DATA LEAKAGE IN ISOLATED VIRTUALIZED ENTERPRISE COMPUTING SYSTEMS

Approved by:

Dr. Mitchell A. Thornton
Computer Science
Thesis Committee Chairperson

Dr. Eric C. Larson
Computer Science

Dr. Michael A. Taylor
External Advisor

DATA LEAKAGE IN ISOLATED VIRTUALIZED ENTERPRISE COMPUTING SYSTEMS

A Thesis Presented to the Graduate Faculty of the

Lyle School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Master of Science

with a

Major in Computer Science

by

Zechariah D.J. Wolf

B.S., Computer Science, Southern Methodist University

May 13, 2023

Wolf, Zechariah D.J. B.S., Computer Science, Southern Methodist University, 2022

Data Leakage in Isolated Virtualized Enterprise Computing Systems

Advisor: Dr. Mitchell A. Thornton

Master of Science conferred May 13, 2023

Thesis completed April 12, 2023

Virtualization and cloud computing have become critical parts of modern enterprise computing infrastructure. One of the benefits of using cloud infrastructure over in-house computing infrastructure is the offloading of security responsibilities. By hosting one's services on the cloud, the responsibility for the security of the infrastructure is transferred to a trusted third party. As such, security of customer data in cloud environments is of critical importance.

Side channels and covert channels have proven to be dangerous avenues for the leakage of sensitive information from computing systems. In this work, we propose and perform two experiments to investigate side and covert channel possibilities in virtual, enterprise environments. The first experiment is centered around the use of sensor data available via Intelligent Platform Management Interface, an open standard for out-of-band management often shipped with enterprise-level servers. We show how power-related sensors available with minimal user privilege over IPMI can be correlated with the levels of CPU stress of a virtual machine on a server.

This leads to our second experiment, in which we demonstrate a power analysis approach for establishing a covert channel for the exfiltration of data from a network-isolated virtual machine on a server rack. By applying the concept of power analysis more broadly to the power consumption of an entire server rack, rather than individual hardware components, we find that basic patterns in system load can be clearly identified using signal processing techniques, demonstrating the potential for establishing a covert channel.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

ACKNOWLEDGMENTS

I would like to thank my advisor, Mitch Thornton, for his constant support throughout my time in the master's program at SMU. I would also like to thank Eric Larson for his guidance on machine learning topics. Additionally, I want to acknowledge Mike Taylor for his mentorship and technical advice.

Finally, I want to thank my friends and family who were supportive of my academic endeavors.

Chapter 1

INTRODUCTION

1.1 What is a Side Channel?

In the most general sense, a side channel is any means by which information about the state of a computing system is unintentionally present. From a defensive perspective, side channels are important to identify and mitigate in order to reduce vulnerabilities in otherwise secure algorithms and systems. From the perspective of an adversary, side channels can be exploited to extract information. A "side channel attack" is defined as the exploitation of a particular side channel to extract private or sensitive information from a system for malicious purposes. The process of extracting information in support of a side channel attack is summarized in Figure 1.1, showing the flow of sensitive information to a listening attacker.

Figure 1.1: Side channel conceptual block diagram.

Side channel attacks are particularly dangerous because the vulnerability can exist, not only in an algorithm or system itself, but in the nature of how a system physically works. For example, an otherwise cryptographically secure encryption algorithm could be vulnerable to

a side channel attack because of information leaked from the hardware components on which the algorithm was performed [1].

However, side channels do not only represent a potential weakness. Recent work has shown that side channel data can be used defensively to identify anomalous behavior in a system that could be indicative of malware. As an example, Taylor et al. demonstrated a machine learning approach for early detection of a ransomware attack using the side channel of physical sensor data as an indicator of anomalous system behavior [2]. Trained models were able to able to identify subtle, randomly timed file encryption operations happening in a simulated ransomware attack with high accuracy. Thus, side channel investigations are important not only for identification and mitigation of vulnerabilities, but also for exploring use cases in defensive strategy.

1.2 Covert Channels

Related to the concept of a side channel is a covert channel. While a side channel can be used by an attacker to gather information from a system unbeknownst to the victim, a covert channel can be established when a source of information leakage is used deliberately to communicate information by encoding it over a channel that was not meant for communication [3]. Covert channels are an important area of research because they provide a means of communication outside the jurisdiction of organizational security policies. Thus, they can be used to transmit sensitive information to unauthorized parties. An effective covert channel, by making use of unconventional means of communication, can be used to leak information while evading detection. Figure 1.2 shows the flow of sensitive information from an attacker to a receiver in a covert channel attack.

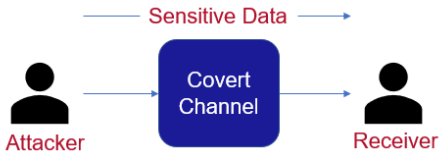

Figure 1.2: Covert channel conceptual block diagram.

Given the nature of computing systems, it is generally acknowledged that it is impossible to completely eliminate the possibility of a covert channel [3]. The possibilities are only limited by the creativity of an attacker, and the maximum rate of information transfer over the channel. Even so, it is still important to identify covert channels, and employ strategies to mitigate their effectiveness, by reducing their maximum data bandwidth, or improving detection capabilities. The utility of a covert channel for an attacker is proportional to the channel capacity, that is, the maximum rate of information transfer using the channel. The mere existence of a covert channel does not guarantee that it would be useful for a bad actor attempting to exfiltrate information from a system, as the channel capacity could be too low to facilitate communication in a reasonable amount of time.

1.3 Approach

In previous literature, the discussion of side channels is often centered around side channel attacks on cryptosystems. This has typically been accomplished by measuring some sort of physical signature of the device that is performing encryption (such as power consumption measurements). This approach is dependent on the fact that the power consumption of computational circuits is correlated with the data represented by the states of the individual transistors in the circuit [4].

1.3.1 Cloud Computing and Enterprise Environments

Today, cloud computing is ubiquitous and supports a wide variety of enterprise computing tasks. In a typical cloud computing environment, a server owned by a cloud computing service provider hosts virtual machines for various clients to access and use remotely. In such a virtualized computing environment, a piece of software known as the hypervisor manages the virtual machines, acting as a layer of abstraction between the host system and the hosted virtual machines. In a virtualized environment, a virtual operating system is running on virtual hardware, which is related to the physical hardware of the system by the hypervisor. In terms of side channel research, the hypervisor adds a layer of complication and isolation, and thus it is an important area of research to quantify the risks of side and covert channels in such environments. The impetus for our research can be summarized by the following questions: is it possible to correlate side channel data with the system activity of a virtual

machine, and if so, what are the applications of this side channel data for an attacker? In this work, we seek to demonstrate that even in virtualized enterprise computing systems, the potential for data leakage through side channel vulnerabilities exists and is not insignificant. We propose and perform two experiments to support our hypothesis in a network-isolated, virtual, enterprise computing system. The apparatus for these experiments is an enterprise-level server rack, which houses ten servers running a bare metal hypervisor operating system. This scenario is of particular importance since many users consider network-isolation, or so-called "air-gapped systems," to be an extremely effective method for the prevention of establishing a covert channel.

1.3.2 Sensor Data as a Side Channel

Modern computing systems are often equipped with a wide variety of onboard sensors, including both virtual and physical sensors, that report information about the state of the hardware. Typical information reported by physical sensors might include processing core temperatures, onboard fan speeds, voltage and current measures, or power consumption metrics. Likewise, virtual sensors include programmable event counters or other system diagnostics reported by host operating systems or firmware. Many of these sensors can be affected by the state of the computing system, which creates the potential for use as a side channel. As a basic example, a program performing a large amount of CPU-intensive calculations would generally cause the temperature of the CPU core to increase. In response to the elevated CPU core temperature, the system fans typically increase their speed to dissipate the excess heat. If the sensor data is recorded over time, a time-series analysis could reveal a source of information leakage regarding the state of stress on the system.

Our first side channel investigation is focused on gathering sensor data from an enterprise computing system, and observing the changes over time in response to different system activities.

1.3.3 Power Analysis of Side Channels

Power consumption of hardware components has proven to be a viable side channel in computing systems. "Power analysis" is the term describing the process of extracting and exploiting information from such an observed power consumption time-series when used as

a side channel. In a power analysis attack, power metrics from a computing system are observed and analyzed to extract important information about the behavior of the system that are subsequently used as the basis of an attack. Research spanning decades has shown power analysis to be an effective means of breaking encryption by monitoring the power consumed over time by the machine hardware. Early examples of power analysis showed the breaking of the Data Encryption Standard (DES) algorithm [5], spawning a multitude of investigations into side channel attacks on cryptosystems. Even with two decades of technological advancement since the earliest examples, researchers are still finding new methods and side channels for power analysis attacks.

In our second experiment, we demonstrate a power analysis approach for analyzing a power side channel in an enterprise-level server rack. By measuring the electrical power signal of the server rack, we show the existence of a side channel leaking basic information about the state of a virtual machine. As a proof-of-concept application of this side channel data, we demonstrate the establishment of a covert channel to exfiltrate information from a virtual machine. In particular, we show how correlations between computational load activity on the virtual machine and the system's electrical power signal is a viable means of encoding sensitive information for the purpose of exfiltration by a malicious insider.

Chapter 2
BACKGROUND

2.1 Hypervisors and Virtualization

A hypervisor is an important piece of software in the context of virtualization. A hypervisor serves as an interface between a computer's physical hardware, and the virtual hardware supporting the virtual operating system hosted by the computer. In general, there are two types of hypervisors. A "bare metal," or type 1, hypervisor runs directly on a computer's physical hardware. Type 1 hypervisors are particularly useful in cloud computing scenarios, where an entire server might be dedicated to hosting virtual machines. Alternatively, a type 2 or "hosted hypervisor" is one that functions in a software layer that is higher than the native operating system directly executing on the physical server hardware. An example of a hosted hypervisor is VirtualBox[1], which can be installed on a variety of operating systems, and allows a user to run virtual machines on top of their native operating system [6].

2.2 Intelligent Platform Management Interface

Intelligent Platform Management Interface (IPMI) is an open standard designed by Intel which provides out-of-band management capabilities. Out-of-band management is achieved typically through a Baseboard Management Controller (BMC) on the server, which is a specialized chip powered separately from the host operating system. If the BMC is connected to a network, the system can be managed remotely through the BMC that bypasses the host operating system, providing flexibility for system administrators. A system with IPMI implemented on a BMC can be managed remotely even while the host system is powered off, which makes IPMI a useful tool for system administration [7].

[1] https://www.virtualbox.org/

2.2.1 IPMI Sensor Data

An important feature of IPMI in terms of side channel investigation is the ability to read sensor data. The IPMI standard released by Intel specifies that sensor data should be stored in a Sensor Data Record Repository (SDR Repository). The SDR repository is continuously updated with readings from the onboard sensors, storing them for later access.

IPMI users can access the information in the SDR repository using the "Get SDR" command, which Intel specifies should be implemented to allow reading of sensor data [8].

2.2.2 IPMI Privileges

According to the IPMI standard, there are four privilege levels (in order of increasing privilege): callback, user, operator, administrator. The privilege level determines which commands a user has access to, as specified in the standard. Importantly, the "Get SDR" command is available at the user level and above, which means even with a very basic set of privileges (the user level allows most read commands but restricts any commands that modify the state of the system), it is possible to access the sensor information in the SDR.

Users can access IPMI over a variety of interfaces, such as serial port access or LAN access. LAN access allows users to remotely login and execute IPMI commands, provided the IPMI device is reachable over the LAN and the LAN interface is enabled. For added security, IPMI systems can support encryption over the LAN interface, using RMCP+ (Remote Management Control Protocol, defined in the IPMI specification [8]). For experimentation, we make use of the LAN interface to conveniently execute IPMI commands from a remote device on the same isolated network.

2.2.3 Using IPMI Commands

The IPMI standard specified by Intel is implemented by a variety of hardware vendors in modern server computers. Hardware vendors can provide their own interfaces (such as HP Integrated Lights Out) to the underlying IPMI system. Alternatively, IPMI can be accessed through raw commands sent over a specific interface. In this work, we make use of an open source tool known as `ipmitool` [9] to send commands to the BMC in our experimental apparatus.

2.3 IPMI Vulnerabilities

IPMI has been used widely in enterprise environments for out-of-band management. IPMI is a convenient way to manage servers in an enterprise context without having to interact directly with the server's operating system. However, a variety of vulnerabilities in IPMI have been documented over the years.

2.3.1 Specification Vulnerabilities

It has been shown that the IPMI standard is vulnerable to a variety of attacks that might give an attacker control over the BMC. In a report on penetration testing of IPMI systems [10], it was found that the clear text authentication option allows logging in with only the username of an account. From this point, it would be trivial for an attacker to create their own privileged account to establish more permanent authenticated access. It was also discovered in this report that the server will send the stored password hash to the client upon login attempt, regardless of whether or not the login attempt was successful. This opens the door for offline password cracking attacks that could allow an attacker entry to privileged IPMI accounts.

