SoK Paper: Power Side-Channel Malware Detection

Alexander Cathis, Ge Li, Shijia Wei, Michael Orshansky, Mohit Tiwari, Andreas Gerstlauer
Electrical and Computer Engineering, The University of Texas at Austin
Austin, TX, USA
{alexander.cathis,lige,shijiawei,orshansky,tiwari,gerstl}@utexas.edu

Abstract

Analyzing power consumption has been proposed as an effective way to detect anomalous system behavior. The power side channel provides a non-intrusive and not easily compromisable out-of-band malware detection mechanism, which is especially beneficial for long-lived and constrained embedded platforms. A variety of detection approaches using ML or other methods have been proposed for a range of target systems and deployment contexts. In this paper, we perform a systemization of prior works to compare and categorize approaches, and provide guidance on limitations and applicability. In the process, we identify 3 research gaps. First, while modern embedded systems often deploy multi-core processors running parallel task sets, existing works only target single-core platforms executing one task at a time. Second, said works don't always apply the correct machine learning formulations, reframing security problems into classic machine learning problems at the expense of information loss and or reliance on stronger assumptions. Third, few if any prior works release their datasets, making it difficult to evaluate and compare approaches and results.

To further investigate these research gaps, we evaluate power-based side-channel detectors on a high-performance embedded multi-core platform running parallel autonomous drone tasks. In addition to the detectors and attacks of prior works, we evaluate an alternative approach that outperforms prior works across all attacks. Finally, we discuss key challenges and considerations for developing a power side-channel detector for a modern, complex embedded system. To support further research in this space, we introduce and release all data collected using our multi-core setup as a new public dataset.

CCS Concepts

• **Security and privacy → Embedded systems security**.

Keywords

Power side-channel, Malware detection, Embedded system security

ACM Reference Format:
Alexander Cathis, Ge Li, Shijia Wei, Michael Orshansky, Mohit Tiwari, Andreas Gerstlauer. 2024. SoK Paper: Power Side-Channel Malware Detection. In *International Workshop on Hardware and Architectural Support for Security and Privacy 2024 (HASP '24), November 02, 2024, Austin, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3696843.3696849

HASP '24, November 02, 2024, Austin, TX, USA

ACM ISBN 979-8-4007-1221-0/24/11
https://doi.org/10.1145/3696843.3696849

1 Introduction

Information leakage through side-channels, such as power draw or electromagnetic (EM) emissions, has been exploited by attackers to exfiltrate data and compromise computer systems [14, 15, 26]. However, these same side-channels can be leveraged for system defense. By comparing against benign system behavior, machine-learning (ML) based malware detectors can be deployed to flag anomalous behavior. When deployed out-of-band, these detectors offer major advantages over traditional in-band, software-based detection systems. Being external to the system, the detector cannot be affected by a compromised application or operating system (OS). Furthermore, these detectors are non-intrusive, requiring no modifications to existing hardware or software, which is beneficial as new attacks emerge. These advantages are crucial for many embedded systems, which are often resource-constrained, have long lifespans, and do not support field updates. Pairing a current probe with a lightweight ML model presents a cost-effective and easy-to-deploy method for securing high-assurance embedded systems.

Power side-channel detectors have garnered widespread research interest. In this paper, we survey the literature on power side-channel detectors by examining deployment targets, ML detector approaches, and relevant threat models. A review of the deployment targets in prior works reveals that they do not evaluate approaches on systems where benign tasks operate in parallel. However, many modern embedded systems, such as drones, utilize complex multi-core platforms where multiple applications are co-located and executed in parallel on independent schedules [7]. We identify the lack of *evaluation on parallel task sets* as the first research gap in this domain.

While reviewing ML approaches, we observe that many studies rely on overly optimistic assumptions regarding real-world anomaly detection performance or misuse ML solutions for problems they were not designed to address. A lot of interest in time-series anomaly detection is largely driven by the desire to transfer the successes of deep learning to other areas [41]. However, a common issue with many anomaly detection systems is the inappropriate utilization of ML tools [34]. We show that this issue is also prevalent in the narrower power side-channel domain. We identify this *inappropriate utilization of ML tools* as the second research gap.

Finally, when comparing the results of various detectors, we observe that few, if any, papers provide access to their datasets, making comparisons and evaluations difficult if not impossible. Moreover, recent findings highlight flaws and trivialities in many popular public datasets for timeseries anomaly detection [41]. If such issues exist with public datasets in the broader ML community, it raises concerns about the appropriateness of private datasets. Thus, we identify the *lack of rigorous public datasets* as the third research gap.

