandbox, binary instrumentation and multidimensional feature extraction," in *International Conference on Broadband and Wireless Computing, Communication and Applications*. Springer, 2017, pp. 427–438.

[288] J. Wang, M. S. Kankanhalli, W. Yan, and R. Jain, "Experiential sampling for video surveillance," in *1st ACM SIGMM International Workshop on Video Surveillance*. ACM, 2003, pp. 77–86.

[289] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting Android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 1096–1109, 2018.

[290] X. Wang, H. Salmani, M. Tehranipoor, and J. Plusquellic, "Hardware trojan detection and isolation using current integration and localized current analysis," in *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFTVS)*, 2008, pp. 87–95.

[291] X. Wang, D. Zhang, X. Su, and W. Li, "Mlifdect: Android malware detection based on parallel machine learning and information fusion," *Security and Communication Networks*, vol. 2017, 2017.

[292] W. Wei. (2014) Zeus botnet updating infected systems with rootkit-equipped trojan. [Online]. Available: https://thehackernews.com/2014/04/zeus-banking-trojan-botnet-rootkit-malware.html

[293] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Security & Privacy*, vol. 5, no. 2, 2007.

[294] E. Wrenn. (2012) Warning from FBI: If you have 'Alureon' virus on your PC, you WILL get kicked off Internet on Monday. [Online]. Available: https://dailym.ai/2WRRZ26

[295] W.-C. Wu and S.-H. Hung, "DroidDolphin: A dynamic Android malware detection framework using big data and machine learning," in *ACM Conference on Research in Adaptive and Convergent Systems*, 2014, pp. 247–252.

[296] Z. Wu, L. Cai, and H. Meng, "Multi-level fusion of audio and visual features for speaker identification," in *International Conference on Biometrics*. Springer, 2006, pp. 493–499.

[297] Z. Wu, Y.-G. Jiang, J. Wang, J. Pu, and X. Xue, "Exploring inter-feature and inter-class relationships with deep neural networks for video classification," in *22nd ACM International Conference on Multimedia*. ACM, 2014, pp. 167–176.

[298] H. Xu and T.-S. Chua, "Fusion of AV features and external information sources for event detection in team sports video," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 2, no. 1, pp. 44–67, 2006.

[299] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 41–61.

[300] P. Yan and Z. Yan, "A survey on dynamic mobile malware detection," *Software Quality Journal*, vol. 26, pp. 891—919, 2018.

[301] Y. Yanfang, "Research on intelligent malware detection methods and their application," Ph.D. dissertation, Department of Computer Science, Xiamen University, 2010.

[302] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 163–182.

[303] H. Yang and R. Tang, "Power consumption based Android malware detection," *Journal of Electrical and Computer Engineering*, vol. 2016, 2016.

[304] Y. Ye, L. Chen, D. Wang, T. Li, Q. Jiang, and M. Zhao, "SBMDS: An interpretable string based malware detection system using SVM ensemble with bagging," *Journal in Computer Virology*, vol. 5, no. 4, p. 283, 2009.

[305] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, p. 41, 2017.

[306] Y. Ye, T. Li, S. Zhu, W. Zhuang, E. Tas, U. Gupta, and M. Abdulhayoglu, "Combining file content and file relations for cloud based malware detection," in *17th ACM International Conference on Knowledge Discovery and Data Mining*. ACM, 2011, pp. 222–230.

[307] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent PE-malware detection system based on association mining," *Journal in Computer Virology*, vol. 4, no. 4, pp. 323–334, 2008.

[308] M. Yeo, Y. Koo, Y. Yoon, T. Hwang, J. Ryu, J. Song, and C. Park, "Flow-based malware detection using convolutional neural network," in *IEEE International Conference on Information Networking (ICOIN)*, 2018, pp. 910–913.

[309] S. Y. Yerima and S. Sezer, "DroidFusion: A novel multilevel classifier fusion approach for Android malware detection," *IEEE Transactions on Cybernetics*, no. 99, pp. 1–14, 2018.