Given these vulnerabilities, it is good practice to keep IPMI devices on an isolated network, with no access to the public Internet [11, 12], which should reduce the risk of exploitation by outside attackers. However, this feature must be configured by system administrators manually. Accordingly, improperly configured IPMI systems are a potential risk. Researchers have found that there are an alarming amount of IPMI devices reachable on the public internet. In 2013, Bonkoski et al. [11] found around 100,000 such devices. In 2016, Gasser et al. [12] used a new method for scanning for IPMI-enabled devices, and found around 225,000 such devices on the public Internet. It is reasonable to estimate that the number of IPMI systems reachable on the public Internet has decreased since 2016, but the issue still remains.

2.3.2 Implementation-specific Vulnerabilities

Aside from vulnerabilities in the IPMI standard itself, researchers have identified many security issues existing in the various implementations of the IPMI standard shipped by certain vendors with their servers. Many of these issues result from the default configurations of the system that system administrators neglect to change. For instance, multiple vendors

provide a web interface for interacting with the BMC. These web consoles are often shipped with insecure default passwords, depending on the manufacturer. An attacker might be able to gain access to IPMI simply by guessing common default login credentials in the web console of a public Internet-facing server with IPMI enabled [12]. Clearly, it is good practice for system administrators to change default passwords associated with vendor-provided IPMI web consoles.

Even if the default login credentials have been changed, the system might still have a default SSL certificate associated with its web console. For example, Felix Emmert found that the default SSL certificate for Dell iDRAC 7 (Dell's OEM implementation of IPMI), is contained in the iDRAC 7 firmware package [13]. An attacker could easily use this default SSL certificate to intercept and decrypt traffic to the BMC.

2.3.3 IPMI as an Attack Vector

Given the significant issues with the IPMI protocol, and the major privileges afforded to administrator accounts, IPMI-enabled BMC's offer serious value to an attacker. Previous research has explored the potential exploits and attacks that could be carried out with privileged access to the BMC. One unexplored topic, however, is the potential for monitoring sensor data via IPMI. As mentioned previously, even unprivileged, user-level accounts can request sensor data over the IPMI protocol. In this work, we perform a side channel investigation of the sensor data that is available to unprivileged users over IPMI.

2.4 Power Analysis Concepts

As introduced previously, power analysis is one approach for extracting information from power side channels that has proven effective in recent literature.

2.4.1 Power Analysis

Power analysis research is not new. In 1999, Kocher et al. demonstrated a power analysis side channel attack on the DES encryption algorithm. By collecting traces of electrical current measurements over time, they were able to extract an encrpytion key using two different power analysis approaches [5]. Despite years of advancement in technology and heightened awareness of security, power analysis of side channels remains a difficult chal-

lenge to overcome, and researchers are still finding ways to exploit side channels to break cryptosystems. For instance [14] recently demonstrated a series of power analysis attacks for extracting encryption keys through power side channels identified in Intel CPUs.

However, looking beyond the scope of cryptosystems, power analysis could have applications in side channels more broadly. Previous research has investigated power analysis of specific hardware components used in encryption algorithms in order to leak specific data being used in the algorithm. We decided to investigate how this kind of approach could reveal a side channel at the scale of an entire computing system, rather than individual hardware components. Specifically, it would be of interest to identify a potential side channel in an enterprise computing system, where tasks are typically run on virtual machines, with the hypervisor acting as a layer of abstraction between the virtual machine and the physical hardware.

In this work, we focus on power analysis of a side channel identified in an enterprise computing system. However, the data capture we employ uses a non-traditional methodology that leverages the electrical power line fluctuations (EPLF) that is generated by a computing system while in operation. This EPLF has been used by a number of researchers for identifying electrical devices and appliances [15, 16], for identifying changes in electrical device signature that can be repurposed for interaction techniques [17, 18], and has been shown to leak information regarding television programming [19]. Here, we use shifts in the EPLF system fingerprint to indicate the internal state of a computing system running a virtual machine, showing that it can be used as a covert side channel.

It is also important to note that the EPLF generated by a computing system is related to the power usage of the system, but can also be influenced by other systems on the same circuit or other devices that cause fluctuations at certain frequencies on the voltage spectrum. The EPLF generated by the computing system is most easily seen by performing a kHz range frequency analysis on the circuit near where the system is obtaining power. Because this EPLF is generated along the entire circuit, this signature can be obtained by sampling from any nearby power outlet. Thus, direct access to the computing system is not required to access this side channel

2.5 Signal Processing Concepts

In our second experiment, we collect traces of the electrical power signal from a server rack under different system conditions. The captured signals represent voltages in the time domain, i.e. each measurement represents the voltage observed by the oscilloscope at a certain point in time. In this raw form, this data is not helpful for fingerprinting the state of the system. This section details the signal processing concepts crucial for understanding our approach to extracting information from the power signal.

2.5.1 Fourier Transform

The Fourier transform is a procedure for extracting the component frequencies that make up a signal. A Fourier transform takes a signal, showing information with respect to time (usually referred to as the time domain), and represents it in the frequency domain. In the frequency domain, the signal is represented by the amplitude and phase of the sinusoids that, when summed, comprise the signal in the time domain. Thus, the Fourier transform is a useful tool for better understanding the nature of a signal as a composition of various sinusoids of different amplitudes and phases, which is otherwise not easily interpretable in the time domain [20]. In this work, we use Fourier transforms in a variety of applications for the spectral analysis of power signals.

2.5.2 Power Spectral Density Estimation

Power spectral density (PSD) estimation is a technique used for spectral analysis of signals in the frequency domain. The result of a PSD estimation is a representation of the power levels of each of the frequency components of a signal.

One method we use for PSD estimation in this work is Welch's method [21]. In short, Welch's method works by dividing time domain signal into overlapping segments of a specified length. A window function is applied to each segment, and then the Fourier transform is applied to each segment. Finally, the results from each segment are averaged to produce a PSD estimate, communicating the power of the signal with respect to its frequency components [22]. We use Welch's method for PSD estimation in order to characterize the spectral content of electrical power signals.

2.5.3 Short-Time Fourier Transform

When a signal is expected to change in the frequency domain over time, a short-time Fourier transform (STFT) can be used to determine the nature of the signal over discrete time intervals. This is accomplished by dividing the signal into windowed segments (usually overlapping) and taking the Fourier transform of each segment of the signal. The result is a complex-valued matrix such that each column represents the Fourier transform of a specific time segment [23].

2.5.3.1 Spectrograms

A useful way to visualize the result of a short-time Fourier transform is the spectrogram. A spectrogram shows the change in the signal over time as a color heatmap, where the color intensity represents the magnitude at a specific frequency (vertical axis) at a certain time (horizontal axis). We use the STFT and spectrogram visualization as a way to show the change in spectral content over time of electrical power signals.

2.5.4 Signal to Noise Ratio and Additive White Gaussian Noise

In any real world scenario where a signal is being monitored, there will be some amount of background noise that is captured along with the signal. Depending on the intensity of the background noise, the information conveyed by the signal can be muddled and harder to analyze. The signal to noise ratio (SNR) is a measurement that quantifies the power of this background noise relative to the power of the signal. This is computed as the ratio of the signal's power, to the power of the background noise. Because SNR is a ratio of power quantities, it is usually expressed on a logarithmic, decibel (dB) scale [24]. For a signal measured in volts, the power can be expressed as the square of the root mean square (RMS) voltage amplitude [25]. This is derived by Ohm's law 2.1a and Joule's first law 2.1b as follows (assuming that resistance R is constant):

$$I = \frac{V}{R} \tag{2.1a}$$

$$P = I^2 R \tag{2.1b}$$

$$P = \left(\frac{V}{R}\right)^2 R = \frac{V^2}{R} \tag{2.1c}$$

For a discrete signal $x(t)$ with n measurements in the time domain, the RMS amplitude can be calculated as:

$$V_{rms} = \sqrt{\frac{1}{n}\sum_{t=0}^{n} x(t)^2} \tag{2.2}$$

Finally, the signal to noise ratio in dB, in terms of the RMS amplitude of a signal measured in volts, can be calculated as:

$$SNR = 20 \cdot log_{10}\left(\frac{V_{rms,signal}}{V_{rms,noise}}\right) \tag{2.3}$$

SNR is relevant for our research as we simulate background noise conditions using an additive white Gaussian noise assumption.

2.5.5 Additive White Gaussian Noise

Additive white Gaussian noise (AWGN) can be used to simulate the effects of noise by adding random numbers to a signal in the time domain. The random numbers are drawn from a Gaussian distribution with a given mean and standard deviation. To add noise to a signal, each measurement from a time domain signal $x(t)$ is added with a random number drawn from a Gaussian distribution, to produce a noisy signal. The amplitude of the noise component, as determined by RMS amplitude, is approximately equal to the standard deviation of the Gaussian distribution, if the mean μ of the distribution is equal to 0. For this reason, it is simple to add Gaussian noise to a signal in order to calculate a specific SNR [26]. Given a signal with a known RMS amplitude, and a target SNR, the standard deviation (expressed as $V_{rms,noise}$) needed for the Gaussian distribution can be determined through algebraic manipulation of equation 2.3, as follows:

$$SNR = 20 \cdot log_{10}\left(\frac{V_{rms,signal}}{V_{rms,noise}}\right) \tag{2.4a}$$

$$V_{rms,noise} = V_{rms,signal} \cdot 10^{(-SNR/20)} \tag{2.4b}$$

We use AWGN in this work for simulating noisy conditions to assess the noise resilience of a classification model.

2.6 Machine Learning and Classification Concepts

In our second experiment, we evaluate the feasibility of a covert channel via the electrical power signal of a server rack. In such a scenario, an attacker would need a means of recognizing data symbols based on the fingerprint of the power signal. This section provides background on the machine learning and classification concepts used in our approach for classification of data symbols.

2.6.1 Classification

In this context, classification refers to the process of labeling data points (samples, observations) as members of a certain class. A classification model is trained to predict the class of a sample based on its characteristics (features). In our application, a segment of time from the power signal defines a sample, and its class is the binary data symbol (0 or 1) which is being encoded. In this context, a 0 is the negative class, while a 1 is the positive class [27].

2.6.2 Classifier Scoring

During the process of binary prediction, a classification model produces a score (based on some algorithm/process inherent to that specific kind of model), which is then filtered through a threshold to decide on a final prediction for the sample (0 or 1). For example, a model with a threshold of 0.5, would predict all samples with a score less than 0.5 as members of the negative class (0), and all samples with a score greater than 0.5 as members of the positive class (1) [27].

2.6.3 Receiver Operating Characteristic

One way to measure the performance of a binary classification model is to use the Receiver Operating Characteristic (ROC). ROC analysis is based on the true positive rate and false positive rate of the model at varying classification thresholds. The true positive rate (TPR) is the proportion of samples that are correctly identified as the positive class, while the false positive rate (FPR) is the proportion of samples which are incorrectly labeled as the positive class.

An ROC curve is created by recording the TPR and FPR for the model as the classification threshold is varied throughout its dynamic range. A plot showing the TPR versus the FPR is known as the ROC curve. A metric that summarizes this curve as a single value is the area under the ROC curve (AUC). The AUC, on a scale of 0 to 1, rates the model's ability to distinguish between negative and positive examples, such that 0.5 is the expected performance of a random chance classifier, and 1.0 means perfect classification [27]. Figure 2.1 showcases a few example ROC curves with different AUC scores.

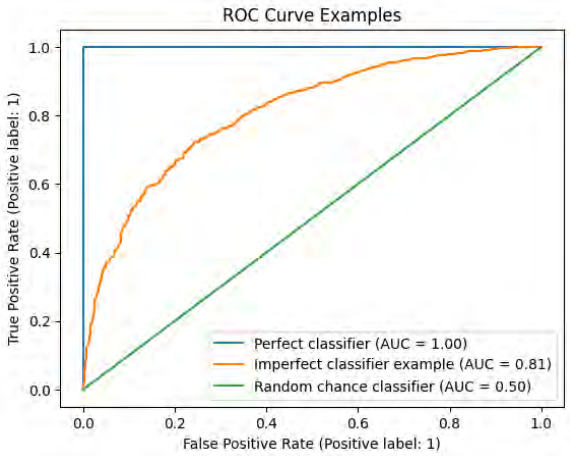

Figure 2.1: Example ROC curves for sample classification models.

2.6.4 Classifier Accuracy

Another metric we use for evaluating classification performance is accuracy. Accuracy is a straightforward measurement that reports the proportion of the classifier's predictions that were correct. This value falls in the range of 0 to 1, and is typically expressed as a percentage [27]. A classifier with 90% accuracy, for example, is able to correctly identify 90% of samples.

In the binary case, accuracy is a relevant performance measure when the number of positive class samples is about equal to the number of negative class examples, i.e. the classes are balanced, and when the classes are of equal importance to identify. These two conditions hold true in our covert channel scenario, so we choose to use accuracy as one of our performance measures for evaluation.

$$accuracy = \frac{Number\ of\ correct\ predictions}{Number\ of\ total\ predictions} \tag{2.5}$$

2.6.5 Extrapolating Accuracy from the ROC Curve

Given that an ROC curve represents the TPR and FPR at varying classification thresholds, the ROC curve can be used to find a specific classification threshold for optimal performance. This threshold is the threshold of the equal error rate (EER). The EER is the point at which the false positive rate is equal to the false negative rate, i.e. the classifier is equally as likely to make a mistake about either class. Visually, the point of equal error rate is the point on the ROC curve that intersects with the line $y = 1 - x$ [28]. Figure 2.2 shows an example point of EER on an ROC curve. We use the classification threshold at the equal error rate in order to make predictions from raw classifier scores, and calculate the accuracy.

