5-2024

Side Channel Detection of PC Rootkits using Nonlinear Phase Space

Rebecca Clark
University of South Alabama, rebeccaclark@icloud.com

Side Channel Detection of PC Rootkits using Nonlinear Phase Space

By

Rebecca Clark

A thesis submitted in full fulfillment of the requirements of the University of South Alabama Honors Program and the Bachelor of Sciences degree in the Computer Science Department in the School of Computing

University of South Alabama

Mobile

April 2024

Approved by:

[illegible]

Mentor: Dr. J. Todd McDonald

Ryan Benton

Committee Member: Dr. Ryan Benton

[illegible]

Committee Member: Dr. Michael Black

[illegible]

Honors Dean, Douglas Marshall

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Cyberattacks are increasing in size and scope yearly, and the most effective and common means of attack is through malicious software executed on target devices of interest. Malware threats vary widely in terms of behavior and impact and, thus, effective methods of detection are constantly being sought from the academic research community to offset both volume and complexity. Rootkits are malware that represent a highly feared threat because they can change operating system integrity and alter otherwise normally functioning software. Although normal methods of detection that are based on signatures of known malware code are the standard line of defense, rootkits that have never been seen before (zero-day threats) are not easily defeated because of their ability to evade scanners and present false system information. In this research, we propose to evaluate a novel approach of rootkit detection based on collection of time-serial voltage data from the internal motherboard of standard desktop PCs.

CHAPTER 1

INTRODUCTION

Cyberattacks are increasing in size and scope in recent years. After the Covid-19 pandemic, cyber criminals took advantage of businesses moving to remote work environments. In 2020, malware attacks increased by 358% compared to 2019. Since then, cyberattacks have increased globally by 125% through 2021. The first half of 2022 alone has seen around 236.1 million ransomware attacks globally [1]. With this increasing threat landscape, organizations are realizing that cyber security concerns need action to be taken and are investing in cyber security. Not only are cyberattacks dangerous for companies, they're also expensive. The cost of data breaches to businesses continues to increase and as of November 2022, they cost businesses an average of $4.35 million [1]. The issue is that most organizations feel that they do not have the skills to deal with these data breaches and are unprepared to fight off a cyberattack.

Adversaries have multiple techniques they use to get access to a system, and they often use malware to obtain credentials or maintain access to a system. Malware is software designed to damage or gain unauthorized access to a computer system. A rootkit is a special type of malware that can gain administrative privilege on a system, or "root" privilege. Rootkits typically work by injecting and altering code in key system processes. With this escalation of privilege, they can use stealth to operate without detection, avoiding tools like malware and virus scanners. Because they can operate undetected, an attacker can maintain access for long-term surveillance or data theft and even remotely control your computer. A rootkit can change anything on the operating system (i.e., Windows/Linux) so that your system does not report infection. Their ability to operate undetected can allow long-term surveillance and data theft for any compromised system, thus making rootkits popular in advanced persistent threat (APT) scenarios. Recent statistics [2] also show that 44% of rootkit attacks target government agencies, representing the largest sector of interest. As a study by Positive Technologies notes [3], rootkits have traditionally been designed to gain administrator or system privileges, but a large number now prioritize the ability to evade detection. Given this current trend, the need for robust detection methods remains a top research priority.

1.1 Research Goals

In this research we investigate whether side-channel power data can be used to detect the presence of rootkits executing on standard desktop personal computers (PCs). Prior experiments in this area showed that a simple multimeter that operates at very low frequency and coarse power measurements could be used to detect the execution of certain rootkits using standard machine learning algorithms. In this study, we plan to evaluate whether a more fine-tuned approach with standard time-series data collection using a data acquisition (DAQ) system is feasible and whether a nonlinear phase space approach can be as effective as other machine learning algorithms in detecting whether a rootkit is running on standard desktop computers.

1.2 Research Questions

Our research methodology will seek to answer several questions. We build upon prior work in this area by creating our own experimental laboratory environment for executing live rootkits on real physical desktop PCs and collecting data related to normal and abnormal operation of those machines. In conducting our case study analysis, we expect to answer the following questions:

1) Can we accurately detect rootkits running on a system with out of band voltage data collected from a DAQ and in-line voltage measurements with the PC motherboard?

2) Can we train a nonlinear phase space algorithm to detect rootkits based on these power signatures?
3) Given multiple channels of voltage data, does one type of voltage level perform better than others in terms of rootkit detection?

The rest of this document is organized as follows. Chapter 2 provides background material and related work in the literature for dynamic detection of rootkits using side-channel power collection. Chapter 3 presents our case study methodology which details the rootkit corpus used for experiments, the environmental setup of computers, collection devices, and monitoring software, and the training of a nonlinear phase space algorithm for detecting anomalous behavior from rootkit execution. Chapter 4 presents results of experimental case-studies using three different rootkits along with the associated data. In Chapter 5, we provide our conclusions and recommendations for future work.

CHAPTER 2

BACKGROUND and LITERATURE REVIEW

This chapter provides background information on rootkits and malware detection techniques. It also provides the context of our work related to other researchers in the field.

2.1 Rootkits

There are many kinds of rootkits, including bootloader, firmware, kernel, memory, and application rootkits. A bootloader rootkit infiltrates and replaces the bootloader that loads the operating system on your computer, infecting your computer with malware before you can use the operating system. A firmware rootkit hides in the firmware which is software that provides control over the piece of hardware it's written for. It can affect your hard drive or the system's BIOS, which is important software in your computer's motherboard. As these rootkits target hardware, attackers use them to log keystrokes and monitor online activity. Kernel rootkits attack the core of the operating system, giving an adversary significant control of the system. These rootkits are the most severe because adversaries use them to access files and add code to change the functionality of the operating system [4]. A memory rootkit infects the computer's Random Access Memory (RAM) and can slow your machine down. They use your computer's resources to carry out malicious activities behind the scenes. Luckily, these rootkits disappear as soon as you reboot the system since they hide in your computer's RAM. An application rootkit can modify your regular files [5] and can change the way a standard application like Microsoft Office or Notepad works. Adversaries use these to obtain access to your computer each time an infected program runs.

There are quite a few ways a system can detect a rootkit, including using alternative trusted medium, signature-based detection mechanisms, and behavioral-based detection mechanisms. An alternative trusted medium is another device that is used to scan an infected device. This runs before the system is booted up, preventing a rootkit from using the infected device's operating system to conceal its presence [6]. Of course, this doesn't allow for real-time detection, as the system must be powered off to use this. Signature-based detection mechanisms are effective against well-known rootkits, using signatures as a basis. Signatures are specific patterns that allow detectors to recognize malicious threats, like known malicious instruction sequences used by families of malware. These signatures are from malware that have already been discovered and cataloged as part of a database. Behavioral analysis is a more reliable method for detecting rootkits. Instead of looking for the rootkit itself, this analysis focuses on looking for rootkit-like behavior. This looks for deviant behavior patterns on the system and notices when the system has started behaving out of character. This analysis focuses on how the rootkit acts with its environment, including the file system, the registry, and the network.

Examples of famous rootkits include Stuxnet, Flame, Necurs, and ZeroAccess. Stuxnet is one of the most notorious rootkits and the first rootkit to infect Industrial Control Systems (ICS). It was a malicious computer worm discovered in 2010 that caused significant damage to Iran's nuclear program. Flame was a rootkit used for espionage in the Middle East that was discovered in 2012. It was able to monitor traffic, capture screenshots of the system and record audio, and log keystrokes. Necurs was reportedly detected in 83,000 infections in 2012 and was technically complex and able to evolve. In 2011, the kernel rootkit ZeroAccess was discovered and found to have infected more than 2 million computers worldwide. This rootkit downloaded and installed malware on the infected machine and would make it part of a global botnet to carry out cyberattacks [4].

Rootkits are dangerous tools that adversaries use in many malicious ways. Despite the many ways to detect them, there is a need for an out-of-bound detection mechanism that is robust against the evasion techniques rootkits use. My research focuses on using low-frequency side-channel power data to detect rootkits with a nonlinear phase space approach.

2.2 Malware Detection Techniques

Static detection is a malware detection method that occurs while the system is not running. Static analysis can identify malicious infrastructure, libraries, or packed files using indicators like file names, hashes, strings, domains, and file header data [7]. Virus and malware scanners that come with your operating system typically rely on static detection. There has been much research done in the static analysis of rootkit detection that could be implemented into those virus and malware scanners. Alazab et al. focuses on detecting zero-day rootkits using application program interface (API) calls and data mining techniques. Their method has a true positive rate of more than 98.5% and a false positive rate of less than 2.5% [8]. They are able to perform a deep analysis of the code and its properties while offline.

Additionally, Liang et al. focuses on creating a detection prototype system called DeepScanner that effectively detects stealthy malware using inter-structure and imported signatures. It uses a hypervisor-based monitor to protect those imported signatures. However, they also note that detection can be compromised by rootkits designed specifically to tamper with those detection mechanisms, showing a need for a detection mechanism outside the operating system [9]. Static analysis allows for an in-depth view at what a rootkit is doing, but this detection mechanism cannot occur in real-time, requiring the system to be off. However, polymorphic malware can bypass this traditional signature-based detection mechanism, and researchers have tried to combat this with different dynamic detection mechanisms [10].

Dynamic analysis relies on detecting malware while the system is running. This method focuses on analyzing how the malware is behaving as it's executing. This allows for an in-depth look at the runtime behavior of the malware as it's running. Because dynamic analysis allows for real-time detection, there has been lots of prior work into this method that consider several approaches. For example, Lobo et al. focuses on creating a dynamic detection algorithm on the operating system level for rootkit classification into malware categories that anti-virus software typically uses. Using Rootkit Behavioral Analysis Classification Systems (RBACS), they focus on Windows processes that are hooked by injected rootkit code [11]. The issue with this method is that rootkits can be designed to modify this detection mechanism.