[310] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy Android malware detection using ensemble learning," *IET Information Security*, vol. 9, no. 6, pp. 313–320, 2015.

[311] E. N. Yolacan, J. G. Dy, and D. R. Kaeli, "System call anomaly detection using multi-HMMs," in *8th IEEE International Conference on Software Security and Reliability-Companion*, 2014, pp. 25–30.

[312] S. Yu, S. Zhou, and R. Yang, "Detecting malware variants by byte frequency," *Journal of Networks*, vol. 6, no. 4, pp. 638–645, Apr. 2011.

[313] T. Zefferer, P. Teufl, D. Derler, K. Potzmader, A. Oprisnik, H. Gasparitz, and A. Hoeller, "Power consumption-based application classification and malware detection on Android using machine-learning techniques," *Future Computing*, pp. 26–31, 2013.

[314] Y. Zhang, Q. Huang, X. Ma, Z. Yang, and J. Jiang, "Using multi-features and ensemble learning method for imbalanced malware classification," in *IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom)*. IEEE, 2016, pp. 965–973.

[315] Y. Zhang, C. Rong, Q. Huang, Y. Wu, Z. Yang, and J. Jiang, "Based on multi-features and clustering ensemble method for automatic malware categorization," in *IEEE Trustcom/BigDataSE/ICESS*. IEEE, 2017, pp. 73–82.

[316] D. Zhao, I. Traore, A. Ghorbani, B. Sayed, S. Saad, and W. Lu, "Peer to peer botnet detection based on flow intervals," in *IFIP International Information Security Conference (SEC)*. Springer, 2012, pp. 87–102.

Appendix A

All features ranked using information gain

Table A.1: All features ranked using information gain

Rank	***Feature***
1	PwrMedian
2	PwrMean
3	PwrMinimum
4	PwrSkewness
5	BytesSntSum
6	PwrKurtosis
7	PktsLength
8	L4ProtoIGMP
9	PktsSentSum
10	BytesSntMax
11	PktSentMax
12	SSDP
13	AC_Pwr_50_32
14	ARP
15	AC_Pwr_50_30

Rank	***Feature***
16	AC_Pwr_5_48
17	NBNS
18	AC_Pwr_50_47
19	AC_Pwr_50_49
20	AC_Pwr_50_50
21	UDP
22	BytesSntAvg
23	PwrVariance
24	AC_Pwr_50_40
25	DurationAvg
26	AC_Pwr_20_1
27	AC_Pwr_50_1
28	AC_Pwr_15_1
29	AC_Pwr_25_1
30	AC_Pwr_10_1

Rank	***Feature***
31	AC_Pwr_5_1
32	IGMPV3
33	ICMPV6
34	PktsRcvdSum
35	TCP
36	AvgPktSzAvg
37	AC_Pwr_50_43
38	PwrMaximum
39	BytesRcvdSum
40	PktSentAvg
41	PktAsmAvg
42	No_bytes_transferred
43	LLMNR
44	No_of_ICMP
45	UniqueSourceIP
46	FlowDirA
47	TcpAckFaultCntMax
48	AC_Pwr_5_2
49	AC_Pwr_20_2
50	AC_Pwr_50_2
51	AC _Pwr_15_2
52	AC_Pwr_25_2
53	AC_Pwr_10_2
54	AC_Pwr_50_44
55	AC_Pwr_25_4
56	AC_Pwr_50_4
57	AC_Pwr_10_4
58	AC_Pwr_15_4
59	AC_Pwr_5_4
60	AC_Pwr_20_4