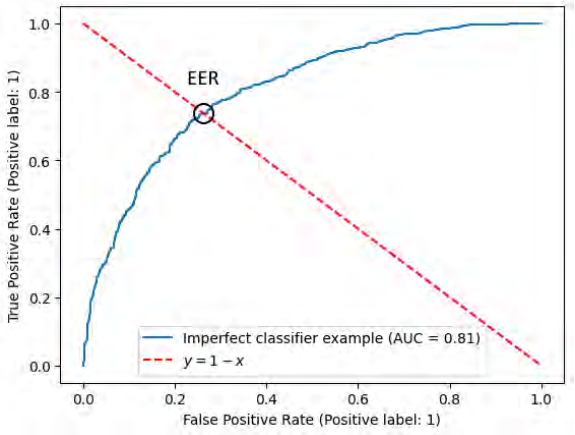

Figure 2.2: An example ROC curve intersecting with the line $y = 1 - x$. The point of intersection is the EER.

Chapter 3
IPMI SIDE CHANNEL EXPERIMENT

The first step of our investigation into side and covert channels in an enterprise computing environment leverages the management capabilities of IPMI for the gathering of sensor data. In this Chapter is detailed the experiments performed to test the potential for data leakage over the side channel of sensor data collected via IPMI.

3.1 Experimental Design

This experiment (and the experiment in the following Chapter) was carried out using a server rack which houses ten servers running a bare metal hypervisor as the primary operating system. The servers are equipped with a BMC that implements the IPMI 2.0 standard as provided by Intel. The original equipment manufacturer (OEM) of this hardware provides a customized implementation of the standard, which is available through a web interface or remote command line session. However, it is not necessary to use the OEM's interface directly, as the BMC responds to raw IPMI commands sent over an appropriate interface (LAN, serial, etc.). This is the method used in this experiment for communication with the BMC. The goal of this side channel investigation is to see if there are any correlations with the activity of a virtual machine, and the readings from the server's onboard sensors.

3.1.1 Data Collection

For this experiment, we set up a Linux virtual machine on one of the servers in the rack. All of the servers on the rack are connected to an isolated LAN, which is only used by the server rack. A laptop is connected to this LAN in order to interface with the server's host operating system, as well as communicate with the BMC. While connected to the LAN, the laptop can control the virtual machine, and issue IPMI commands to the BMC. There are a variety of onboard sensors on each server that report to the BMC, including power consumption in watts, fan speeds in RPM, CPU temperature, and various hardware

Figure 3.1: A picture of the server rack used for the experiments in this work.

component voltages.

As mentioned previously, IPMI specifies a command for querying the current sensor values from the SDR repository. Using a script, this process can be automated to query the sensors, and record the sensor values at given time intervals. In order to interface with the IPMI system, we use an open source tool called `ipmitool` [9], which can login and send raw IPMI commands to the BMC. In our setup, the most convenient interface for this is using a laptop connected to the rack's LAN. The laptop sends commands over the IPMI LAN interface, and receives back the sensor data from the server. Figure 3.2 shows the details of the experimental setup.

The data collection script is written in Python, and uses a package `pexpect` to establish an IPMI session, send `ipmitool` commands to sample the sensor data, and store the output in memory. `ipmitool` supports a continuous connection over LAN with a shell interface, using the `shell` subcommand. Once the shell session has been opened (by authenticating with username and password), the script repeatedly issues the subcommand `sdr elist full` to poll the sensors. The sensors are polled as fast as is possible over the network, which is about 7 readings per second in our particular setup. When a specified time limit expires, the connection is closed and the sensor data is written to a file. The results are recorded as a

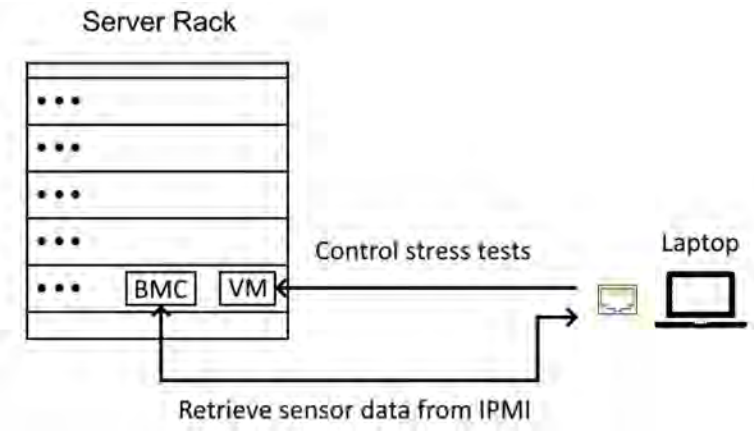

Figure 3.2: Block diagram showing the experimental apparatus.

comma-separated values (CSV) file, where each row contains a timestamp, and the reported values for each sensor at that timestamp. The script can be summarized by the following pseudocode:

```
readings = {} #stores readings in memory before writing to file
child_thread = pexpect.spawn("ipmitool shell")
repeat until time elapsed:
    reading = child_thread.send_command("sdr elist full")
    readings[current_timestamp] = reading
readings.write_csv("output.csv")
```

3.1.2 Procedure

In order to observe the response of the sensors to system activity, we make use of an open-source Linux package called `stress-ng` [29]. The `stress-ng` tool is a command line interface that allows the user precise control over a machine's hardware, enabling creation of highly customizable system loads. `stress-ng` is an extensive package with a large variety of options for generating system stress, but in this experiment, we use `stress-ng` specifically for generating CPU utilization. To accomplish this, a shell script is used to initiate a `stress-ng` load for a certain time interval, and then sleep for an interval. By alternating the system

between states of high stress and low stress, we should be able to observe any effect on the sensors, and evaluate this data as a possible side channel for data leakage regarding the state of the virtual machine. The shell script used in the experiment is summarized by the following pseudocode:

```
for ((i=0; i < numCycles; i++)) do
    sleep 5 #idle for 5 seconds
    stress-ng --cpu 16 -t 5s
done
```

3.2 Experimental Results

To get a first impression of the sensor response to system stress, we collect the sensor data for 70 seconds, while running the stress test loop script. The script runs with a period of five seconds idle, and five seconds of stress testing.

3.2.1 Fan Speeds

The first set of sensors to consider is the speeds of the system fans. The server has several onboard fans, such as CPU fans and exhaust fans. When polling the sensors via IPMI, there are five fans whose speeds are available in the SDR repository, labelled simply as "fan2" through "fan6." As an initial test, the stress test looping script is run with a period of five seconds. The system fan response can be visualized by plotting the reported speed in RPM against the timestamp for the measurements. The result is shown in Figure 3.3.

The dashed red lines in Figure 3.3 show either the start or completion of a stress test. Fan 2 does not seem to respond at all to the stress tests, and stays static throughout the data collection process. Fan 3 has some level of response, but it is not always in time with the stress test and idle periods. The rest of the fans have a strong, fairly immediate response to the initiation or termination of a stress test.

3.2.2 CPU Sensors

The next category of sensors to look at is CPU-related sensors. The server used in this experiment has two processors, and each reports its temperature in degrees C, and VCore (the voltage supplied to the CPU) in volts. Since the stress testing process generates 100%

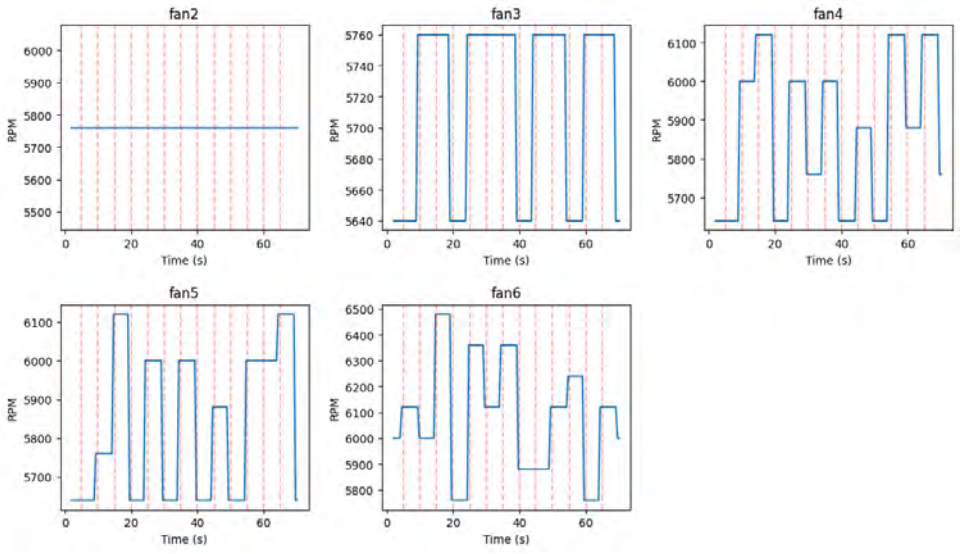

Figure 3.3: Fan speed (RPM) vs. time for the five fan speed sensors reported on by IPMI.

CPU utilization on the virtual CPU, we can expect that these CPU-related sensors should show a strong response. Figure 3.4 shows the CPU-related sensor measurements with respect to time.

The CPU temperatures show a strong correlation with the stress test and idle periods, although the response is somewhat delayed, as compared to the fan speeds, such that the temperatures rise and fall periodically, but slightly out of phase with the stress tests. The VCore sensors seem to correlate better with the stress tests for most of the data collection period, with minimal delay.

3.2.3 Power Related Sensors

Besides fan speeds and CPU temperatures, the server is also equipped with several sensors that report on power related metrics. This includes power consumption, power supply currents, and power supply voltages. The server uses two power supply units, and both report their voltage and current measurements to the BMC. Figure 3.5 shows the power related sensor data with respect to time.

The first thing to notice is that the current measurements from the second power supply appear to be in error, as a constant current measurement of 0 amps would not be possible

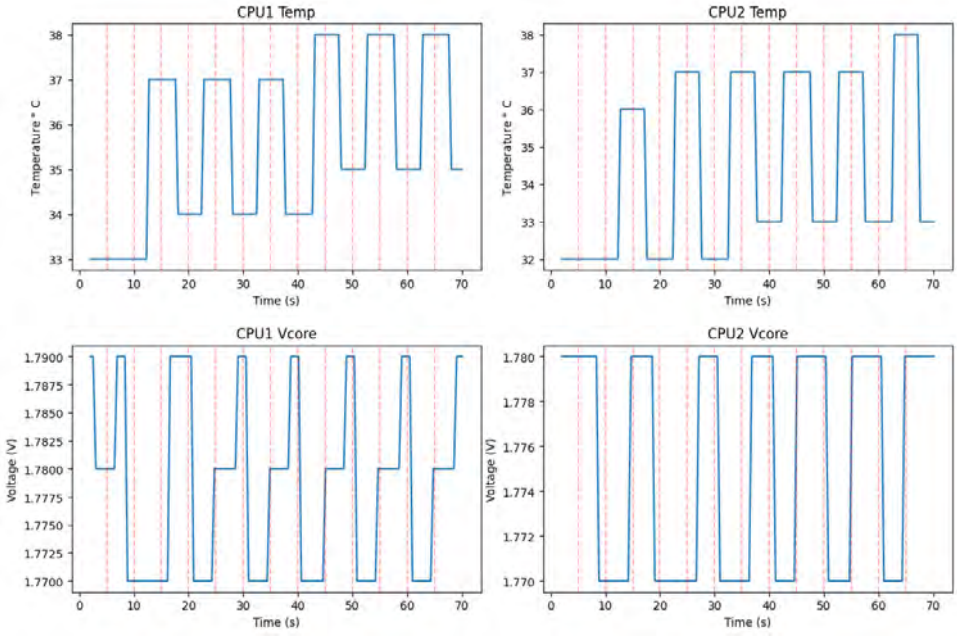

Figure 3.4: CPU-related sensor values vs. time during stress testing.

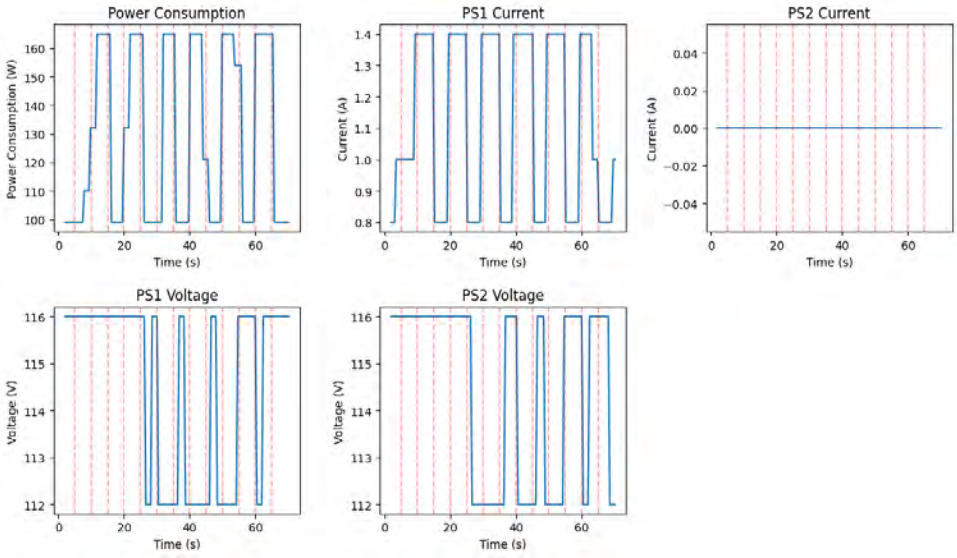

Figure 3.5: Power-related sensor values vs. time during stress testing.

given the reported changes in the power supply's voltage throughout the time series. This may be due to an error in the data collection process, or an issue with the sensor's reported

value to the BMC. Besides this peculiarity, however, the other sensors definitely show a correlation with the stress testing. In particular, the power consumption and power supply current respond promptly to the beginning and end of the stress tests.