Khasawneh et al. considers using Hardware Malware Detectors (HMDs) that use low-level features to detect malware. They prove that existing HMDs can be reverse-engineered and evaded, allowing malware to avoid detection. Their work focuses on creating a new type of resilient HMD or RHMD that switches between different detectors to resist reverse engineering and evasion [12]. Singh et al. focuses on using hardware performance counters (HPCs) to detect hooking rootkits through behavioral detection. With an accuracy of over 99%, they are able to detect zero-day rootkits, or those that had never been seen before. However, when detecting zero-day rootkits, the rootkit has to have been using previously seen attack mechanisms [13]. Despite these results, HPCs are most impacted by a hooking mechanism and this detection will not necessarily work for other types of rootkits. For example, rootkits employing a direct kernel object manipulation do not have a significant impact on HPCs and will not be detected. This shows that there is a need for a more robust out-of-bound rootkit detection mechanism.

2.3 Side Channel Data Collection

Side-channels are mostly associated with side-channel attacks which are any attacks based on extra information that is leaked from measuring or analyzing various physical parameters like supply current, execution time, or electromagnetic emission. However, many studies have used side-channel data for cyber event detection. For instance, Pham et al. presents a novel approach to detecting malware on Internet of Things (IoT) devices that uses electromagnetic signal acquisition to obtain information about the malware type and identity. They can avoid any obfuscation techniques that could prevent static or symbolic binary analysis. From 30 different malware samples, they can predict three generic malware types with an accuracy of 99.82%, demonstrating the potential for side-channel detection mechanisms resistant to obfuscation techniques [14].

Additionally, Hernandez et al. focuses on using a different side-channel parameter: power. Theoretically, malware is just more code running on top of the system's code, thereby producing more power. Hernandez proposes that by going outside of the system being monitored we can see this spike of power and detect an anomalous event. Hernandez proves that malware does indeed leave a signal on the power consumption of a general PC [10], but does not consider different states of the system. Furthermore, Luckett et al. focuses on using low-frequency power data and various machine learning algorithms to detect rootkits. With a 97% accuracy rate using neural networks, Luckett successfully validated the power data approach to detect rootkits [15]. However, the machine learning algorithms Luckett used for detection varied based on operating systems, as different operating systems performed better with different machine learning algorithms. This shows that there may be a better approach than traditional machine learning algorithms for rootkit detection.

2.4 Nonlinear Phase Space Analysis (NLPSA)

We will focus on using a nonlinear phase space analysis (NLPSA) to analyze our data developed by Hively et al. [16] – [18]. This nonlinear approach demonstrates great flexibility across a variety of domains including biomedical, industrial, and cyberspace for detecting anomalous behavior. A phase space is a representation of all possible systems states. These states represent unique points, which ultimately become vertices in a graph. Figure 1 provides a graphic representation of NLPSA for phase space creation. Table 1 describes the trainable parameters of NLPSA. The assignment of values to each parameter allows for processing of cutsets in each observation file, allowing creation of phase space graphs for each cutset.

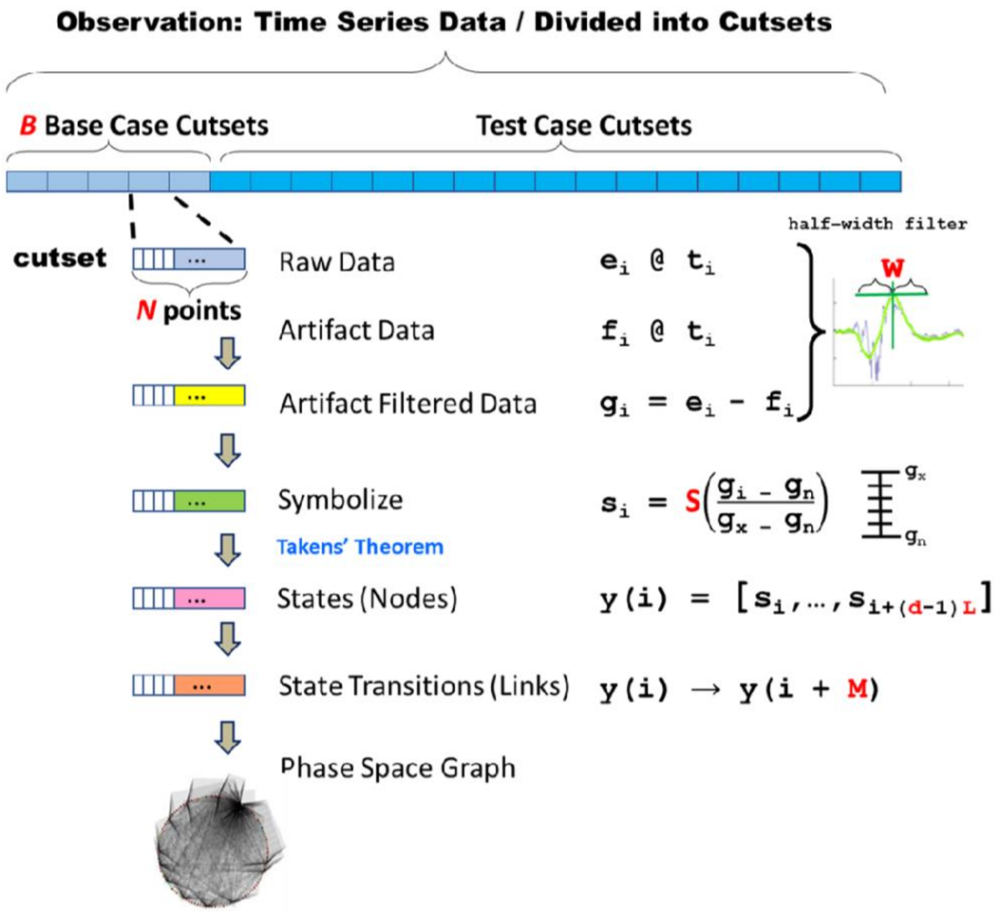
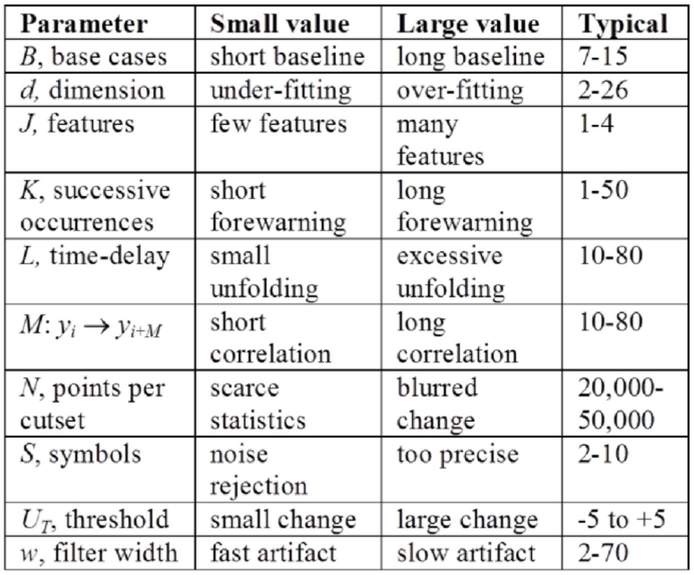


Figure 1: Nonlinear Phase Space Analysis

Table 1: Trainable Parameters



Parameter	Small value	Large value	Typical
B, base cases	short baseline	long baseline	7-15
d, dimension	under-fitting	over-fitting	2-26
J, features	few features	many features	1-4
K, successive occurrences	short forewarning	long forewarning	1-50
L, time-delay	small unfolding	excessive unfolding	10-80
M: $y_i \rightarrow y_{i+M}$	short correlation	long correlation	10-80
N, points per cutset	scarce statistics	blurred change	20,000-50,000
S, symbols	noise rejection	too precise	2-10
U_T, threshold	small change	large change	-5 to +5
w, filter width	fast artifact	slow artifact	2-70

We can reconstruct a phase space by sampling time series data uniformly before applying Takens' time delay embedding theorem to form state vectors. With multiple dimensions, these vectors collectively represent each specific state. We can analyze the change in the dynamics of a system over time by looking at the succession of those state vectors. For each phase space graph, we can look at the graph features and compare them against other phase space graphs to detect the change from normal to abnormal behavior (i.e., a rootkit executing).

An observation is a set of time-series data that is sampled at a regular frequency from a system. A single observation represents some time where the system is either in a normal or potentially abnormal state. We split observation data into cutsets, which are equal partitions containing *N* data points each. Data are sampled at time t_i, giving raw data points for each cutset, $e_i = e(t_i)$. NLPSA then treats some *B* number of cutsets at the beginning of each observation as normal or baseline behavior (base-cases) and the remaining cutsets in an observation as test-cases for training.