Rank	***Feature***
61	Flows
62	BROWSER
63	TcpInitWinSzAvg
64	AC_Pwr_50_34
65	AC_Pwr_50_28
66	AC_Pwr_25_19
67	AC_Pwr_50_19
68	AC_Pwr_20_19
69	TcpAckFaultCntAvg
70	AC_Pwr_25_14
71	AC_Pwr_20_14
72	AC_Pwr_50_14
73	AC_Pwr_15_14
74	TcpInitWinSzMax
75	AC_Pwr_5_3
76	AC_Pwr_20_3
77	AC_Pwr_25_3
78	AC_Pwr_15_3
79	AC_Pwr_50_3
80	AC_Pwr_10_3
81	AC_Pwr_50_46
82	BytesSntMin
83	AC_Pwr_50_35
84	AC_Pwr_50_33
85	PktsRcvdMax
86	AC_Pwr_50_31
87	AC_Pwr_50_36
88	AC_Pwr_10_6
89	AC_Pwr_50_6
90	AC_Pwr_20_6

Rank	***Feature***
91	AC_Pwr_15_6
92	AC_Pwr_25_6
93	BytAsmAvg
94	AC_Pwr_15_7
95	AC_Pwr_50_7
96	AC_Pwr_10_7
97	AC_Pwr_25_7
98	AC_Pwr_20_7
99	BytpsAvg
100	AC_Pwr_25_12
101	AC_Pwr_20_12
102	AC_Pwr_50_39
103	AC_Pwr_15_12
104	AC_Pwr_50_12
105	AvgPktSzMedian
106	FlowDirB
107	AC_Pwr_50_9
108	AC_Pwr_25_9
109	AC_Pwr_10_9
110	AC_Pwr_20_9
111	BytesRcvdMax
112	AC_Pwr_15_9
113	AC_Pwr_50_41
114	UniqueDestIP
115	BytesRcvdAvg
116	AC_Pwr_50_10
117	AC_Pwr_15_10
118	AC_Pwr_10_10
119	AC_Pwr_25_10
120	AC_Pwr_20_10

Rank	***Feature***
121	RegistryChgs
122	SetValueKeyChgs
123	StdPktSzStd
124	AC_Pwr_50_18
125	AC_Pwr_20_18
126	AC_Pwr_25_18
127	AC_Pwr_50_29
128	AC_Pwr_50_42
129	TcpAveWinSzAvg
130	L4ProtoTCP
131	MinPktSizeMin
132	PktsRcvdAvg
133	AC_Pwr_50_38
134	AC_Pwr_20_13
135	AC_Pwr_50_13
136	AC_Pwr_25_13
137	AC_Pwr_15_13
138	DNS
139	AC_Pwr_50_5
140	AC_Pwr_20_5
141	AC_Pwr_5_5
142	AC_Pwr_25_5
143	AC_Pwr_15_5
144	AC_Pwr_10_5
145	AC_Pwr_10_8
146	AC_Pwr_50_8
147	AC_Pwr_50_27
148	AC_Pwr_50_45
149	AC_Pwr_15_8
150	AC_Pwr_25_8

Rank	***Feature***
151	AC_Pwr_20_8
152	TcpSeqFaultCntMax
153	AC_Pwr_15_15
154	AC_Pwr_25_15
155	AC_Pwr_50_26
156	AC_Pwr_20_15
157	AC_Pwr_50_15
158	L4ProtoUDP
159	PktpsMax
160	AC_Pwr_50_25
161	AC_Pwr_50_20
162	TcpWinSzDwCnMax
163	AC_Pwr_25_20
164	AC_Ntwk_25_11
165	TcpPAckCntMax
166	AC_Pwr_20_20
167	TcpPSeqCntMax
168	AC_Pwr_25_25
169	AC_Ntwk_20_11
170	AC_Ntwk_50_11
171	AC_Ntwk_15_11
172	AC_Pwr_50_11
173	AC_Pwr_20_17
174	AC_Pwr_25_11
175	AC_Ntwk_25_17
176	AC_Pwr_20_11
177	AC_Ntwk_50_17
178	AC_Ntwk_20_17
179	AC_Pwr_25_17
180	AC_Pwr_15_11