3.2.4 Limits of the Sensor Response Time

With a period of five seconds for stress testing, and five seconds idling, it is easy to see the sensors responding to the changes in CPU utilization. The power supply current and power consumption metrics were particularly sensitive to the changes in system state, showing very immediate responses to the stress tests. We perform further testing to see just how quickly these two sensors can respond to system stress by decreasing the period of the stress test and idle periods on the virtual machine. Starting with a period of two seconds, we perform the same data collection procedure, and record the sensor data for 60 seconds while the virtual machine continually alternates between stress testing and idling at a period of two seconds. Figure 3.6 shows the power consumption and power supply current measurements from this collection period.

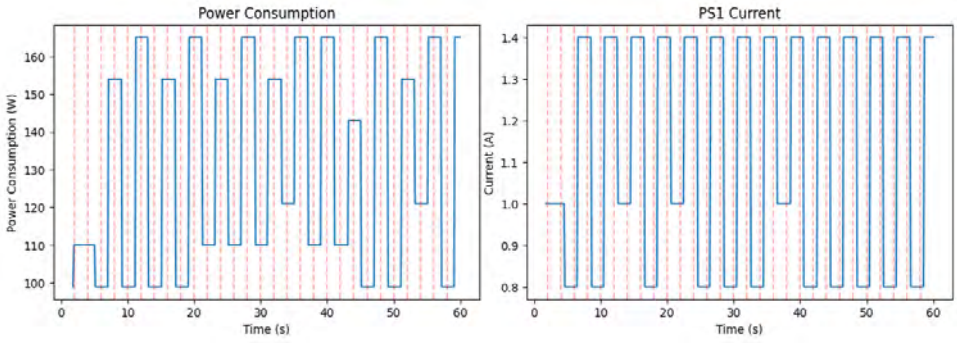

Figure 3.6: Power-related sensor values vs. time during stress testing with a 2 second period.

Once again, these two sensors track very closely with the stress testing and idle periods on the virtual machine. As a final test, we decrease the amount of time per stress test down to just one second and repeat the same test. Figure 3.7 shows the results from a thirty

second data collection period.

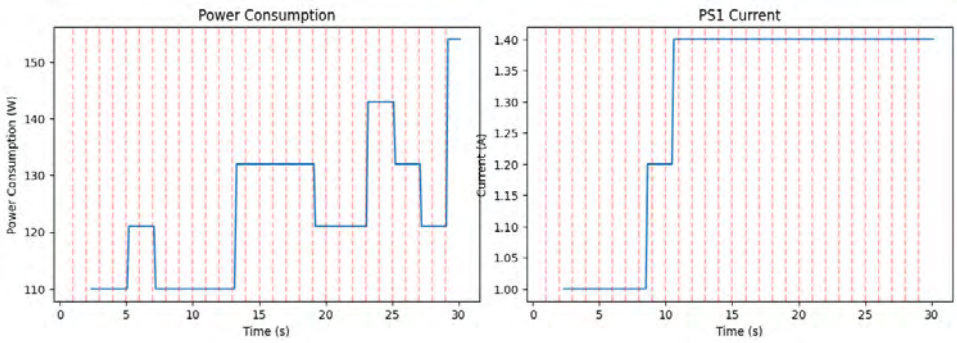

Figure 3.7: Power-related sensor values vs. time during stress testing with a 1 second period.

With the virtual machine spending only one second stress testing or idling, the sensor response is not very clear. The two power sensors seem to trend upward in general, but there is a lack of precise correlation with the system load.

3.2.5 Analysis

Our experimental results show that the sensors available via IPMI, particularly the power consumption and power supply current metrics, correlate strongly with the system activity of a virtual machine, showing whether the virtual machine is in a state of stress, or is idle. The sensors show near immediate response when the system alternates between these two states periodically, with a period of alternation down to two seconds. At a period of one second, the power sensors do not show a strong correlation to the one-second-long periods of stress and idle conditions. The reason for this result is likely due to the low sample rates that limit the frequency of updating values in the SDR repository. Although the data collection script is able to sample the sensors at a rate of about 7 Hz, the values in the SDR repository are likely not updated that quickly. We were unable to determine how often the SDR repository is updated with new sensor readings, as this parameter is not specified in the IPMI standard. Based on the testing shown in Figure 3.6, it seems reasonable to estimate

that the SDR repository is updated at least once every two seconds. It may also be the case that the true power consumption and power supply current readings do not change fast enough to show the system changing between states every second, as were the attempts in Figure 3.7.

3.2.6 Covert Channel Possibilities

Up to this point in the research investigation, we have shown that gathering sensor data via IPMI can leak very basic information about the state of a virtual machine on the server, i.e. whether or not the virtual machine is in a state of high stress or low stress. As a side channel, this data may not be as exploitable as other methods that are described in a latter section of this thesis. However, the IPMI-based results have demonstrated the feasibility of establishing a covert channel in this scenario. Given that the power consumption and power supply current track closely with periods of high and low stress on the virtual machine, it is feasible for an attacker to instantiate a covert channel by adjusting the power consumption of the virtual machine in order to encode sensitive information in the channel's time domain, e.g. low power consumption represents a binary 0 and high power consumption represents a binary 1.

Based on these results, communication over the channel is possible by monitoring the power consumption and power supply current via information contained within IPMI packets allowing one to interpret binary data encoded into the power consumption levels of the server. However, there are two main reasons why this would not be attractive to an attacker for covert communication. First, the IPMI-based approach requires a valid account – albeit an account with minimal privilege level – registered with the IPMI subsystem in order to poll the sensors. A dedicated inside attacker looking to exfiltrate sensitive information might exploit vulnerabilities demonstrated in [11–13] in order to gain access to the BMC, but it would be an obstacle nonetheless. Secondly, and perhaps more importantly, our approach thus far requires sending hundreds of IPMI commands over the management network, in order to monitor the sensors. This quantity of IPMI traffic is atypical and thus is not a very covert approach, since such a volume of IPMI traffic could easily be detected by system administrators monitoring traffic on the management network.

Nevertheless, we have shown that the power consumption of the server can be indirectly controlled by running stress tests on a virtual machine. If an inside attacker were able to monitor the power consumption on the server in a more discreet manner, then a covert channel would be more difficult to detect.

Chapter 4
POWER ANALYSIS EXPERIMENT

4.1 Experimental Design

In order to evaluate the potential of a power analysis side channel in an enterprise computing system with virtualization, we propose an experimental approach to collect and analyze power consumption metrics directly from a server rack without requiring IPMI access. An important factor in comparing the IPMI-based approach for monitoring power consumption levels versus the direct approach described here is that the IPMI-based method does not require physical access to the server rack power lines. In fact, the IPMI-based method could be accomplished in a remote manner if BMC access is available via a network connection.

We reported on our initial findings in a paper presented at the 9^{th} International Conference on Information Systems and Privacy [30] in February 2023. Portions of this work have been taken or adapted from that publication although the results in this Chapter include significant extensions to those in the publication.

4.1.1 Server Rack

For this experiment, we use the same server rack setup as is used for the IPMI experiments (see Figure 3.1). The power supplies for each server, as well as the network switches and other rack components, use the rack's power distribution unit (PDU) that is connected to a 2U 120 VAC power supply that services the entire rack. A laptop is used to connect to and control the virtual machine on the server over a LAN connection. The operating system supporting the virtual machine is Kubuntu 22, a distribution of Ubuntu using the KDE Plasma desktop environment.

In order to modulate the power consumption of the server, we once again use `stress-ng`. The goal with using `stress-ng` is to generate a load pattern that, in turn, causes a corresponding variation in power consumption that is recognizable and distinguishable, even

in the presence of background noise. To accomplish CPU load variations, a shell script is used to initiate a `stress-ng` load for a certain time interval, and then to sleep for a subsequent interval. The shell script thus serves to modulate the power consumption profile that varies in proportion to the CPU load changes. Therefore, the shell script modulator creates states of high and low power consumption of varying respective duration that are observable as a power analysis side channel. A particular binary string corresponds to the different duration times of the high and low load states within the script that can be observed by monitoring the 120 VAC power line serving as a side channel for exfiltrating the binary stream of information. The shell script used in the experiment is summarized by the following pseudocode:

```
for ((i=0; i < numCycles; i++)) do
    sleep 1 #idle for 1 second
    stress-ng --cpu 16 -t 1s
done
```

For side and covert channel investigations, we use a procedure that is similar to the IPMI experiment procedure, using `stress-ng` to create stress on the virtual machine with a regular period that represents an alternating bit string comprised of 0 and 1 values.

4.1.2 Power Analysis Approach

In order to record measurements of the power consumption, the rack is plugged into a surge protector, and our customized power line interface (PLI) module is plugged into a nearby power outlet on the same circuit breaker. This PLI module acts as a high pass filter, and is a reconstruction of a circuit originally used in an experiment to classify the use of home appliances [16]. This PLI module allows us to collect high frequency EPLF in the range of approximately 0 to 100 kHz. The PLI module connects the power line from the surge protector to the oscilloscope. The side channel monitoring device is thus the combination of the PLI module with the oscilloscope. The oscilloscope used in this experiment is from the Picoscope 4000a series. The oscilloscope is connected via USB to a laptop which has the required software and drivers to monitor and collect data from the device. Picoscope provides monitoring software that displays real-time measurements from the connected oscilloscope.

We also make use of a Python wrapper library for the Picoscope software development kit that allows us to collect data from the oscilloscope with a custom Python script. This script is based on examples provided by the maintainers of the Python package [31]. While the `stress-ng` induced loads are running on the virtual machine, the laptop connected to the oscilloscope runs a data collection script that collects voltage measurements at a specified sampling rate, and outputs the result to a file. Figure 4.1 details the experimental apparatus at a high level. A picture of the setup is shown in Figure 4.2

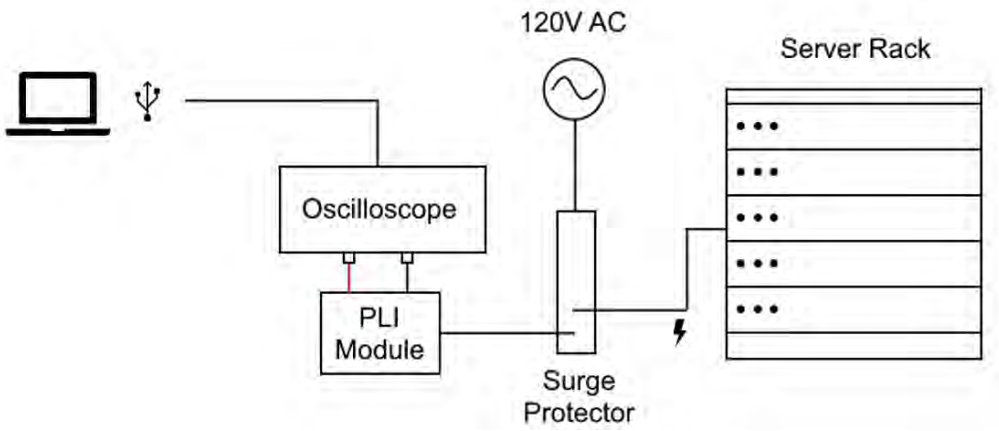

Figure 4.1: Experimental apparatus for monitoring electrical power signal.

4.2 Experimental Results

The described experimental apparatus is used to collect data under varying system load conditions. We explore different test cases under ideal circumstances (minimal background noise from other servers on the rack), as well as noisier conditions with other servers on the rack under load. We also investigate the effectiveness of the side channel under noisy conditions by adding various levels of simulated noise to the signals collected under the ideal circumstances.

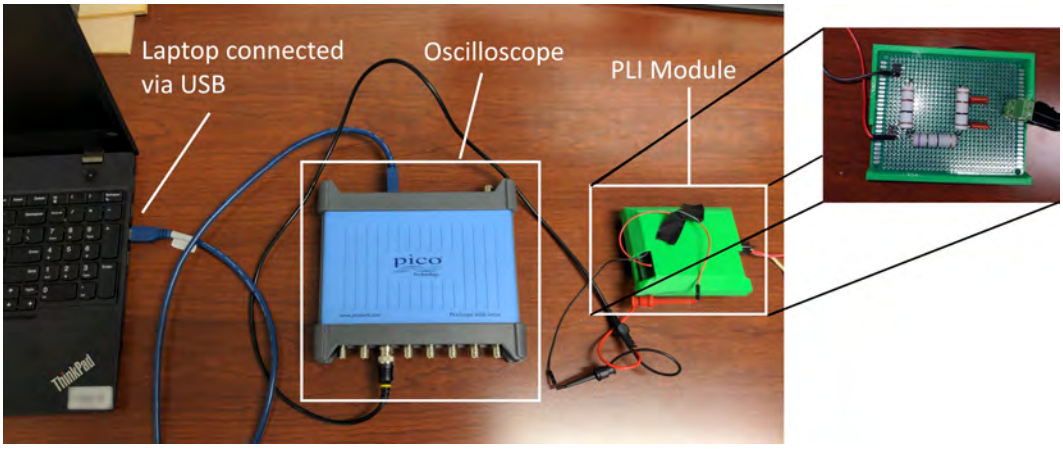

Figure 4.2: A picture of the experimental apparatus, featuring the PLI module, oscilloscope, and laptop.