We process each cutset individually by first filtering noise and artifacts from the data using a zero-phase quadratic filter, fitting the e_i-data to a parabola in a least-squares sense and using the central point as an estimate of the artifact (f_i) data. The filter then calculates artifact f_i for each data point e_i, leaving a filtered value $g_i = e_i - f_i$ (Artifact Filtered Data). We apply symbolization to each data point to help normalize the data. One of a predefined number of symbols (S) is substituted for each data point (g_i) leaving a symbolized value s_i. The number of symbols is predetermined such that s_i is one of S different integers 0, 1, …, $S - 1$:

$$0 \leq s_i = INT\left[S\frac{g_i - g_{min}}{g_{max} - g_{min}}\right] \leq S - 1$$

The INT function rounds down a value to the lowest integer. g_{max} and g_{min} represent the largest and smallest values of g_i. Takens' theorem allows for dynamical reconstruction by a time-delay embedding that is smooth and non-intersecting. After symbolization, symbolized data are converted into vectors y_i:

$$y_i = [s_i, s_{i+L}, \dots, s_{i+(D-1)L}]$$

The y_i states capture the topology of the underlying dynamics. This dimension is a trainable parameter to achieve the best detection. L is the time-delay lag, which cannot be too small (as s_i and s_{i+L} would be indistinguishable) or too large (as s_i and s_{i+L} would become independent by long-time unpredictability). D is the embedding dimension, which must not be too large to avoid over-fitting but large enough to capture the dynamics. The y_i states form nodes in the phase space with $y_i \Rightarrow y_{(i+M)}$, forming state-to-state links for some value M. An undirected graph $G = (Y, E)$ is formed, with Y representing the set $Y = \{y_1, y_2, \dots, y_n\}$ of phase space nodes and E representing the set $E = \{e_1, e_2, \dots, e_n\}$ of edges that connect two of the phase space nodes. The set of nodes Y have topologically-invariant properties independent of unique labeling.

2.5 Graph Dissimilarity

We use features of the phase space graphs as a basis for determining how the system is changing from normal behavior. Graph invariant properties (features) are computed for the base-case cutset graphs and the test-case cutset graphs of each observation which is then used as the basis for determining whether the system has changed in behavior due to malware possibly executing. Each graph dissimilarity feature is computed for phase space graphs in each observation. Base-case graphs are compared amongst themselves in $(\frac{B}{2})$ fashion (comparing *B* graphs two at a time). A base-case dissimilarity average $\underline{u}$ and standard deviation $\sigma_{\underline{u}}$ are computed from the $(\frac{B}{2})$ comparisons. Each test-case graph is then compared against each base-case graph to compute graph differences based on the feature being considered, producing dissimilarity values u_k. We then computer a normalized dissimilarity measure for the test-case graph as $\underline{u_k} = (u_k - \underline{u})/\sigma_{\underline{u}}$.

We will use the following 16 graph invariant feature differences for comparing any two graphs, *A* and *B*.

Node Directed (NDD): the number of (unique) nodes in graph *A* that are not in *B*: $|(Y_A \setminus Y_B)| / |Y_A|$

Node Inverted (NDI): the number of (unique) nodes in graph *B* that are not in *A*: $|(Y_B \setminus Y_A)| / |Y_B|$

Link Directed (LKD): the number of (unique) links in graph *A* that are not in *B*: $|(E_A \setminus E_B)| / |E_A|$

Link Inverted (LKI): the number of (unique) links in graph *B* that are not in *A*: | (E_B \ E_A) | / | E_B |

Algebraic Connectivity (ACD): *AC* is the second-smallest eigenvalue of the Laplacian matrix of some graph *G*. *AC* difference is computed as | *AC(AC) – AC(B)* | or 0 if either graph does not give at least two Laplacian eigenvalues.

Adjacency Symbol Eigenvalue (ASE): Given the ordered vector of eigenvalues of the adjacency matrices for graph *A* and *B*, notated as eig_A and eig_B, we compute the difference as | *sym*(eig_A) – *sym*(eig_B) |. The function *sym* is computed as:

$$sym(eig) = \begin{cases} 0, & if\ |eig| < 3 \\ \sum_{n=1}^{|Y|} (\lambda_n - \lambda_{|eig|-n-1})^2, & otherwise \end{cases}$$

The number of unique nodes/links in each cutset can vary widely. Consequently, the normalized form of the adjacency matrix (which is used in the entropy feature) is preferred.

Normal Laplacian Eigenvalue (NLP): Computed in the same manner as the adjacency symbol eigenvalue (AdjSymEig) dissimilarity, with the only difference being the vector of normalized Laplacian eigenvalues is used for values. The difference is computed as $\sqrt{count}$, where $count = \sum_{n=1}^{|Y|} (\lambda_{A_n} - \lambda_{B_n})^2$

Entropy (ENT): Measures complexity of a graph based on its node and link distributions. We compute the entropy of each graph *A* and *B* and calculate the difference as: | *ent*(*A*) – *ent*(*B*) |. We calculate the entropy of a graph *G* based on the vector of its normalized Laplacian eigenvalues ($norm_G$ ∈ R) taken from the normalized Laplacian matrix. Entropy is computed given the normalized eigenvalue vector (*norm*) as follows:

$ent(G) = \sum_{n=1}^{|Y\}} norm[n] * (ln(norm[n])/ln(2))$

Triangles (TRI): A triangle in a graph is a set of three vertices *u, v, w,* where every two of the three are connected by an edge. They are often used for complex network analysis in fields of study related to graph mining. A two-dimensional array stores all triangles for a graph, derived from its adjacency matrix. Given the set of all triangles tri_A and tri_B, we computer triangle difference as $| tri_A \cap tri_B | / | tri_A \cup tri_B |$

Singular Value Decomposition (SVD): Compares the *n* largest singular value decomposition of two adjacency matrices via Euclidean distance, where *n* is the size of the smaller list of singular values. The singular value decomposition of the adjacency matrix is computed for each adjacency matrix and the difference is computed as $\sqrt{count}$, where $count = \sum_{x=1}^{n} (svd_A[x] - svd_B[x])^2$

Average Centrality: Centrality is a graph feature that identifies important vertices (nodes) in a graph and is used to measure how “central” a node/edge is among other nodes/edges in the graph. Average centrality is thus computed by taking the mean of a characteristic property for all nodes or edges in the graph. We consider six different versions of this feature:

- **Graph Centrality (CGR)**: the reciprocal of the maximum of all shortest path distances from a vertex to all other vertices in the graph. Vertices with high graph centrality have short distances to all other vertices in the graph.
- **Degree Centrality (CDG)**: computes the average degree of all vertices in the graph where degree is the number of vertices adjacent to a vertex.

- **Closeness Centrality (CCL)**: the reciprocal of the sum of shortest path distances of a vertex to all other vertices in the graph. Vertices with high closeness centrality have short distances to all other vertices of a graph.

Betweenness centrality is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that the number of edges that the path passes through is minimized. The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex.

- **Node Betweenness Centrality (CNB)**: computes how often a vertex lies on a shortest path between each pair of vertices in the graph.
- **Edge Betweenness Centrality (CEB)**: computes how often an edge lies on a shortest path between each pair of nodes in the graph.
- **NodeEdge Betweenness Centrality (CNE)**: combines node and edge betweenness together.

2.6 Related Phase Space Experiments

Hernandez et al. demonstrates a novel approach for cyber event detection using a theorem-based, data-driven, phase space analysis. Using time series data, they create a prototype implementation for intrusion detections by measuring the precise execution time of events in energy delivery systems. By establishing a baseline execution profile and capturing the effect of perturbations in the state from malware, they extract the changing dynamics of the system to determine that malware is running on it. They prove that a nonlinear approach can detect anomalous events. Similarly, Dawson et al. uses the same nonlinear, phase space algorithm developed by Hively on low-frequency power data. Using the rootkit kBeast on the Ubuntu operating system, Dawson produces a true rate of about 80% or better for about 0.69% of the tested parameter sets of the algorithm [19]. As Dawson only uses a prediction version of this algorithm that was previously used for seizure prediction, this shows the need for a classical detection algorithm for the nonlinear phase space algorithm to consider any detection after the rootkit would have been installed.

CHAPTER 3

METHODOLOGY

This chapter provides an overview of our laboratory setup for experiments as well as details our data collection methods. It also covers details about phase spacing training.

3.1 Laboratory Setup

Our experimental framework involves the setup of a PC testbed configured with power sensors for data collection shown in Figure 2. Figure 3 shows how we modified Luckett's testbed environment into the environment we will use. The testbed is not connected to the university network to ensure that the rootkit does not infect it. The testbed computer is a x86 (32-bit) Dell Optiplex 320 tower with 8 GB RAM and 3 GHz Pentium processors.