Rank	***Feature***
181	AC_Pwr_50_17
182	PktpsAvg
183	AC_Pwr_50_21
184	AC_Pwr_25_21
185	Changes
186	FlsWrite
187	AC_Pwr_50_37
188	FileChgs
189	L4ProtoICMP
190	TcpWiSzUpCntMax
191	ResourceTblSz
192	StackCommitSz
193	ImportTblSz
194	indexMin
195	Sections
196	TLSTblSz
197	oleaut32
198	comctl32
199	wininet
200	DelayImportDescriptors
201	urlmon
202	comdlg32
203	mpr
204	StackReserveSz
205	version
206	wintrust
207	winspool
208	IAT-TblSz
209	rasapi32
210	HeaderSz

Rank	***Feature***
211	DHCP
212	DHCPV6
213	indexMax
214	lz32
215	winmm
216	BaseOfCode
217	ImageSz
218	BaseOfData
219	imm32
220	msimg32
221	opengl32
222	HeapReserveSz
223	AC_Pwr_25_23
224	kernel32
225	LoadConfigTblSz
226	advapi32
227	EntryPointAddr
228	Parameters
229	AC_Pwr_50_16
230	AC_Pwr_50_24
231	AC_Pwr_50_23
232	AC_Pwr_50_22
233	AC_Pwr_25_24
234	RelocationTblSz
235	shell32
236	AC_Pwr_25_22
237	AC_Pwr_20_16
238	HeapCommitSz
239	glu32
240	DataDirectories

Rank	***Feature***
241	ExportTblSz
242	user32
243	AC_Pwr_25_16
244	gdi32
245	ole32
246	HTTP
247	AC_Ntwk_50_45
248	InitializedDataSz
249	AC_Ntwk_25_18
250	AC_Ntwk_25_20
251	AC_Ntwk_25_19
252	AC_Ntwk_25_16
253	AC_Ntwk_50_6
254	AC_Ntwk_25_15
255	AC_Ntwk_25_14
256	AC_Ntwk_25_21
257	AC_Ntwk_25_22
258	AC_Ntwk_25_23
259	AC_Ntwk_25_24
260	AC_Ntwk_50_4
261	AC_Ntwk_50_3
262	AC_Ntwk_50_2
263	AC_Ntwk_50_1
264	AC_Ntwk_25_25
265	AC_Ntwk_25_13
266	AC_Ntwk_25_12
267	AC_Ntwk_25_10
268	AC_Ntwk_25_1
299	AC_Ntwk_20_19
270	AC_Ntwk_20_18

Rank	***Feature***
271	AC_Ntwk_20_16
272	AC_Ntwk_20_15
273	AC_Ntwk_20_14
274	AC_Ntwk_20_20
275	AC_Ntwk_25_2
276	AC_Ntwk_25_9
277	AC_Ntwk_25_3
278	AC_Ntwk_25_8
279	AC_Ntwk_25_7
280	AC_Ntwk_25_6
281	AC_Ntwk_25_5
282	AC_Ntwk_25_4
283	AC_Ntwk_50_5
284	AC_Ntwk_50_7
285	CodeSz
286	AC_Ntwk_50_32
287	AC_Ntwk_50_34
288	AC_Ntwk_50_33
289	AC_Ntwk_50_31
290	AC_Ntwk_50_8
291	AC_Ntwk_50_30
292	AC_Ntwk_50_29
293	AC_Ntwk_50_35
294	AC_Ntwk_50_36
295	AC_Ntwk_50_37
296	AC_Ntwk_50_38
297	AC_Ntwk_50_43
298	AC_Ntwk_50_42
299	AC_Ntwk_50_41
300	AC_Ntwk_50_40