4.2.1 Signal Processing Procedure

Graphs of voltage versus frequency are useful for observing the EPLF in real-time. In order to evaluate the viability of a covert channel, we must be able to reliably detect the high and low states occurring over a time range. A transformation that enables visualization of the time-varying states is in the form of a voltage magnitude spectrogram. The magnitude spectrogram is displayed as a heat map showing how the voltage changes represented as different heat map intensities with the vertical axis representing frequency and the horizontal axis representing time.

The output of the data collection script is a single array of values that represent the voltage signal in the time domain. In order to produce a spectrogram, the raw voltage information collected from the oscilloscope is processed via a short-time Fourier transform that produces a complex-valued matrix, such that each vertical slice represents the frequency spectrum of the signal at a given point in time. The matrix is then logarithmically-scaled with units of decibels based on the magnitude of the complex values in order to reduce the dynamic range of the output. The final result shows the voltage intensity in dBW at each frequency (vertical axis) for each point in time (horizontal axis). In particular, we use a spectrogram with a window length of 2048 points (about 20 ms), 75% overlap, and von Hann windowing. The signal processing procedure is performed using open-source

packages in Python. For signal processing algorithms and spectrogram visualizations we use Librosa [32] and SciPy [33]. The visualizations are supported by Matplotlib [34] and Seaborn [35]. Finally, we make use of Numpy for representing the arrays of data during processing [36].

If the proposed side channel is viable for establishing a covert communications channel, we should expect to see an easily distinguishable pattern of high intensity followed by low intensity, since the shell script is modulating an alternating bit sequence, at around the 30-35kHz range that is correlated with the idle and high CPU utilization periods respectively.

4.2.2 Initial Visual Results

Before performing formal experiments, we used Picoscope's software to observe visualizations of the data obtained from the oscilloscope in real-time as an initial test. The oscilloscope includes a spectrum analyzer function and the software generates a live frequency spectrum of the input signal that displays the intensity of the signal at different frequency components. We observed this graph while running the power modulation shell script on one of the servers in the rack, with the goal being to determine if the modulation in power consumption is visually discernible. The results were quite clear in this initial visualization.

Figure 4.3 shows the real-time graph when the virtual machine is idle, with no high-load tasks running. There is a noticeable spike in the spectrum at around 30-35 kHz as denoted by the blue box annotating the graph.

Figure 4.4 shows the same graph when a 100% CPU utilization stress test is running. The spike that was present under idle conditions falls dramatically in magnitude as also emphasized with a blue box. During the real-time experiment, this fall and rise pattern correlated exactly with the initiation and ending of the stress test load. Since this rise and fall is occurring at the 30-35 kHz range, this indicates that the data collection script should sample above the 70 kHz Nyquist rate in order to properly capture the EPLF.

4.2.3 Power Spectral Density Estimation Using Welch's Method

This live view, however, is quite susceptible to noise. A better approach for quantifying and visualizing the frequency spectra under different conditions is to use Welch's method to calculate a power spectral density (PSD) estimate of the signal. For this purpose, we collect

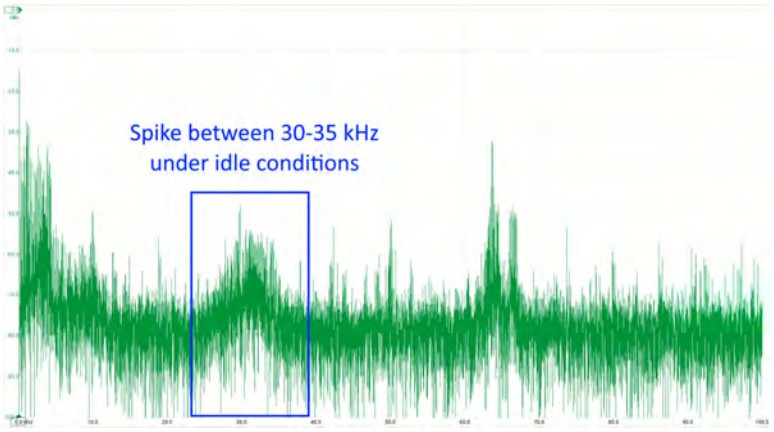

Figure 4.3: Real-time frequency spectrum during idle conditions showing voltage amplitude (dBu) versus frequency (kHz).

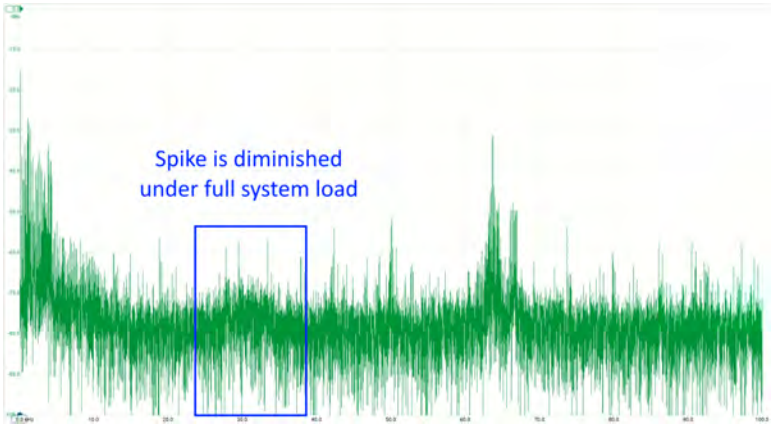

Figure 4.4: Real-time frequency spectrum during load conditions showing voltage amplitude (dBu) versus frequency (kHz).

two thirty second traces, one under idle conditions, and one under 100% CPU utilization. Applying Welch's method to these time domain signals provides an estimate of the spectral content of the electrical power signal under the two different system states. The output of Welch's method is in units of $\frac{V^2}{Hz}$ that has a relatively high dynamic range. Thus, for visualization purposes, the result is logarithmically scaled so that the units are dBW (decibel Watts) according to equation (4.1), where P_{xx} represents the PSD in units of $\frac{V^2}{Hz}$.

$$P_{dBW} = 10 \cdot \log_{10}\left(\frac{P_{xx}}{1W}\right) \tag{4.1}$$

Comparing the two power spectra provides the frequency ranges where the spectral content of the signal variations is most distinct. Figure 4.5 illustrates the power spectral density estimates as calculated by Welch's method.

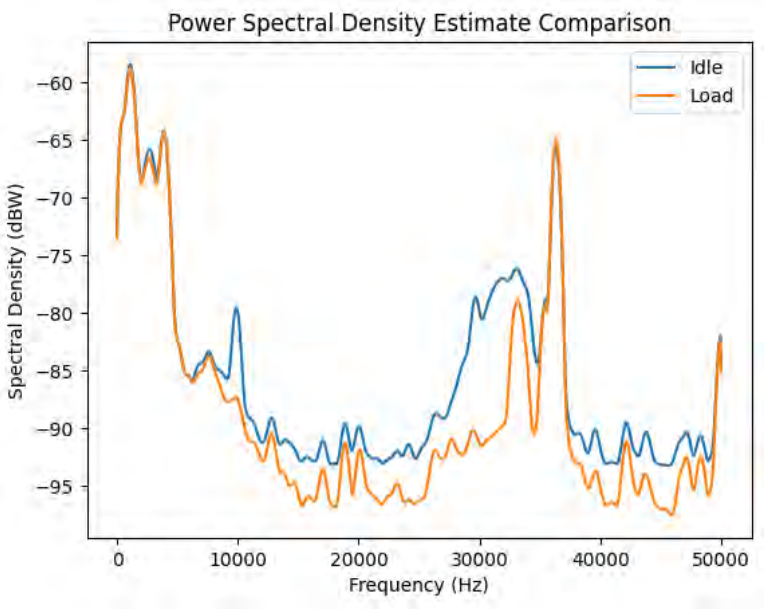

Figure 4.5: Spectral content comparison using Welch's method for power spectral density estimation.

The power spectral density estimates further demonstrate the clear difference between the two states of the system, not only in the 30-40 kHz range, but also at 10 kHz. This result is useful for developing a methodology for quantifying the PSD differences for the purpose of predicting the state based on measurements.

4.2.4 Spectrograms

The previous visualizations are helpful for showing the differences between the system in one of the two states. However, when the system is alternating between the two states (as would be the case if an attacker was encoding a stream of binary information), the spectral content of the power signal will be changing over time. For this reason, we use a short-time Fourier transform and a magnitude spectrogram to visualize the signal in the frequency domain as its spectral content changes with time. A magnitude spectrogram can also be used to visualize the traces captured under fully idle, or full system load conditions, to compare the differences. Again, for visualization and data processing purposes, the high dynamic range of the STFT output is logarithmically-scaled. The process is slightly different than that in Equation (4.1) however, since the magnitude of the STFT output represents a voltage magnitude, the magnitude is squared to obtain a power quantity. Equation (4.2) details the process, where Z_{xx} represents the STFT matrix.

$$P_{dBW} = 10 \cdot \log_{10}\left(\frac{|Z_{xx}|^2}{1W}\right) \tag{4.2a}$$

$$= 20 \cdot \log_{10}\left(\frac{|Z_{xx}|}{1W}\right) \tag{4.2b}$$

Figure 4.6 contains the two spectrograms generated from the 30 second traces capturing the electrical power signal during static system states. The difference between the spectrograms correspond with what would be expected from the real-time view and PSD estimation, i.e. high intensity at 10 kHz and 30-40 kHz frequency ranges for idle conditions that are not present during high system load conditions.

To provide a data source for the covert channel, we use `stress-ng` in a similar manner as previously described to create an alternating pattern of system load and idle states to represent an alternating bit sequence that is present within the channel. For this test case, one server with one virtual machine attempts to generate a square wave (sharp transitions between high power consumption and lower power consumption states) with a period of two seconds. The shell script repeats by looping over a sequence of a 1 second stress test, at 100% CPU utilization, followed by a 1 second period of inactivity. This is repeated while the

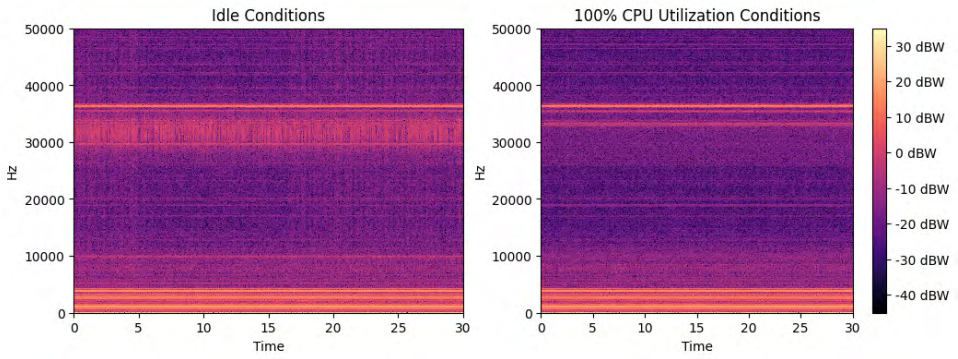

Figure 4.6: Magnitude spectrograms comparing spectral content over time for idle and load states.

data collection script records the signal in units of millivolts. Since the frequency spectrum was affected in the 30-35 kHz range, we use a sample rate of 100 kHz to exceed the Nyquist bound. The data is collected over a period of 30 seconds. After applying the data processing steps, the spectrogram is generated as shown in Figure 4.7.

This spectrogram corroborates our hypothesis that data can be exfiltrated. While the machine is in an idle state, we observe bands of high intensity in the 30-40 kHz range. While the machine is under high stress, the high intensity region fades to match the background magnitude value. An interesting observation is that the high and low states are recognizable not only between 30 and 40 kHz, but also quite clearly at 10 kHz, although the bands are not as intense. This is important to note because it means that a high frequency sampling circuit is not strictly necessary to be able to observe the power consumption as an EPLF signal.

For the sake of demonstrating reproducibility, the experiment is repeated in the same fashion, except with the square wave having a period of ten seconds (five seconds low, five seconds high) corresponding to a lower data rate of alternating bit values within the channel. Figure 4.8 shows the spectrogram generated from this lower data rate process.