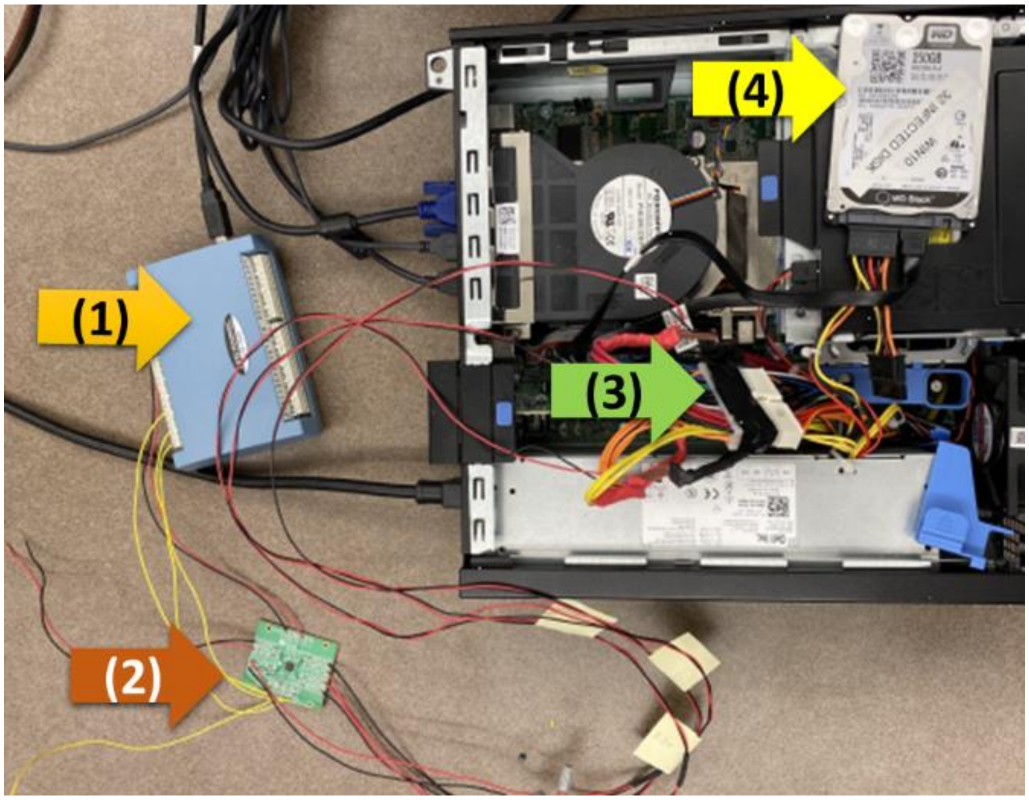


Figure 2: Testbed Layout

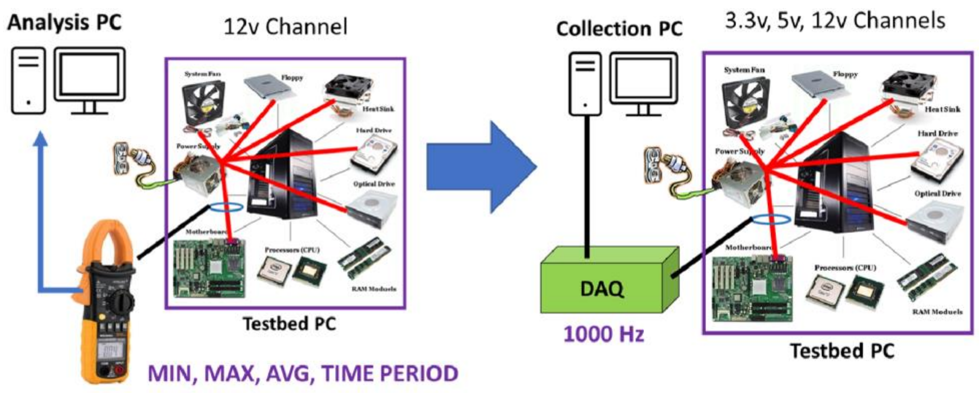


Figure 3: Testbed Environment

In the PC, we use a Standard Western Digital 250GB hard drive for operating system installation. All operating systems are installed fresh, with no patches, onto a master disk drive. Copies are made from this master drive and each drive is zeroized after each rootkit injection experiment. We use a disk jockey to copy and clean the hard drives as seen in Figure 4.

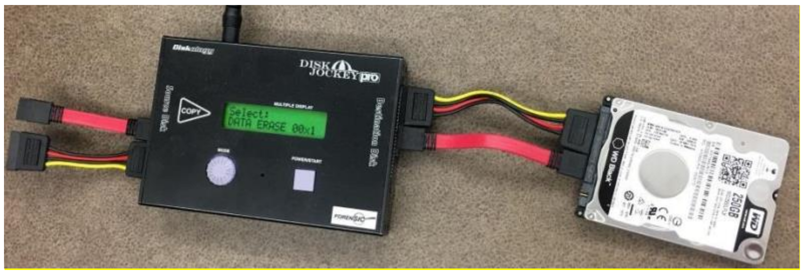


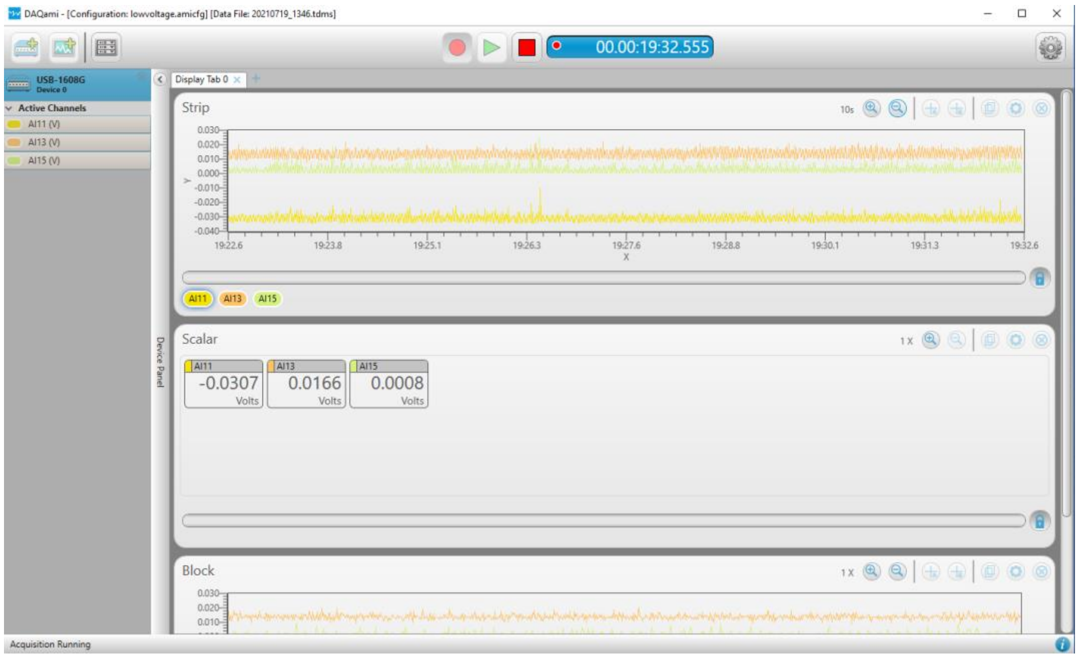
Figure 4: Disk Jockey

Attached to the PC testbed, there is an adaptor for the standard ATX 24-pin power supply adapter (number 3 in Figure 2). This adaptor provides a voltage and ground signal across a resistor shunt from three channels that are normally connected across the ATX pin. This feeds into a Texas Instruments INA4180-A4 current-sense amplifier (number 2 in Figure 2). This will sense any voltage drops across current-sense resistors at common-mode voltages from -0.2V to +26V. The three channels we look at are the 3.3VDC, 5VDC, and 12VDC from the internal PC power supply. These channels then feed into the Measurement Computing USB-1608G Series

Data Acquisition System (DAQ) version 6.72, which is number 1 in Figure 2. The TracerDAQ software is used to acquire, analyze, and display the data and is shown in Figure 5.

<u>3.2 Data Collection</u>

To collect the data, experiments are run using the data acquisition software on low-frequency voltage channels while running the operating system on the test machine in one of three activity states (normal, manual, or stress) and either with or without a rootkit executing (referred to as clean or infected). Each collection runs for one hour at a rate of 1000 Hz with a range of $\pm$10V. The tests for the different activity states are summarized below. The DAQ collection software can visualize collected data that is transferred from the DAQ to the collection PC. A visual of the DAQ capture software is seen in Figure 5.


DAQami - [Configuration: lowvoltage.amicfg] [Data File: 20210719_1346.tdms]

00.00:19:32.555

USB-1608G
Device 0
Active Channels
AI11 (V)
AI13 (V)
AI15 (V)

Display Tab 0

Strip

Scalar

AI11	AI13	AI15
-0.0307 Volts	0.0166 Volts	0.0008 Volts

Block

Acquisition Running

Figure 5: DAQ Software

Normal Collection: The system is kept at a constant rate where the OS operates without any user interaction.

(1) Begin recording using the DAQ;

(2) Power on the computer and log-in;

(3) Allow computer to run for 57 minutes;

(4) Turn off the computer after 57 minutes;

(5) Recording on the DAQ ends after 1 hour.

Manual Collection: These tests follow a procedure where basic tasks are conducted within different applications on the operating system. For Step 3, Ubuntu application equivalents are performed for Ubuntu Desktop and Ubuntu Server operating systems.

(1) Begin recording using the DAQ;

(2) Power on the computer and log-in;

(3) Open the following and perform each activity for 5 minutes each and then close: Notepad, PaintDraw, Calculator, Control Panel, Paint, Notepad, WordPad, Paint, Control Panel, WordPad;

(4) Turn off the computer after 57 minutes;

(5) Recording on the DAQ ends after 1 hour.

Stressed Collection: Stress tests are performed three times for five minutes each. HeavyLoad is used for the Windows operating system and stress-ng is used for Ubuntu Desktop and Ubuntu Server.

(1) Begin recording using the DAQ;

(2) Power on the computer and log-in;

(3) Start HeavyLoad/stress-ng and run for 5 minutes after 10/25/40 minutes have passed;

(4) Turn off the computer after 57 minutes;

(5) Recording on the DAQ ends after 1 hour.

After all data is collected, we convert the data into a format that can be read by the nonlinear phase space algorithm. A Java program separates the channel values into separate files based on the different voltage channels. We run four collections for each type of test for both uninfected and infected states. This results in 72 raw data files per operating system. We then trim each data file to remove data signals while the test machine is powered off. Only data from the same test type are combined, whether under normal (U), manual (M), or stressed (S) conditions. For our last step, we create uniform size time-series observation files so that event (infected) and non-event (uninfected) data files have the same record size. Event files contain 3,300,000 points of nominal (uninfected) data from raw files of the same test type. All observation files have 6,600,000 data values after being combined. Data are collected at 1000 Hz, meaning 1000 data values translates to 1 second of wall clock time. The event (rootkit infection) occurs after 3,300 seconds, or at the 3,300,000 data point of each event file.