Rank	***Feature***
301	AC_Ntwk_50_39
302	AC_Ntwk_50_28
303	AC_Ntwk_50_27
304	AC_Ntwk_50_26
305	AC_Ntwk_50_16
306	AC_Ntwk_50_14
307	AC_Ntwk_50_13
308	AC_Ntwk_50_12
309	AC_Ntwk_50_10
310	AC_Ntwk_50_9
311	AC_Ntwk_50_15
312	AC_Ntwk_50_18
313	AC_Ntwk_50_25
314	AC_Ntwk_50_19
315	AC_Ntwk_50_24
316	AC_Ntwk_50_23
317	AC_Ntwk_50_22
318	AC_Ntwk_50_21
319	AC_Ntwk_50_20
320	AC_Ntwk_20_13
321	AC_Ntwk_20_12
322	AC_Ntwk_20_10
323	TcpFlLAcRcBytAvg
324	TcpAveWinSzMedian
325	AC_Ntwk_50_47
326	TcpFlLAcRcBytMax
327	AC_Ntwk_20_9
328	TcpPAckCntAvg
329	TcpSeqFaultCntAvg
330	TcpWinSzDwCnAvg

Rank	***Feature***
331	TcpWinSzUpCntAvg
332	TcpWiSzChDiCnMax
333	TcpWinSzChDiCnAvg
334	AC_Ntwk_5_1
335	StdIATStd
336	AvgIATMedian
337	AvgIATAvg
338	AC_Ntwk_50_46
339	TcpSeqSntBytesAvg
340	TcpSeqSntBytesMax
341	TcpPSeqCntAvg
342	ProcessesChgs
343	CreatedPrcs
344	TerminatedPrcs
345	DurationMax
346	DelValueKeyChgs
347	Magic
348	FlsDelete
349	AC _Ntwk_50_44
350	BytAsmMin
351	AC_Ntwk_50_50
352	AC_Ntwk_50_48
353	PktAsmMin
354	BytpsMax
355	AC_Ntwk_50_49
356	MaxPktSizeMax
357	AC_Ntwk_5_2
358	AC_Ntwk_5_3
359	AC_Ntwk_5_4
360	AC_Ntwk_20_1
361	AC_Ntwk_15_14
362	AC_Ntwk_15_13

Rank	***Feature***
363	AC_Ntwk_15_12
364	AC_Ntwk_15_10
365	AC_Ntwk_15_9
366	AC_Ntwk_15_15
367	AC_Ntwk_20_2
368	AC_Ntwk_15 _7
369	AC_Ntwk_20_3
370	AC_Ntwk_20_8
371	AC_Ntwk_20_7
372	AC_Ntwk_20_6
373	AC_Ntwk_20_5
374	AC_Ntwk_20_4
375	AC_Ntwk_15_8
376	AC_Ntwk_15_6
377	AC_Ntwk_5_5
378	AC_Ntwk_10_7
379	AC_Ntwk_10_5
380	AC_Ntwk_10_4
381	AC_Ntwk_10_3
382	AC_Ntwk_10_2
383	AC_Ntwk_10_1
384	AC_Ntwk_10_6
385	AC_Ntwk_10_8
386	AC_Ntwk_15_5
387	AC_Ntwk_10_9
388	AC_Ntwk_15_4
389	AC_Ntwk_15_3
390	AC_Ntwk_15_2
391	AC_Ntwk_15_1
392	AC_Ntwk_10_10
393	indexAvg

Appendix B

Basic Statistics of F-score for each modality individually

This Appendix includes the basic statistics of F-score when using each modality individually (i.e., power consumption, network traffic data, system logs, and code-based static data). Note that the learners in bold are the ones who attained highest performance for that particular modality.

Table B.1: Basics Statistics of F-score using power-based features

Learners	*Mean*	*Median*	*Variance*	*IQR*
RF	0.978	0.978	$1.503 \cdot 10^{-5}$	0.006
J48	0.966	0.968	$3.024 \cdot 10^{-5}$	0.006
JRip	0.973	0.974	$9.651 \cdot 10^{-6}$	0.003
PART	0.964	0.964	$3.333 \cdot 10^{-5}$	0.003
NB	0.936	**0.939**	$6.652 \cdot 10^{-5}$	0.008
SMO	0.953	0.953	$1.370 \cdot 10^{-32}$	0.000
Deep NN	**0.952**	0.959	$5.185 \cdot 10^{-4}$	0.017