The spectrogram shows once again that the square wave is easily identifiable, both between 30 and 40 kHz, and at 10 kHz. Thus we conclude that under minimal background noise, the data within the channel is observable. We now turn our attention to non-binary

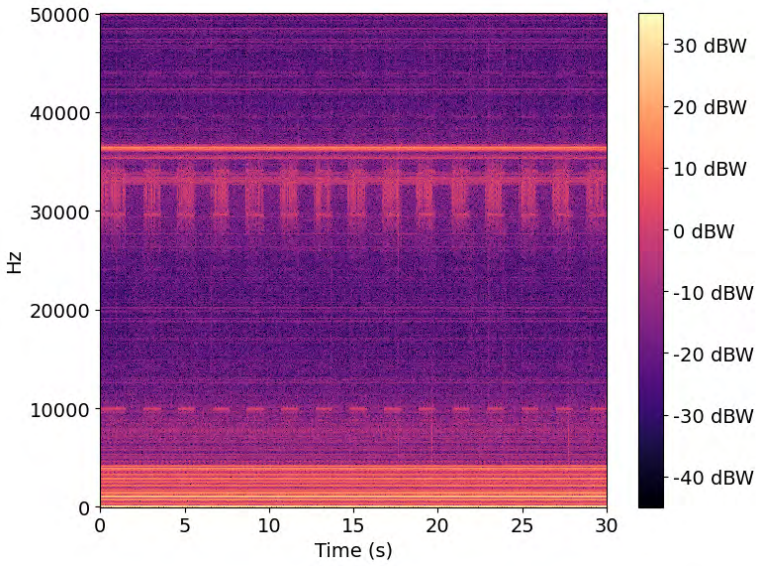

Figure 4.7: Magnitude spectrogram for covert channel sending symbols at a rate of 1 bit per second.

exfiltration, using the CPU usage to influence the EPLF signature.

4.2.5 Ternary State Identification

Thus far, we have demonstrated that a covert channel is possible with a communication rate of at least 1 bit per second. There are two ways to improve the communication rate: decreasing the time interval for the stress tests, or using a more efficient encoding scheme. Unfortunately, `stress-ng` cannot run tests for sub-second intervals. However, it can precisely control the percentage of CPU utilization that ideally would allow more than just two discernible intensity levels in the spectrogram. If this is the case, it would mean that a ternary (or better) encoding scheme could be used to increase the communication rate by utilizing more than two data symbols. For experimentation, the shell script is modified to cycle from idle, to 50% CPU utilization, to 100% CPU utilization, repetitively. We expect this to show three different levels of intensity in the spectrogram that could be used as three separate data symbols in a streaming information side-channel.

The spectrogram in Figure 4.9 shows the processed thirty second trace. The goal in this test was to generate discernible states of low, medium, and high intensity. As in the prior

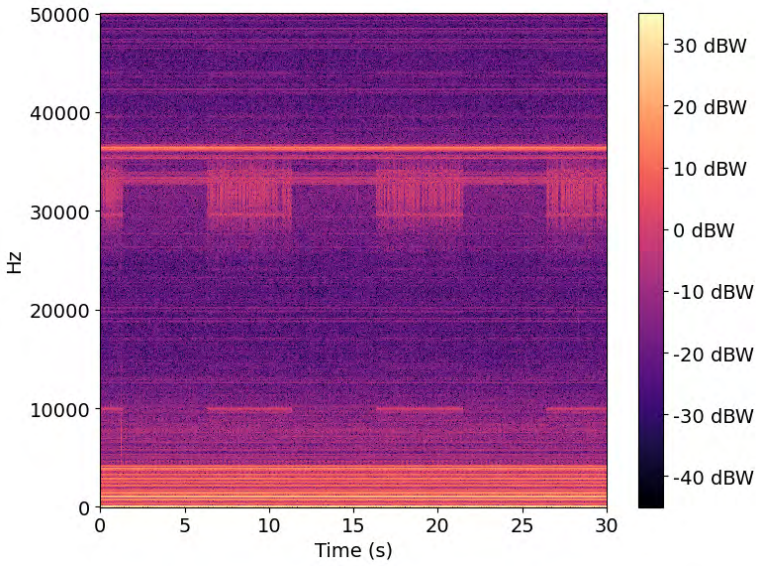

Figure 4.8: Magnitude spectrogram for covert channel sending symbols at a rate of 0.2 bits per second.

tests, the low and high states are quite different visually. However, the medium state is not as clearly defined. Interestingly, the pattern does not manifest as clearly at the 10 kHz range. It is unclear if this ternary state can be easily detected automatically. For instance, a machine learning algorithm may be able to discern the different states from one another although visual inspection by a human is more difficult as compared to the binary encoded signal.

4.3 Binary Information Exfiltration

The magnitude spectrograms show that it is possible to encode information from the virtual machine by modulating the CPU utilization and observing the system's EPLF. If an attacker were to implement this covert channel for data exfiltration, they would need a method to decode the binary information from traces of the power signal. This could be done by classifying the observed EPLF of the system at a given time as either a 0 or 1. Thus, the next step in our analysis is to devise such a classification model, and evaluate its performance under varying conditions. For this purpose, we use an ROC curve to determine classification thresholds, and measure the area under the ROC curve (AUC) and classification accuracy as

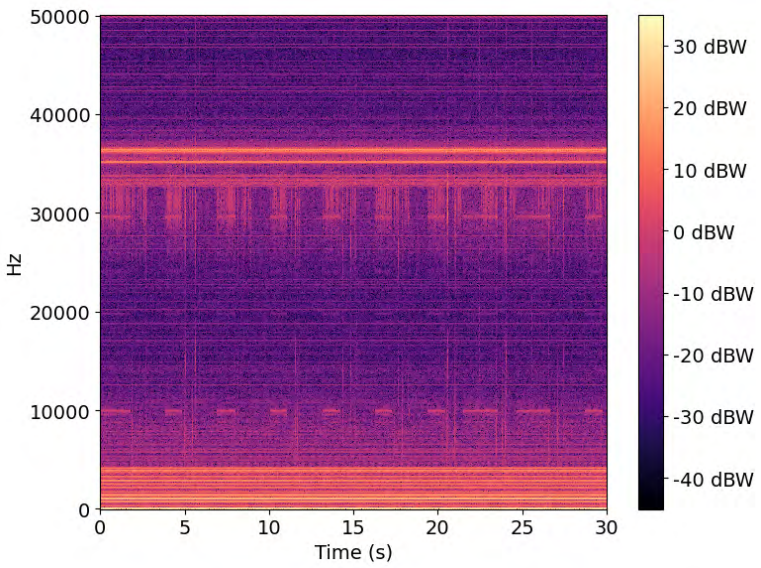

Figure 4.9: Magnitude spectrogram for covert channel sending ternary data symbols.

performance metrics. We also evaluate the resilience to noise by simulating noisy conditions using additive white Gaussian noise (AWGN). Finally, we perform a hyperparameter search to optimize the classifier to operate under the most noise resilient classification framework.

4.3.1 Classification Task Formulation

In the practical application of a covert channel, a classification model would need to classify segments of a captured signal as either a low power consumption state, or a high power consumption state (or possibly low, medium, and high states in the case of a ternary encoding scheme). These high and low states will be represented with labels of 0 and 1, representing the data bits present within the channel.

The length of the signal segment per data symbol could be variable depending on the application, and levels of background noise. For generalization purposes, we define a sample as one column of the STFT matrix. Each column of the STFT is representative of the spectral content of the signal over a small interval of time. The length of time represented by each column is dependent on the time resolution of the STFT, determined by the window length and percent overlap used in the STFT. Under this framework, the high and low states of the system would be recognized by the signature of their spectral content.

4.3.2 Samples and Scoring

First, the raw signals in the time domain are used to create an STFT matrix. For initial testing, the free parameters of the STFT are kept the same as described previously for the creation of spectrograms. Given two STFT matrices, one for the idle signal and one for the load signal, we can extract samples of either class, and use an ROC curve to evaluate the classification performance at varying classification thresholds.

Each column of the STFT is labeled as either 0 or 1, based on the signal from which the STFT is computed. The second piece of information needed for an ROC curve is a score for each sample, which determines the predicted class based on a given scoring threshold for dividing the classes. For this purpose, we develop a methodology for scoring samples based on their spectral content.

4.3.3 Computing a Spectral Content Signature

In order to achieve this objective, we compare the spectra of the idle and load signals and determine the frequency ranges of interest where the signals are most distinct in the frequency domain. This process is essentially a computational approach for what was observed in Figures 4.3 and 4.4, where it is possible to visually distinguish the system state by human observation of the spectrum analyzer display.

This can be achieved computationally by taking a small sample (5%) of the traces that capture the system in a static state (idle or load). From each of the samples of these traces, we compute the STFT, logarithmically scale the result, and average the magnitudes across the time frames of the STFT. The result is a single spectrum, representing the average logarithmically scaled magnitudes of the Fourier coefficients from each time frame (column) of the STFT. When this process is accomplished for the signals for each system state, the results can be subtracted to produce an array of the difference (measured in dB units) between the two spectra. From this array of differences, we select the frequency bins to use for classification if the difference surpasses a given difference threshold specified with a dB unit that serves as a hyperparameter of the classification process. This process is visualized in Figure 4.10. Once the frequency bins have been selected, the remaining 95% of the time domain signals are used for assessment of classification performance.

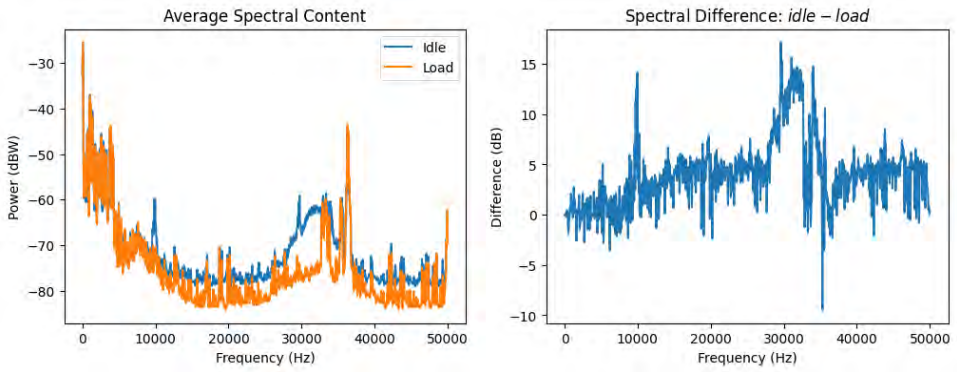

Figure 4.10: Comparison and subtraction of mean spectral content from the STFT.

Figure 4.10 shows, on the left, the comparison of spectral content between the idle and load signals by averaging across the time frames of the STFT. This corresponds roughly with the real-time spectrum analysis view, and the PSD calculation. On the right is displayed the subtraction of the averaged spectra. This difference plot visualizes the regions of the signal in the frequency domain where the spectral content is most distinct. As could have been expected from the PSD visualization in Figure 4.5, the difference is, once again, most pronounced at 10 kHz and 30-40 kHz. Given the difference between the two signals at a set of frequency bins where the difference exceeds a chosen threshold value in dB, the mean value of the frequency bins should summarize the distinction in a singular value. Thus, the score is defined as the mean value of the selected frequency bins. For assessing classification performance in this task formulation, we use an ROC curve to test different thresholds in the dynamic range of the score as classification boundaries, to get an understanding of the generalized performance of a classification model on this data.

4.4 Evaluating Noise Resilience

Since the signals are quite distinct in the frequency domain at certain ranges, it would not be unexpected for the ROC curve to show near-perfect performance under the relatively noise-free testing environment. The data for this experiment is collected under ideal conditions with minimal background noise obfuscating the signal. In a practical application of a covert channel however, there would likely be significant background noise from other

virtual machines running on the same host machine, or different host machines in the server rack that share the same power source. Accordingly, this investigation would be incomplete without attempting to quantify the classification process' resilience to noise.

For this purpose, we use AWGN to simulate the addition of noise to the signal. By adding a random noise vector to the signal in the time domain that conforms to a Gaussian distribution to simulate white noise, we can observe the decline in classification performance as the SNR decreases. This observation will provide an idea of how effective the covert channel would be under more realistic, noisier conditions. It also provides a point of comparison between the classification methodologies.

Quantitatively, there are two metrics that we use to measure the performance under noise conditions: area under the curve (AUC) of the ROC curve, and accuracy. Accuracy is calculated by using the classification threshold of the equal error rate to predict a 0 or 1 for each sample, and computing the percentage of correct predictions. For ROC analysis and metric calculations, we use an open source machine learning package in Python, `scikit-learn` [37].

4.4.1 Noise Results with Static STFT Parameters

As outlined above, the classification procedure uses the mean of the magnitude in dBW of the Fourier coefficients at the frequency bins of interest as a sample. We test a range of SNR values from 20 dB down to -5 dB.

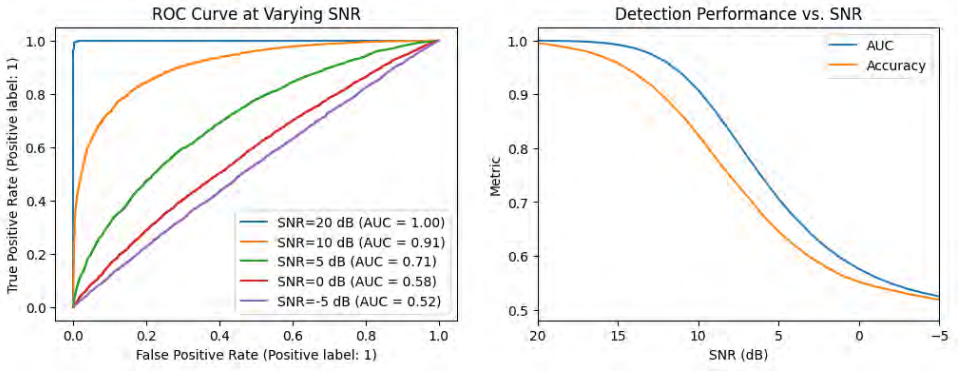

Figure 4.11: Classification metrics for initial STFT parameter selection with AWGN.