3.3 NLPSA Training

For training, we run NLPSA experiments where a random assignment of values is made to the parameter space. We utilize a Dell Precision T5600 workstation with 128GB RAM and 64 cores for training with NLPSA. We train each operating system and voltage channel independently, resulting in six categories of training experiments. The parameters were assigned random values in the following ranges: $36 \leq B \leq 36$, $90000 \leq N \leq 89000$, $2 \leq d \leq 5$, $2 \leq s \leq 5$, $30 \leq w \leq 90$, $30 \leq L \leq 90$, and $30 \leq M \leq 90$. A set of observation files created by pre-processing described in Section 3.2 was used for each training experiment. They both had a total of 8 (4 event, 4 non-event). For FUTo, we ran 1000 training experiments per voltage rail, evaluating a total of 3000 parameter sets. For Azazel, we ran a total of 100 training experiments, evaluating a total of 299 parameter sets. Lastly, we ran a total of 100 training experiments for kBeast, evaluating a total of 300 parameter sets. For each of the 3599 NLPSA training experiments, each training experiment also evaluates all 16 graph dissimilarity features in combinatorial fashion, resulting in $2^{16} = 65,536$ combinations. Thus, we evaluated a total of 235,864,064 possible sets of graph dissimilarity values that are processed by the detection algorithm.

We use a threshold and successive occurrences above threshold (SOAT) approach to classify observation files as either normal or anomalous, which is visualized in Figure 6. The algorithm trains the assignment of an independent threshold value for each normalized dissimilarity feature produced per cutset, per observation. After a threshold is assigned, values are either above or below that threshold, represented as a 1 or 0. The algorithm then optimizes the detection accuracy of each feature individually by brute-forcing all possible threshold values based on the underlying normalized dissimilarity values and the best detection accuracy achievable. A single plot combines the best-possible threshold values for each feature. Further analysis is conducted on successive occurrences to find the optimal occurrence number above threshold with the highest accuracy.

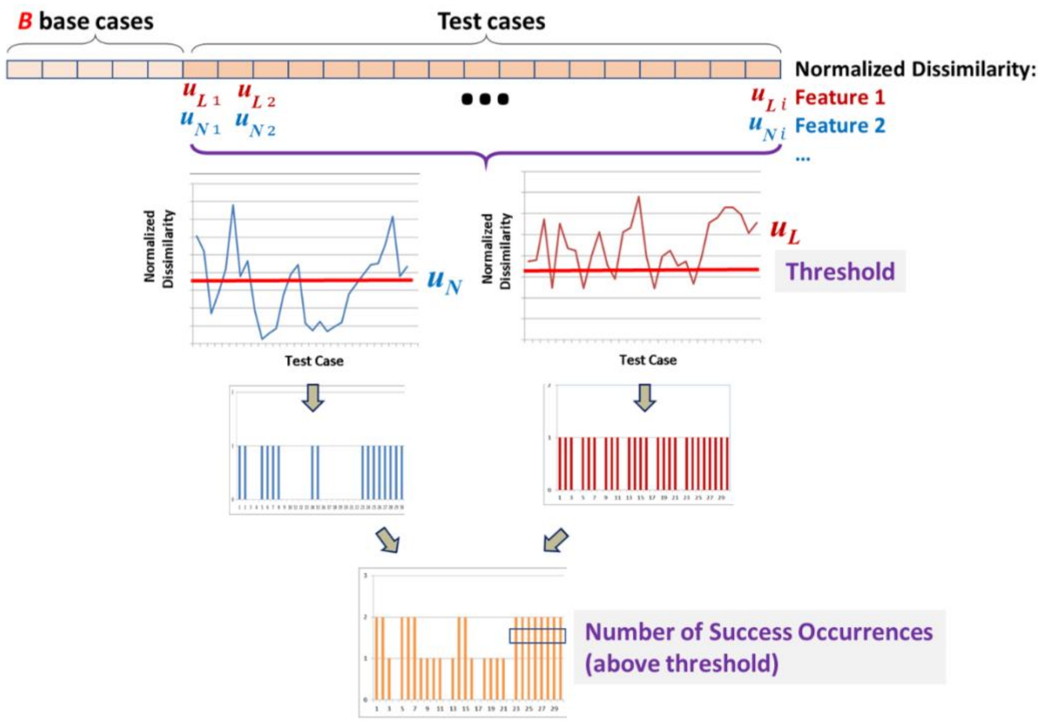


Figure 6: Visual of Threshold and SOAT Analysis

We use the combination of two common measures for statistical evaluation of accuracy. The number of true positives (*TP*) for known event data sets (*EV*) yields the true positive rate (sensitivity) of *TP/EV*. Next, the number of true negatives (*TN*) for known non-event data sets (*NEV*) yields the true negative rate of *TN/NEV* (specificity). We aim to minimize the detection distance (*DD*):

$$DD = \{\left[1 - \frac{TP}{EV}\right]^2 + \left[1 - \frac{TN}{NEV}\right]^2\}^{1/2}$$

We feed the full set of graph dissimilarity feature values into the detection algorithm, training both the individual feature thresholds and SOAT values for the highest detection accuracy. It considers all possible combinations of features, resulting in 2^k comparisons given *k* features. The algorithm ultimately assigns guesses for each observation file in the training set, which comprises two classes: no detection or a detection based on a specific cutset. The time between the detection guess and actual event (if present) is the time to detect (*minFW)*. The algorithm is optimized for min_{DD} = 0.0 while considering the minimum of *minFW* as factors in evaluating training experiments.

CHAPTER 4

RESULTS

We use a receiver operating space (ROS) for detection distance that was evaluated during the experiments using NLPSA with the rootkits: FUTo, Azazel, and kBeast. On the ROS, a dot shows that some number of parameter sets created the appropriate specificity and sensitivity. For rootkits FUTo and Azazel, we found parameter sets that produced a minimum detection distance (DD) = 0.0 for nearly all three voltage channels, indicating a perfect true positive rate (TPR) and perfect true negative rate (TNR). This is indicated by the red dots on the ROS at the point (0, 1) seen in Figures 7, 8, 9, 10, 11, and 12.