Table B.2: Basics Statistics of F-score using network traffic-based features

Learners	*Mean*	*Median*	*Variance*	*IQR*
RF	0.945	0.947	$8.355 \cdot 10^{-5}$	0.005
J48	0.960	0.961	$9.854 \cdot 10^{-5}$	0.008
JRip	0.956	0.960	$7.825 \cdot 10^{-5}$	0.013
PART	0.954	0.951	$8.674 \cdot 10^{-5}$	0.013
NB	**0.818**	**0.821**	$1.882 \cdot 10^{-4}$	0.009
SMO	0.930	0.937	$1.525 \cdot 10^{-4}$	0.018
Deep NN	0.853	0.862	$4.690 \cdot 10^{-4}$	0.035

Table B.3: Basics Statistics of F-score using system logs-based features

Learners	*Mean*	*Median*	*Variance*	*IQR*
RF	0.836	0.835	$7.318 \cdot 10^{-5}$	0.010
J48	0.854	0.857	$9.258 \cdot 10^{-5}$	0.015
JRip	0.838	0.843	$1.638 \cdot 10^{-4}$	0.017
PART	0.805	0.805	$2.924 \cdot 10^{-4}$	0.027
NB	**0.566**	**0.565**	$8.930 \cdot 10^{-5}$	0.008
SMO	0.836	0.836	$0.000 \cdot 10^{0}$	0.000
Deep NN	0.874	0.882	$3.658 \cdot 10^{-4}$	0.003

Table B.4: Basics Statistics of F-score using code-based static features

Learners	*Mean*	*Median*	*Variance*	*IQR*
RF	0.837	0.836	$4.965 \cdot 10^{-6}$	0.002
J48	0.826	0.825	$6.249 \cdot 10^{-6}$	0.003
JRip	0.825	0.822	$1.167 \cdot 10^{-4}$	0.014
PART	0.828	0.828	$1.791 \cdot 10^{-5}$	0.005
NB	0.829	0.830	$3.134 \cdot 10^{-5}$	0.007
SMO	0.839	0.840	$1.496 \cdot 10^{-5}$	0.005
Deep NN	**0.767**	**0.789**	$2.750 \cdot 10^{-5}$	0.000

Appendix C

Basic Statistics for feature level and decision level fusion

This Appendix shows the basic statistics for all learners when all features were combined in one feature vector (feature level fusion). Note that in Tables C.1, C.2, C.3, C.4, C.5, and C.6, FL refers to feature level fusion, while DL refers to decision level fusion.

Table C.1: Basics Statistics of Accuracy

Learners	*Mean*	*Median*	*Variance*	*IQR*
Random Forest-FL	0.967	0.968	$3.267 \cdot 10^{-5}$	0.008
J48-FL	0.967	0.963	$9.290 \cdot 10^{-5}$	0.011
JRip-FL	0.958	0.957	$3.151 \cdot 10^{-5}$	0.008
PART-FL	0.958	0.961	$9.174 \cdot 10^{-5}$	0.011
Naive Bayes-FL	0.942	0.943	$5.120 \cdot 10^{-5}$	0.009
SMO-FL	0.963	0.963	$9.267 \cdot 10^{-5}$	0.015
Deep NN-FL	**0.830**	**0.830**	$3.730 \cdot 10^{-4}$	0.023
Deep NN-DL	0.970	0.977	$1.205 \cdot 10^{-4}$	0.017

Table C.2: Basics Statistics of Recall

Learners	*Mean*	*Median*	*Variance*	*IQR*
Random Forest-FL	1.00	1.00	$0.000 \cdot 10^{0}$	0.000
J48-FL	0.982	0.980	$5.506 \cdot 10^{-5}$	0.007
JRip-FL	0.978	0.974	$4.936 \cdot 10^{-5}$	0.011
PART-FL	0.975	0.974	$9.920 \cdot 10^{-5}$	0.011
Naive Bayes-FL	0.942	0.941	$8.354 \cdot 10^{-5}$	0.015
SMO-FL	0.986	0.987	$2.658 \cdot 10^{-5}$	0.000
Deep NN-FL	**0.923**	**0.933**	$5.062 \cdot 10^{-4}$	0.000
Deep NN-DL	0.963	0.967	$1.111 \cdot 10^{-4}$	0.000