Figure 4.11 shows the ROC curves for increasing noise levels on the left. On the right, the AUC and accuracy (vertical axis) are plotted against decreasing SNR on the horizontal axis. These plots visualize the decaying classification performance as noise increases. From a visual point of view, a more noise resilient classification method should show a higher-valued knee for the ROC curves and a more gradual downward slope for AUC and accuracy against decreasing SNR. The ROC curves show that the classification is near-perfect until the SNR reaches about 20 dB. As the SNR decreases past 20 dB, the performance deteriorates, and the AUC score reaches random chance levels (0.52) by -5 dB. Given this promising initial result, we investigate methods for optimizing the classification performance under noisy conditions.

4.4.2 Noise Results Using DPSS Windowing

One method for improving performance could be to apply windowing in the frequency domain in an attempt to reduce the effect of noise. For this purpose, we use a multi-tapering approach with the discrete prolate spheroidal sequences (DPSS) [38]; i.e. a specific sequence of window functions. The classification procedure is similar using DPSS, except the STFT is transformed by multiplying each complex-valued column by several DPSS tapers, and taking the mean of the products. The DPSS tapers are computed according to the following free parameters in Table 4.1:

Table 4.1: Parameter selection for DPSS windowing.

Parameter	**Value**
Window Length	2048
Standardized Half Bandwidth (NW)	500.0
Number of Windows (K_{max})	5

Each column in the STFT is multiplied element-wise by the first K_{max} tapers in the

DPSS sequence, to produce K_{max} products. The mean of the products across each frequency bin is taken to produce a single complex-valued vector of Fourier coefficients. The mean of the logarithmically-scaled magnitudes is understood as a sample in this method. After the DPSS pre-processing is complete, the same noise resilience testing procedure is applied. Figure 4.12 shows the classification performance with DPSS windowing in the frequency domain.

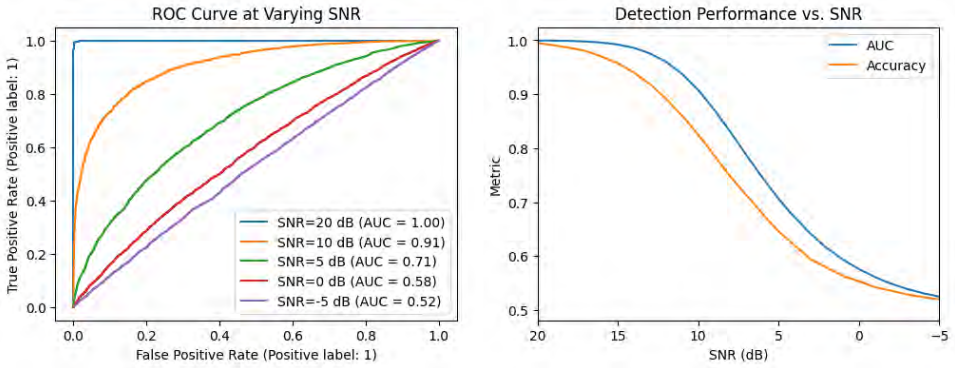

Figure 4.12: Classification metrics for initial STFT parameter selection with DPSS windowing applied.

Initially, these results seem to indicate that using DPSS does not seem to have a significant impact on the classification performance. To confirm this, we observe the weighted averages of the AUC scores, using weights that increase linearly with SNR, as a measure of noise resilience. After performing classification with and without multi-tapering, the weighted AUC score is the exact same for each method, equalling 0.63 for a SNR range of 20 dB down to -5 dB. Thus, it can be concluded quantitatively in this scenario that multi-tapering with DPSS does not offer any significant improvement in terms of noise resilience.

4.5 STFT Parameter Optimization

In addition to applying multi-tapering in the frequency domain with DPSS, another approach for potentially improving classification performance is adjusting the parameters of

the STFT. The methodology so far has assumed a static free parameter selection for the STFT. However, there are different approaches in terms of window length, window type, and overlap percentage that are worth investigating as well to see if there is an effect on classification performance.

Moreover, these parameters have a direct effect on the frequency resolution, and time resolution of the STFT, which in turn affects the theoretical data bit rate of the channel. For these experiments, we match the window length to the FFT (fast Fourier transform) length, i.e. the number of Fourier coefficients which are computed from segments of the time domain signal. For this reason, a larger window length equates to a more precise frequency resolution, since each frequency bin represents a smaller portion of the frequency range of the signal. However, this also means that the time resolution and the frequency resolution of the STFT are inversely correlated; as the frequency resolution becomes more precise, the time resolution becomes less precise, and vice versa. This occurs since a larger window length means that the signal is broken up into larger segments in the time domain, so the resulting time frames (columns) of the STFT represent larger intervals of time.

Because of this phenomenon, we can expect a trade-off between noise resilience, and the theoretical data bandwidth of the covert channel as consistent with Shannon's SNR versus data rate theorem. That is to say, a better frequency resolution should result in superior classification performance at lower SNR, since there are more frequency bins to use in differentiation of the system states. On the other hand, a better time resolution results in a better theoretical data bandwidth. For this experiment, we define the theoretical data bandwidth as the maximum bit rate, measured in bits per second, that could be possible within the covert channel, given the time resolution of the STFT. This is calculated based on the window length, and overlap percentage. Given a window length N, a percent overlap p, and a sampling frequency f_s (the sampling frequency is 100 kHz throughout all trials), the theoretical data bandwidth of the channel, in bits per second, is calculated as shown in equation 4.3.

$$Bandwidth = \left[\frac{N}{f_s}(1-p)\right]^{-1} \tag{4.3}$$

Thus, when optimizing the free parameters, we have two inversely proportional objective functions to consider: data bandwidth, and classification performance (AUC). In the following section, we provide details of our procedure for testing hyperparameter selections in order to observe the trade-off between data bandwidth and classification performance. We also decide on a set of parameters that maximizes either objective function. Once found, we can use this parameter set to quantify the noise resilience of this method of covert communication.

4.5.1 Procedure

In Python, we make use of a package called Optuna [39] for the purpose of parameter optimization. Similar to the previous tests using AWGN, we assess the classification performance as noise increases (SNR decreases). The most noise resilient model is the one whose performance deteriorates the least as the SNR decreases. We quantify this by classifying the system state at different noise levels for each set of parameters. Then the classification performance is summarized by taking the weighted average of the AUC scores over the SNR range. The weights increase linearly as the SNR decreases, so that the AUC for lower SNR values factors more heavily into the overall score.

While the classification performance has to be determined by experimentation with simulated noise, the theoretical data bandwidth is deterministically calculated using a given set of STFT hyperparameters, as shown in Equation 4.3.

When optimizing a set of hyperparameters, Optuna performs a study, in which some number of trials is performed, each trial representing a selection of values for the hyperparameters. When finished, the value of the objective function is reported for each trial. When all trials have completed, the optimal hyperparameter values can be determined by examining the trials with the highest objective function scores. In our case, a trial consists of the following steps:

1. Select test values for the hyperparameters
2. Determine the frequency bins that will be used for classification
3. For each SNR in the range of 10 to -20 dB (increments of 1 dB):
    (a) Add noise to the time domain signals to achieve the given SNR

(b) Take the STFT of each signal and score each time frame by the mean of the magnitudes in dBW at the selected frequency bins

(c) Report the AUC for this SNR value

4. Report the weighted average of the AUC scores for all SNR, and the theoretical data bandwidth for this selection of hyperparameter values

4.5.2 Hyperparameters

Aside from the parameters of the STFT, another hyperparameter that may affect classification performance is the difference threshold at which frequency bins are selected for inclusion in the calculation of the mean spectral content. As was portrayed in Figure 4.10, this is determined by averaging across the time frames of the STFT for a sample of the idle and load signals, and subtracting the two spectra, which produces an array of the differences between the two signals in dB. If the difference in dB is greater than or equal to the difference threshold, than the frequency bin represented at that index is considered in the computation of the classification score. Thus, the difference threshold may be an important hyperparameter to consider for optimization.

In summary, the four hyperparameters that we optimize over are: window length, window function type, overlap percentage, and difference threshold. Each hyperparameter is assessed with a different range of possible values in order to find the most optimal combination.

Table 4.2: Summary of hyperparameters and the range of values used for testing.

Hyperparameter	**Possible Values**
Window Length	$[2048, 131072]$
Window Type	$\{Hann, Hamming, Blackman\}$
Overlap Percentage	$[0\%, 25\%, 50\%, 75\%, 90\%]$
Difference Threshold	$\geq 1\ dB$

4.5.3 Optimization Results

After performing a study consisting of 200 different selections of hyperparameter values, we found that the most important factor is the window length. In Optuna, the parameter importances can be calculated using fANOVA [40], a method for evaluating the importance of a model's hyperparameters. Figure 4.13 shows the importance of the parameters relative to the weighted average of the AUC scores.

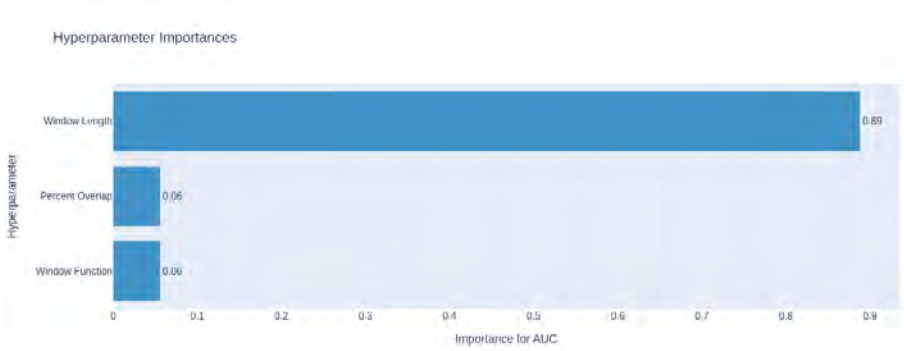

Figure 4.13: Hyperparameter importance scores according to fANOVA.

This result confirms that window length is by far the most important factor. Window function and overlap percent are not completely unimportant, but their impact on classification performance is not as significant as the window length. This is further confirmed by examining the Pareto frontier plot for the study. The Pareto frontier shows the result of each trial in terms of its theoretical data bandwidth, and classification performance.

Figure 4.14 shows the Pareto frontier plot, with the trials colored by their window size. The trend is clear: a larger window size results in better classification performance. Trivially, a smaller window size results in a better data bandwidth (this was computed deterministically), which comes at the cost of classification performance. The best hyperparameter values for maximizing the objective functions (weighted AUC and theoretical bit rate), are the points which reside on the Pareto frontier, i.e. the points on the curve which are not

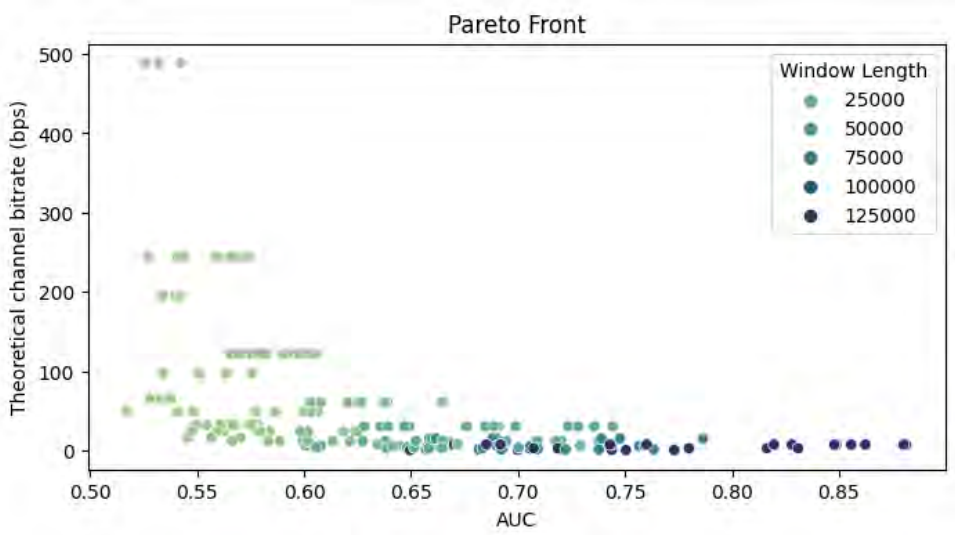

Figure 4.14: Pareto frontier plot showing theoretical bit rate vs. weighted AUC score. Each trial is colored by its window length.

dominated by any others [39]. The hyperparameter values on the Pareto frontier are shown in Table 4.3.

Judging by classification performance, the best set of parameters found in the study were:

- Window Length: 131072 (2^{17})
- Percent Overlap: 90%
- Window Function: Hamming
- Difference Threshold: 17.5 dB

Using this parameter selection, we can visualize the classification process relative to noise levels, to get a better understanding of the best possible noise resilience. Figure 4.15 shows the ROC curves and the AUC/accuracy plotted against decreasing SNR.