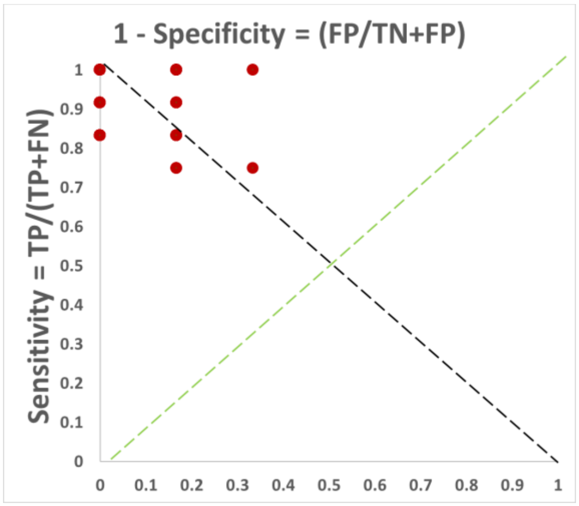


Figure 7: ROS for FUTo 12V

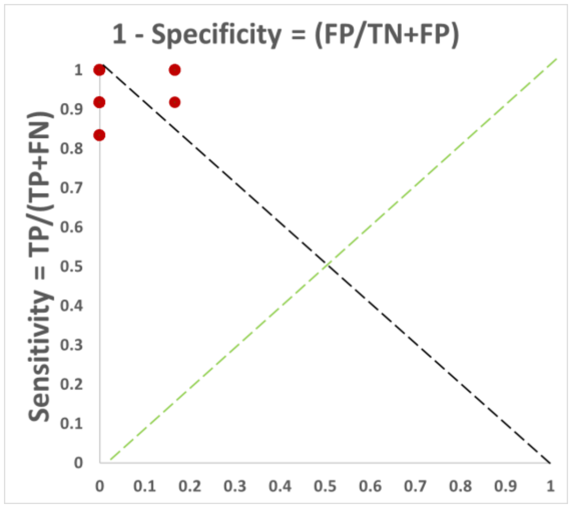


Figure 8: ROS for FUTo 5V

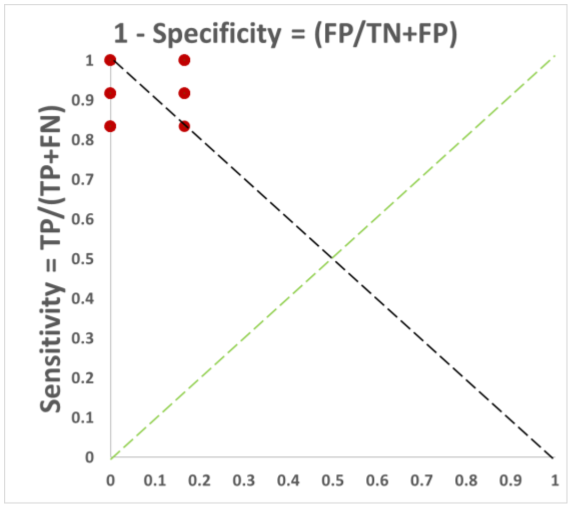


Figure 9: ROS for FUTo 3.3V

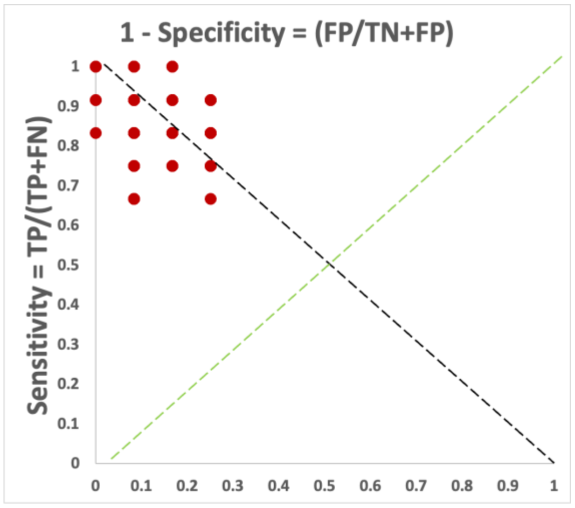


Figure 10: ROS for Azazel 12V

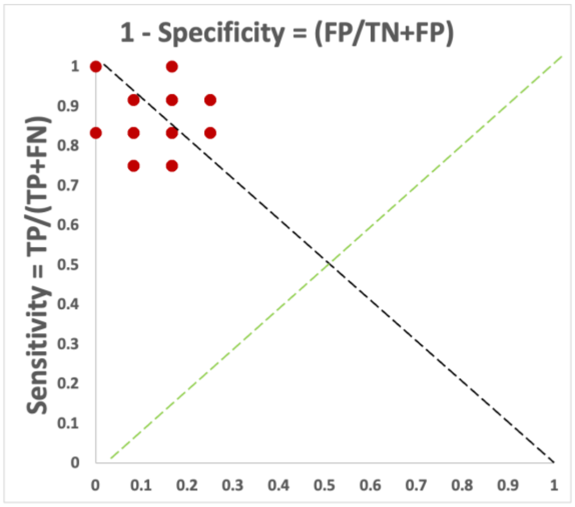


Figure 11: ROS for Azazel 5V

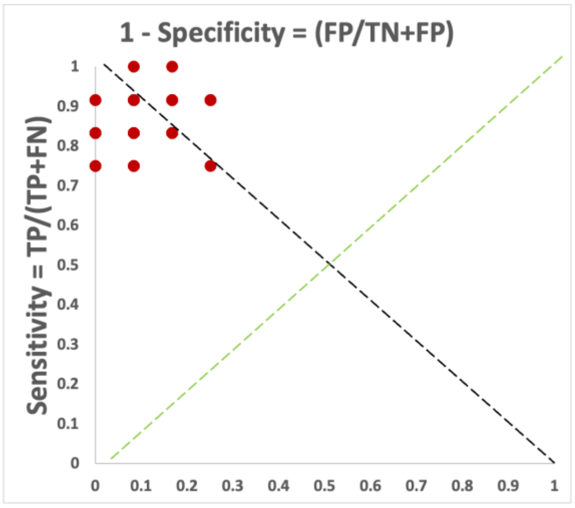


Figure 12: ROS for Azazel 3.3V

Although none of the 100 parameters for kBeast produced a detection distance of 0, the 5V and 12V channels produced results with DD = 0.083, which means it only had one FP or one FN. The 3.3V channel produced results with DD = 0.186, which means that it still had a majority of TP's and TN's. The receiver operating space can be seen in Figures 13 – 15.

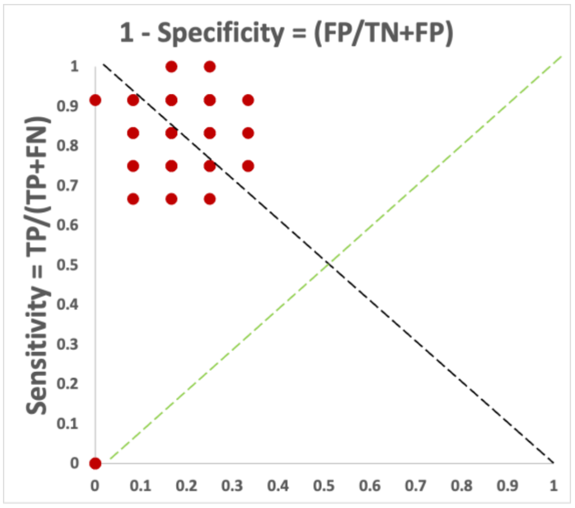


Figure 13: ROS for kBeast 12V

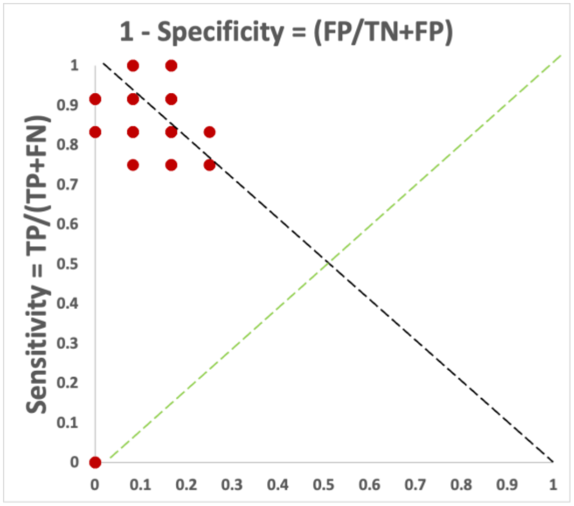


Figure 14: ROS for kBeast 5V

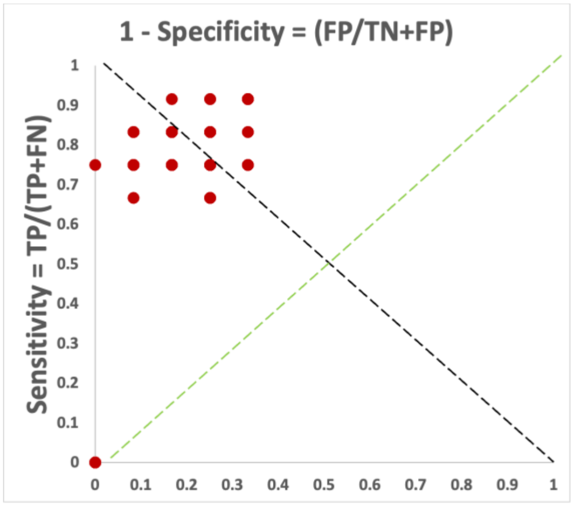


Figure 15: ROS for kBeast 3.3V

1,908 of the 3599 parameters we evaluated produced perfect detection (DD = 0.0) for the Azazel and FUTo rootkits. Each parameter set involved combinatorial evaluation of all 16 graph dissimilarity features, but not all 65,536 combinations of graph features per parameter set were responsible for minimum detection distance. This means that the minimal detection distance is recorded for each parameter set and that more than one combination of graph dissimilarity features may have produced the minimal. Figure 16 and Table 2 provide a summary of detection distances for all 3599 parameter sets that we evaluated.

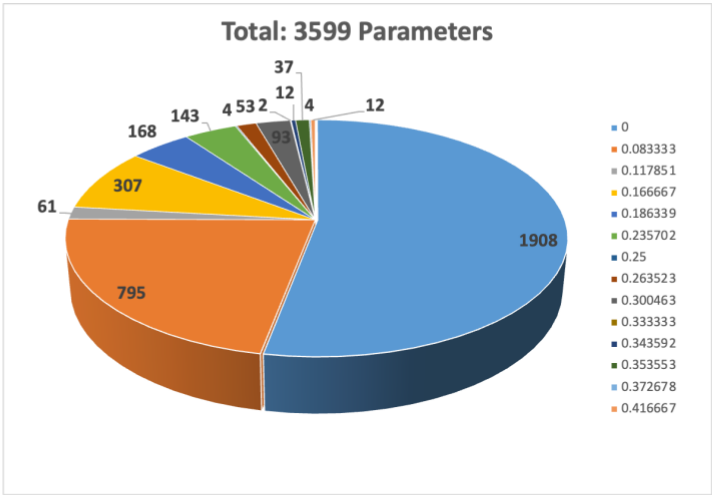


Figure 16: Detection Distance Summary for All Rootkits

Of note, for FUTo, at least 80% or more of the 1000 parameter sets for the 5V channel we tested produced min_{DD} = 0.0, as seen in Table 2. As the 3.3V channel for Azazel did not produce min_{DD} = 0.0, the percentage of parameter sets was 2% that achieve the best accuracy of min_{DD} = 0.083. None of the 100 parameter sets for kBeast produced a min_{DD} = 0.0 result, but the percentage of parameter sets that produced the best accuracy of min_{DD} = 0.083 are shown in Table 3 for kBeast. The 3.3V channel had a best prediction distance of min_{DD} = 0.19 for 3% of its parameter sets.