Table C.3: Basics Statistics of Precision

Learners	*Mean*	*Median*	*Variance*	*IQR*
Random Forest-FL	0.955	0.956	$5.488 \cdot 10^{-5}$	0.010
J48-FL	0.971	0.968	$9.02 \cdot 10^{-5}$	0.011
JRip-FL	0.963	0.965	$5.585 \cdot 10^{-5}$	0.011
PART-FL	0.966	0.968	$1.102 \cdot 10^{-4}$	0.010
Naive Bayes-FL	0.974	0.976	$7.00 \cdot 10^{-5}$	0.013
SMO-FL	0.963	0.965	$1.406 \cdot 10^{-4}$	0.017
Deep NN-FL	**0.842**	**0.848**	$3.838 \cdot 10^{-4}$	0.025
Deep NN-DL	0.993	1.000	$1.975 \cdot 10^{-4}$	0.000

Table C.4: Basics Statistics of G-score

Learners	*Mean*	*Median*	*Variance*	*IQR*
Random Forest-FL	0.941	0.944	$1.148 \cdot 10^{-4}$	0.015
J48-FL	0.956	0.953	$1.792 \cdot 10^{-4}$	0.015
JRip-FL	0.944	0.946	$8.25 \cdot 10^{-5}$	0.012
PART-FL	0.946	0.952	$1.933 \cdot 10^{-4}$	0.018
Naive Bayes-FL	0.942	0.940	$8.798 \cdot 10^{-5}$	0.009
SMO-FL	0.947	0.949	$2.592 \cdot 10^{-4}$	0.026
Deep NN-FL	**0.746**	**0.761**	$1.407 \cdot 10^{-3}$	0.052
Deep NN-DL	0.974	0.983	$2.309 \cdot 10^{-4}$	0.013

Table C.5: Basic Statistics of F-score

Learners	*Mean*	*Median*	*Variance*	*IQR*
Random Forest-FL	0.977	0.978	$1.513 \cdot 10^{-5}$	0.005
J48-FL	0.976	0.974	$4.670 \cdot 10^{-5}$	0.008
JRip-FL	0.970	0.969	$1.567 \cdot 10^{-4}$	0.005
PART-FL	0.970	0.972	$4.61 \cdot 10^{-5}$	0.008
Naive Bayes-FL	0.958	0.958	$2.740 \cdot 10^{-4}$	0.007
SMO-FL	0.974	0.974	$4.475 \cdot 10^{-5}$	0.010
Deep NN-FL	**0.881**	**0.882**	$4.475 \cdot 10^{-5}$	0.010
Deep NN-DL	0.978	0.983	$6.570 \cdot 10^{-5}$	0.012

Table C.6: Basics Statistics of FPR

Learners	*Mean*	*Median*	*Variance*	*IQR*
Random Forest-FL	0.111	0.106	$3.597 \cdot 10^{-4}$	0.027
J48-FL	0.068	0.076	$5.229 \cdot 10^{-4}$	0.027
JRip-FL	0.088	0.083	$3.469 \cdot 10^{-4}$	0.027
PART-FL	0.080	0.076	$6.657 \cdot 10^{-4}$	0.023
Naive Bayes-FL	0.059	0.053	$3.801 \cdot 10^{-4}$	0.030
SMO-FL	0.088	0.083	$8.571 \cdot 10^{-4}$	0.042
Deep NN-FL	**0.371**	**0.357**	$3.175 \cdot 10^{-3}$	0.071
Deep NN-DL	0.014	0.000	$9.070 \cdot 10^{-4}$	0.000