The AUC and accuracy are initially near perfect, gradually decreasing until the SNR reaches about -7 dB. At this point, the accuracy score starts decreasing sharply. The AUC starts trending downwards as well around -10 dB, and both metrics reach random chance levels by -30 dB. This means that with these parameters, an attacker that is exploiting a

Table 4.3: Hyperparameter values for the trial runs on the Pareto frontier.

Window Length	**Window Type**	**Overlap Percentage**	**Difference Threshold (dB)**	**Weighted AUC**	**Theoretical bit rate (b/s)**
2048	Blackman	90%	8.5	0.542	488
4096	Hamming	90%	9.5	0.574	244
8192	Hann	90%	16.0	0.605	122
16384	Hamming	90%	17.5	0.665	61.0
32768	Hann	90%	22.5	0.744	30.5
65536	Hann	90%	20.5	0.786	15.3
131072	Hamming	90%	17.5	0.881	7.63

covert channel for unauthorized data exfiltration could expect near-perfect symbol detection accuracy up to an SNR of about -8 dB. After this point, accuracy would decrease sharply, but it would still exceed random chance classification until around -30 dB SNR. It may be possible to achieve even better noise resilience using larger window sizes, however, because we only tested a 1 second period per system state in our experiments, at this time it makes sense to evaluate the noise resilience of the covert channel when the window size is close to this one second period (a window size of 131072 is about equal to 1.3 seconds at our sample rate of 100 kHz).

4.5.4 Analysis

With the parameters optimized for noise resilience, we found that the maximum theoretical bit rate for communication would be about 7.63 bits per second, as determined by the time resolution of the STFT. This number represents the rate of communication an attacker would be capable of, assuming it was possible to alternate between systems states over seven times per second. Future experimentation would be required, however, to analyze just how quickly the system could alternate between recognizable states of high and low power consumption in a way that is interpretable by analysis in the frequency domain as we

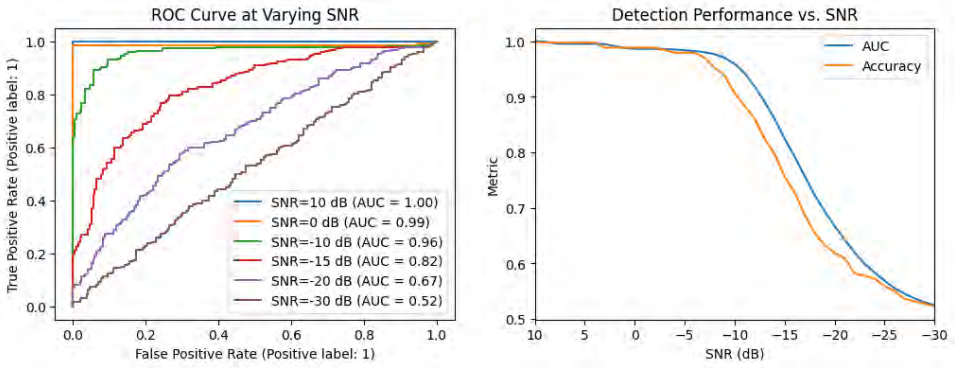

Figure 4.15: ROC curves and classification metrics vs. SNR for optimal hyperparameters.

have shown thus far. This would provide a better estimate of the maximum rate of covert communication using this channel.

It is likely that some physical process within the server, which responds to system stress, is responsible for the change in the power signal's spectral characteristics. As our experiments with sensor data obtained via IPMI showed (see Figure 3.3), the system's cooling fans respond quickly to changes in the virtual machine's CPU usage, which may be the reason that enables this unauthorized data exfiltration security exploit. There is some physical limit to how quickly the fans could respond to system stress due to mechanical inertia that would result in a physical cap on the maximum communication rate. At this time, however, we can conclude that it is possible to exfiltrate binary information using this method, at a rate of at least 1 bit per second, with a high degree of resilience to background noise, such that greater than 95% accuracy when recognizing data symbols would be possible even at an SNR of -8 dB.

Chapter 5
CONCLUSION

In this research project, we designed and carried out two experiments to show the feasibility of exploiting a power side channel in an enterprise computing environment to leak information about the state of a virtual machine. Using IPMI as a source of sensor data, we were able to correlate the direct power consumption measurements of the server with the system activity of a virtual machine running on the server. Given the known security issues with IPMI and the fact that sensor data can be collected even with minimal privileges, this side channel is a potentially attractive exploitation source for unauthorized data exfiltration by an attacker.

5.0.1 Power Analysis Summary

Using the electrical power signal of an enterprise level server rack as a side channel to perform unauthorized data exfiltration within a network-isolated computing system is shown to be feasible. In particular, we show that binary information can be ex-filtrated from the system at a rate of at least 1 bit per second. The receiver of the information does not need to be co-located with the system under attack—it is only required that the receiver of the exfiltrated data have physical access to the same power line as that being used by the exploited computing system (such as a power outlet in a nearby room connected to a common circuit breaker). Using a frequency based analysis of the data, a clear signal can be received as a binary bit stream. Additional higher-radix data symbols may also be possible, but require further investigation to assess detection accuracy. The experimental results indicate that our hypothesis that a network-isolated virtual computer is capable of leaking sensitive information via a power-based side channel.

Additionally, by simulating noisier conditions, it was shown that information leakage by spectral analysis of the electrical power signal is relatively noise resilient with high data

symbol classification accuracy even at low SNRs. By optimizing the parameters of the STFT, a theoretical covert channel can be expected to achieve greater than 95% accuracy even at an SNR of -8 dB.

5.1 Future Work

5.1.1 Experimental Background Noise Conditions

While our experiments show promise as a proof of concept, our test cases represent somewhat ideal circumstances that may not be typical of an enterprise computing environment. We performed an initial investigation of this limitation by simulating noise conditions with additive white Gaussian noise. While these simulations produced interesting and promising results, they only offer a prediction of how the covert channel could perform under more realistic circumstances. Future research could take the next logical step and evaluate the feasibility of the channel under practically implemented background noise conditions, by powering up and stressing other virtual machines within the rack during a communication session using the covert channel to simulate normal conditions. These results could be compared to our predictions based on noise simulation.

We have performed initial testing with some background noise conditions, but not enough data was collected for a full evaluation. Thus, a far more extensive exploration should be done before drawing any conclusions about the effectiveness of using this side channel for data exfiltration in a typical enterprise computing environment with multiple servers running.

5.1.2 Translation to Analogous Environments

It is possible that the applicability of this side and covert channel analysis extends beyond the realm of bare metal hypervisor servers on network-isolated server racks. A similar approach could be attempted in a variety of different environments, including containerized systems, type 2 hypervisors, or even non-virtualized operating systems running on bare metal that hosts multiple concurrent processes. Further experimentation would be required to show if a similar power analysis approach would work as a method of covert communication in these environments.

5.1.3 Applications of Power Side Channel Data

Finally, a captivating area of side channel research has been the applications of side channel data for attack predictions by malware, intrusions, and other anomaly detection related scenarios. The power side channels identified in this work, either by IPMI or electrical power signal measurement, could be sources of data in this regard. If it is possible to glean information about the system activity of virtual machines from these side channels, it is possible that this information has defensive applications for anomaly detection-related scenarios as outlined in recent literature.

BIBLIOGRAPHY

[1] J. Szefer, "Survey of microarchitectural side and covert channels, attacks, and defenses," *Journal of Hardware and Systems Security*, vol. 3, no. 3, pp. 219–234, Sep 2019. [Online]. Available: https://doi.org/10.1007/s41635-018-0046-1 2

[2] M. A. Taylor, E. C. Larson, and M. A. Thornton, "Rapid ransomware detection through side channel exploitation," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2021, pp. 47–54. 2

[3] F. C. Freiling and S. Schinzel, "Detecting hidden storage side channel vulnerabilities in networked applications," in *Future Challenges in Security and Privacy for Academia and Industry*, J. Camenisch, S. Fischer-Hübner, Y. Murayama, A. Portmann, and C. Rieder, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 41–55. 2, 3

[4] M. Randolph and W. Diehl, "Power side-channel attack analysis: A review of 20 years of study for the layman," *Cryptography*, vol. 4, p. 15, 05 2020. 3

[5] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397. 5, 9

[6] VMWare, "What is a bare metal hypervisor?: Vmware glossary." [Online]. Available: https://www.vmware.com/topics/glossary/content/bare-metal-hypervisor.html 6

[7] Intel, "Out-of-band management (oobm)." [Online]. Available: https://www.intel.com/content/www/us/en/business/enterprise-computers/resources/out-of-band-management.html 6

[8] ——, "Intelligent platform management interface specification v2.0 rev. 1.1," 2013. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html 7

[9] A. Amelkin, "ipmitool," https://github.com/ipmitool/ipmitool, 2022. 7, 19

[10] H. Moore, "A penetration tester's guide to ipmi and bmcs: Rapid7 blog," Oct 2020. [Online]. Available: https://www.rapid7.com/blog/post/2013/07/02/a-penetration-testers-guide-to-ipmi/ 8

[11] A. J. Bonkoski, R. Bielawski, and J. A. Halderman, "Illuminating the security issues surrounding lights-out server management," in *Proceedings of the 7th USENIX Conference on Offensive Technologies*, ser. WOOT'13. USA: USENIX Association, 2013, p. 10. 8, 26

[12] O. Gasser, F. Emmert, and G. Carle, "Digging for dark ipmi devices: Advancing bmc detection and evaluating operational security," in *Traffic Monitoring and Analysis*, 2016. 8, 9, 26

[13] F. E. Betreuer, "Out-of-band network management," 2015. [Online]. Available: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2015-03-1/NET-2015-03-1_10.pdf 9, 26

[14] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 355–371. 10

[15] S. N. Patel, T. Robertson, J. A. Kientz, M. S. Reynolds, and G. D. Abowd, "At the flick of a switch: Detecting and classifying unique electrical events on the residential power line (nominated for the best paper award)," in *International Conference on Ubiquitous Computing*. Springer, 2007, pp. 271–288. 10

[16] S. Gupta, M. S. Reynolds, and S. N. Patel, "Electrisense: Single-point sensing using emi for electrical event detection and classification in the home," in *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, ser. UbiComp '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 139–148. [Online]. Available: https://doi.org/10.1145/1864349.1864375 10, 29

[17] S. Gupta, K.-Y. Chen, M. S. Reynolds, and S. N. Patel, "Lightwave: using compact fluorescent lights as sensors," in *Proceedings of the 13th international conference on Ubiquitous computing*, 2011, pp. 65–74. 10

[18] K.-Y. Chen, G. A. Cohn, S. Gupta, and S. N. Patel, "utouch: sensing touch gestures on unmodified lcds," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013, pp. 2581–2584. 10

[19] M. Enev, S. Gupta, T. Kohno, and S. N. Patel, "Televisions, video privacy, and powerline electromagnetic interference," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 537–550. 10

[20] M. Müller, *The Fourier Transform in a Nutshell*, 08 2015, pp. 39–57. 11

[21] P. Welch, "The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967. 11

[22] P. Stoica and R. L. Moses, *Spectral analysis of signals*, 2005. 11

[23] D. Griffin and J. Lim, "Signal estimation from modified short-time fourier transform," in *ICASSP '83. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 8, 1983, pp. 804–807. 12

[24] C. P. Solutions, "What is signal to noise ratio and how to calculate it." [Online]. Available: https://resources.pcb.cadence.com/blog/2020-what-is-signal-to-noise-ratio-and-how-to-calculate-it 12

[25] Electrical4U, "Rms voltage: What it is? (formula and how to calculate it)." [Online]. Available: https://www.electrical4u.com/rms-or-root-mean-square-value-of-ac-signal/ 12

[26] MatLab, "Add white gaussian noise to a signal." [Online]. Available: https://www.mathworks.com/help/comm/ref/awgn.html 13

[27] Google, "Machine learning glossary," Nov 2022. [Online]. Available: https://developers.google.com/machine-learning/glossary 14, 15, 16

[28] S. Furui, "Chapter 7 - speaker recognition in smart environments," in *Human-Centric Interfaces for Ambient Intelligence*, H. Aghajan, R. L.-C. Delgado, and J. C. Augusto, Eds. Oxford: Academic Press, 2010, pp. 163–184. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123747082000073 16

[29] ColinIanKing, "stress-ng," https://github.com/ColinIanKing/stress-ng, 2023. 20

[30] Z. Wolf., E. C. Larson., and M. A. Thornton., "Data leakage in isolated virtualized enterprise computing systems," in *Proceedings of the 9th International Conference on Information Systems Security and Privacy - ICISSP,*, INSTICC. SciTePress, 2023, pp. 118–123. 28

[31] P. T. Ltd., "picosdk-python-wrappers," https://github.com/picotech/picosdk-python-wrappers, 2022. 30

[32] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," 2015. 32

[33] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. 32

[34] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. 32

[35] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021 32

[36] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2 32

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 42

[38] D. Slepian, "Prolate spheroidal wave functions, fourier analysis, and uncertainty — v: the discrete case," *The Bell System Technical Journal*, vol. 57, no. 5, pp. 1371–1430, 1978. 43

[39] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. 46, 49

[40] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proceedings of International Conference on Machine Learning 2014 (ICML 2014)*, Jun. 2014, p. 754–762. 48