Table 2: Percent of Parameter Sets with min_{DD} = 0.0

	3.3V	**5V**	**12V**
Azazel (of 100)	0%	1.01%	3%
FUTo (of 1000)	56.9%	81.5%	52.0%

Table 3: Percent of Parameter Sets with minDD = 0.083 for kBeast

	3.3V	**5V**	**12V**
kBeast (of 100)	0%	9 %	1%

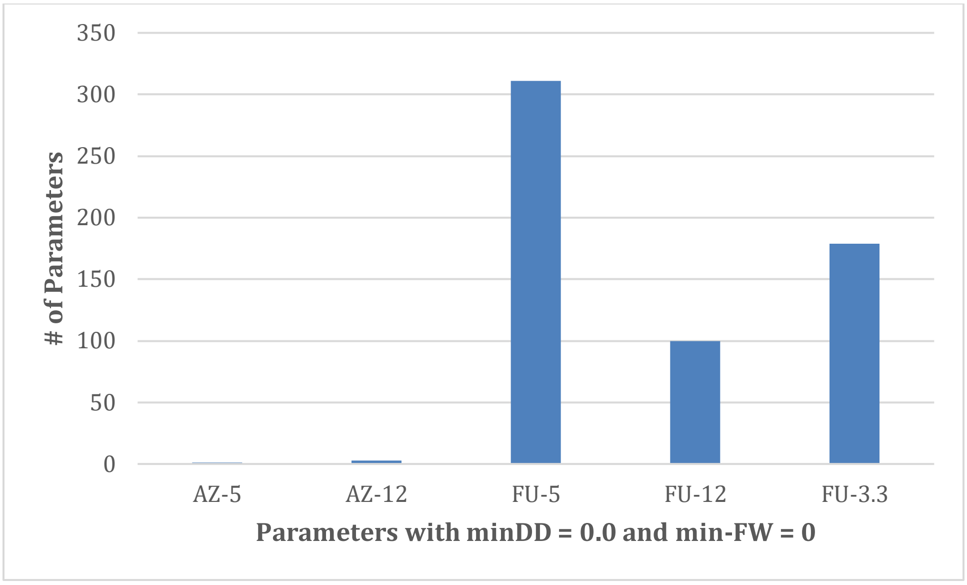


Figure 17: Parameters with min_{DD} = 0.0/minFW = 0

Figure 17 shows the number of parameters across the five categories that produced both perfect accuracy and detection within the same cutset, or where *minFW* = 0. Figures 18 and 19 show further analysis of best parameter sets producing minimal detection distance (perfect accuracy) by showing the distribution of graph dissimilarity features that were associated. They show specifically how many instances of numbers of features (1 to 16) were combined to produce the optimal results across the each rootkit and their voltage channels. For each figure shown, these are the graph dissimilarity features that produced the best accuracy result and detected rootkit infection within the same cutset, or where *minFW* = 0. As the 3.3V channel for Azazel and the channels for kBeast did not produce min_{DD} = 0.0, Figures 20 – 22 show the best accuracy result of either min_{DD} = 0.083 or min_{DD} = 0.19 for those specific channels.

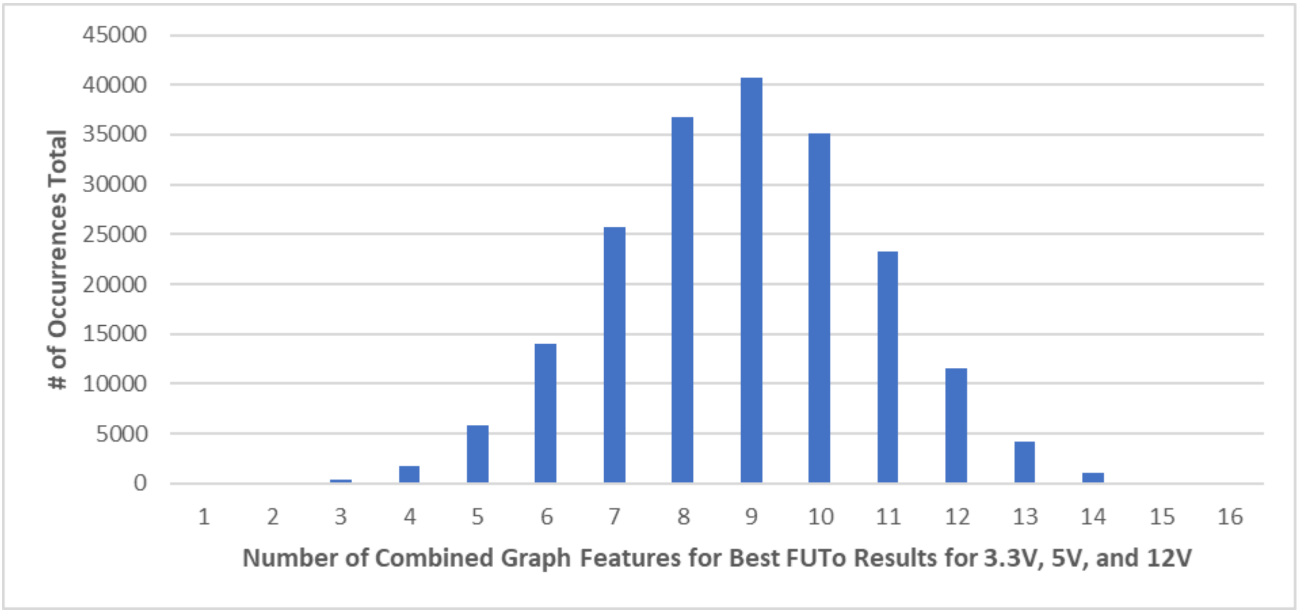


Figure 18: Combined Features Producing $\min_{DD}$ = 0.0 for FUTo

Figure 18 has a total of 200,500 graph features for the 590 parameter sets that produced a perfect accuracy result and determined rootkit execution within the same cutset, and this is the only rootkit result that includes graph feature combinations ranging from 1 to 16.

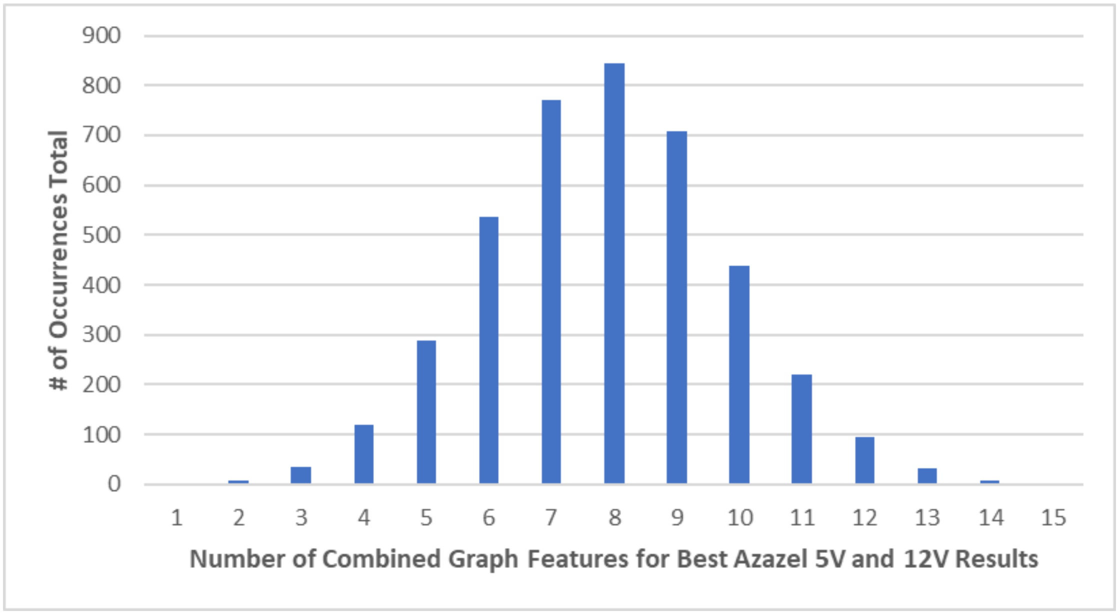


Figure 19: Combined Features Producing minDD = 0.0 for Azazel 5V and 12V

Figure 19 has a total of 4,100 graph features for the 6 parameter sets that have perfect accuracy and determine rootkit execution within the same cutset. The 5V and 12V Azazel graph features include graph combinations from 1 to 15.

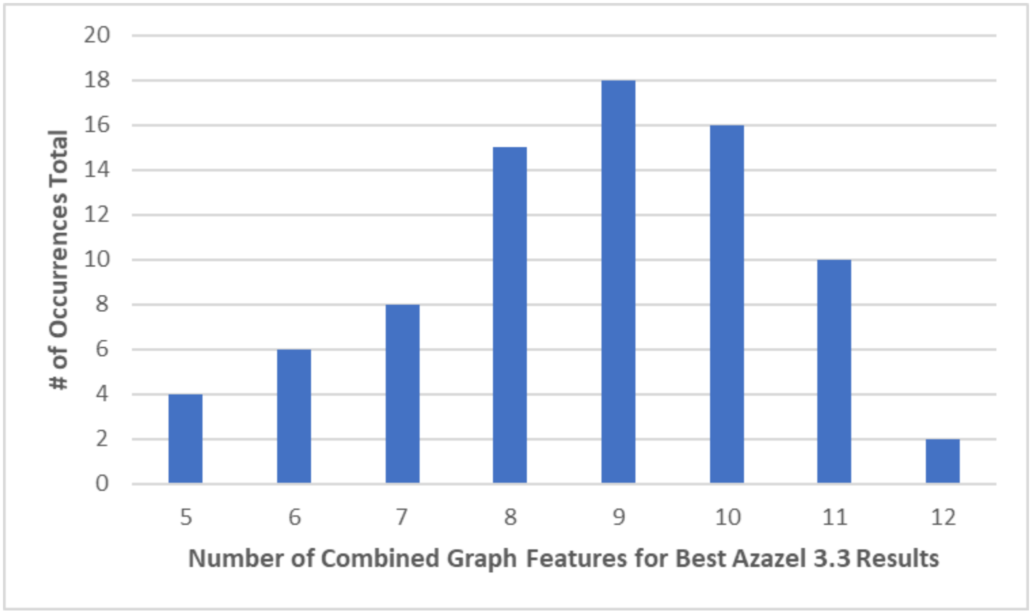


Figure 20: Combined Features Producing minDD = 0.083 for Azazel 3.3V

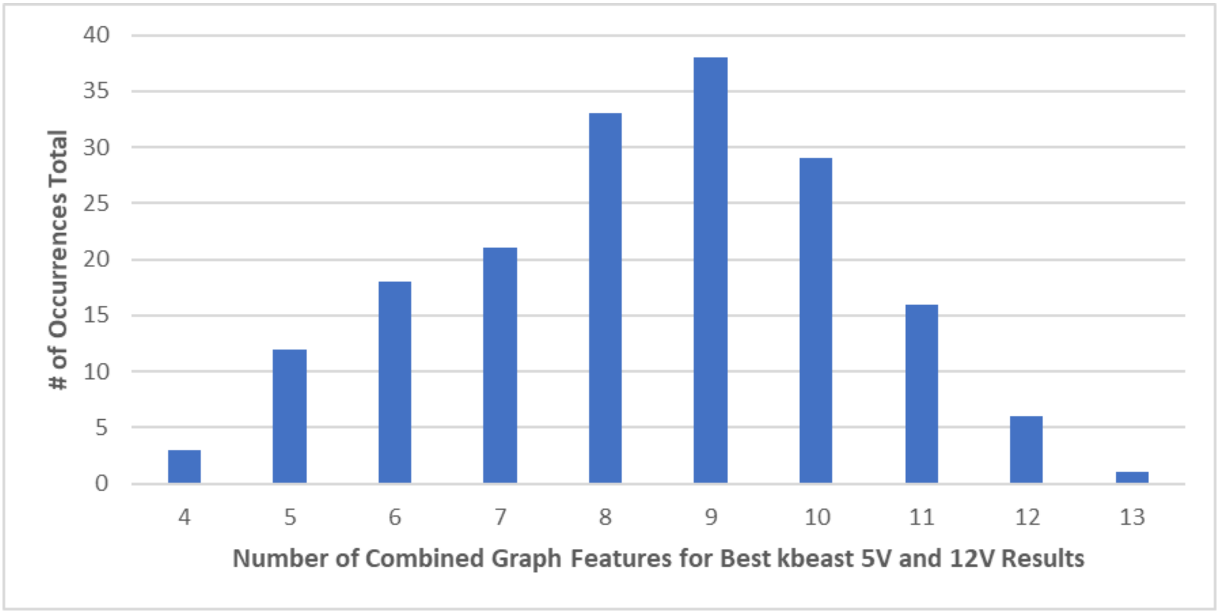


Figure 21: Combined Features Producing minDD = 0.083 for kBeast 5V and 12V

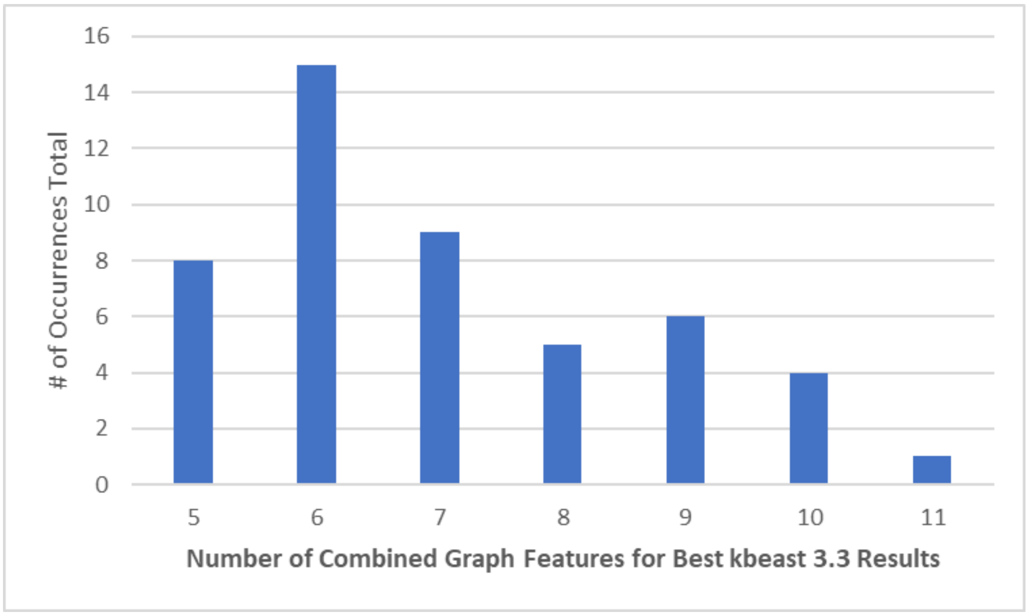


Figure 22: Combined Features Producing minDD = 0.19 for kBeast 3.3V

CHAPTER 5

CONCLUSION and FUTURE WORK

5.1 Conclusions and Key Contributions

Our case study presents results for using power-based data to detect rootkit execution in PC platforms. We extend prior work utilizing a nonlinear phase space algorithm for detection of anomalous behavior in dynamical chaotic systems while considering data from multiple voltage channels of an internal PC power supply. Our study considers 16 separate graph dissimilarity measures as feature input into a threshold/successive occurrence-based detection approach. We showed through our study of the Azazel, FUTo, and kBeast rootkits on three different operating systems that NLPSA could easily train a high percentage of random parameters and a number of feature combinations to produce perfect detection in five of the nine categories. We see some distinctions among rootkits with some parameters resulting in perfect detection in training and some variance between those where the minimal time to detect was also in view. The case study gives further validation to the efficacy of the NLPSA/side-channel approach as the data produced detection accuracies with 100% true positive and true negative rates and provides greater insight for future studies.

5.2 Future Work

In the future we will test a wider number of parameters, as well as implement a test runtime version of the nonlinear phase space algorithm that is designed to test the best parameter sets against new data. We will also analyze which features and feature combinations produce the best detection accuracy with the quickest time. We will compare our NLPSA approach to common machine learning algorithms while also considering a wider range of rootkits and general malware to see if power is indicative of system infection for all malware families.

Having shown that this approach works for PC based systems, we will also expand this work to consider other types of dynamical chaotic systems at risk of malware infection with both physical systems and virtual environments.

REFERENCES

[1] “The Latest 2022 Cyber Crime Statistics (updated November 2022) | AAG IT Support.” https://aag-it.com/the-latest-2022-cyber-crime-statistics/ (accessed Nov. 14, 2022).

[2] “Report: Government agencies are top target for rootkit attacks | VentureBeat.” https://venturebeat.com/security/report-government-agencies-are-top-target-for-rootkit-attacks/ (accessed Dec. 07, 2022).

[3] “Rootkits: evolution and detection methods.” https://www.ptsecurity.com/ww-en/analytics/rootkits-evolution-and-detection-methods/ (accessed Dec. 07, 2022).

[4] “How to detect & prevent rootkits.” https://www.kaspersky.com/resource-center/definitions/what-is-rootkit (accessed Nov. 12, 2022).

[5] “Rootkit | What is a Rootkit? | Malwarebytes.” https://www.malwarebytes.com/rootkit (accessed Nov. 12, 2022).

[6] “What Is a Rootkit & How to Prevent a Rootkit Infection in 2022?” https://www.safetydetectives.com/blog/what-is-a-rootkit/ (accessed Nov. 12, 2022).

[7] “Malware Analysis Explained | Steps & Examples | CrowdStrike.” https://www.crowdstrike.com/cybersecurity-101/malware/malware-analysis/ (accessed Nov. 12, 2022).

[8] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, “Zero-Day Malware Detection Based on Supervised Learning Algorithms of API Call Signatures,” in *Proceedings of the Ninth Australasian Data Mining Conference - Volume 121*, 2011, pp. 171–182.

[9] B. Liang, W. You, W. Shi, and Z. Liang, “Detecting Stealthy Malware with Inter-Structure and Imported Signatures,” in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, 2011, pp. 217–227. doi: 10.1145/1966913.1966941.

[10] J. Hernandez Jimenez, J. Nichols, K. Goseva-Popstojanova, S. Prowell, and R. Bridges, “Malware Detection on General-Purpose Computers Using Power Consumption Monitoring: A Proof of Concept and Case Study,” May 2017.

[11] D. Lobo, P. Watters, and X. Wu, “RBACS: Rootkit Behavioral Analysis and Classification System,” in *2010 Third International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 75–80. doi: 10.1109/WKDD.2010.23.

[12] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, “RHMD: Evasion-Resilient Hardware Malware Detectors,” in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 315–327.

[13] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato, “On the Detection of Kernel-Level Rootkits Using Hardware Performance Counters,” in *Proceedings of the*

2017 ACM on Asia Conference on Computer and Communications Security, 2017, pp. 483–493. doi: 10.1145/3052973.3052999.

[14] D.-P. Pham, D. Marion, M. Mastio, and A. Heuser, "Obfuscation Revealed: Leveraging Electromagnetic Signals for Obfuscated Malware Classification," in *Annual Computer Security Applications Conference*, 2021, pp. 706–719. doi: 10.1145/3485832.3485894.

[15] P. Luckett, J. T. McDonald, W. Glisson, R. Benton, J. Dawson, and B. A. Doyle, "Identifying Stealth Malware Using CPU Power Consumption and Learning Algorithms," Jul. 2018.

[16] L. M. Hively and J. T. McDonald, "Theorem-Based, Data-Driven, Cyber Event Detection," in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, 2013. doi: 10.1145/2459976.2460041.

[17] L. M. Hively, J. T. McDonald, N. Munro, and E. Cornelius, "Forewarning of epileptic events from scalp EEG," in *2013 Biomedical Sciences and Engineering Conference (BSEC)*, 2013, pp. 1–4. doi: 10.1109/BSEC.2013.6618498.

[18] L. M. Hively, V. A. Protopopescu, and N. B. Munro, "Enhancements in Epilepsy Forewarning via Phase-Space Dissimilarity," *Journal of Clinical Neurophysiology*, vol. 22, pp. 402–409, 2005.

[19] J. Dawson, J. T. McDonald, J. Shropshire, T. R. Andel, P. Luckett, and L. M. Hively, "Rootkit detection through phase-space analysis of power voltage measurements," in *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, 2017, pp. 19–27. doi: 10.1109/MALWARE.2017.8323953.