Several Systematization of Knowledge (SoK) papers have previously studied various variables such as deployment strategies and ML approaches for network intrusion detection and general malware detection [10, 34]. Some recent surveys have focused specifically on power side-channel detectors [32]. Our SoK goes further by systematizing the research space, emphasizing unique challenges and considerations in deploying side-channel detectors. Additionally, we contribute empirical studies that explore and address gaps in current power side-channel detection methodologies. Through our systematic review, practitioners can navigate the power side-channel detector landscape and make well-informed decisions on suitable defense techniques tailored to their device and threat model.

We further perform a case study to explore gaps and evaluate prior works on a modern high-performance embedded multi-core platform. We find that despite good performance in the single-core context, existing methods can be easily evaded in a multi-core setup. We implement an alternative ML pipeline that combines an ensemble of problem-specific anomaly detectors, showing that it can outperform other solutions. We also discuss design considerations for detection in parallel contexts.

This SoK makes the following contributions:

- We survey and analyze previous power side-channel detection works and identify three key research gaps; *evaluation on parallel task sets, inappropriate utilization of ML tools* and *lack of rigorous public datasets*.
- We evaluate previous approaches on a multicore system running drone-based tasks, and find that previous approaches are not suitable for parallel tasks.
- We propose a novel approach utilizing an ensemble of mode-specific one-class classifiers that achieves high detection performance while scaling to a large number of benign application modes on multicore systems. Results show that our approach achieves a worst-case ROC-AUC of 0.90 against microarchitectural attacks that previous approaches fail to detect.
- We publicly release our dataset of parallel benchmarks and associated power traces at [1].

The rest of this paper is organized as follows: In Sections 2-4 we survey and contextualize existing malware detection approaches in terms of deployment context, ML pipeline and attacks considered. In Section 5, we further explore the real-world impact of research gaps. Finally, Section 6 discusses key takeaways and recommendations, and Section 7 summarizes and concludes this paper.

2 Power-Based Malware Detection

We begin by summarizing key power-based detectors in Table 1. Papers are categorized by the complexity of their target devices under the taxonomy's first level. Together with the application complexity, this defines the critical factor of deployment context of a detector.

2.1 Target Device

In Table 1 we categorize papers at the highest level based on the complexity of the target device's hardware and system software. Dynamic hardware features such as multilevel caches, hyperthreading, and out-of-order processing expand the spectrum of behaviors that must be classified as benign, while also introducing new attack vectors [19, 25, 28]. Similarly, the presence of an operating system (OS) amplifies the diversity of benign behaviors while introducing new vulnerabilities.

We categorize devices into three classes based on their hardware and OS: microcontroller units (MCUs), mobile phones, and desktop computers. In the MCU class, devices typically feature single-core processors without an OS, allowing only one task to execute at a time. This category includes programmable logic controllers (PLCs), and medical devices. For instance, Sehatbakhsh et al. [33] deploy their detector on a Linux IoT mini-computer used solely for the "Syringe Pump" function, aligning it with MCU characteristics. Similarly, the medical device studied by Clark et al. [11], running Windows XP Embedded, fits within this class due to its limited function.

Power-based detectors operating on MCU-grade devices primarily focus on detecting code modifications. These detectors often sample power consumption at frequencies ranging from GHz to KHz. Research findings indicate that detectors can effectively characterize the power side-channel during benign operation. This capability is facilitated by the constrained nature of MCU-grade devices, which typically exhibit limited variability in behaviors such as PLC codes, single-function operations, and power grid management. Such detectors demonstrate the ability to detect subtle changes in power consumption caused by even a single instruction modification [2, 22, 42].

In the mobile phone class, we encompass mobile phones, Raspberry Pis, embedded System-on-Chips (SoCs) and other devices with comparable computing capabilities. Devices in this class typically feature multi-core processors and are equipped with mobile OSs. Most studies surveyed in this category utilized mobile phones running Android or Windows Mobile OS, with the exception of Wei et al. [39], who employed an Odroid XU3 board.

The increased system complexity of mobile devices precludes the instruction-level detection observed in MCU-grade devices. Instead, detectors characterize higher-level activities such as application events or user behaviors like placing phone calls using coarser-grained sampling rates. When a series of measurements deviates significantly from established patterns of benign behavior, an alert is triggered. As detectors shift from instruction-level to event-level characterization, some train on known malware samples. While this approach aids in detecting previously identified malware, it does not ensure detection of zero-day malware.

The desktop computer class includes the most complex devices, characterized by power-intensive multi-core processors and desktop-grade OSs. While desktop-grade devices exhibit greater power and complexity compared to mobile devices, the primary distinction lies in the deployment of detectors at the board versus the SoC or even on-chip level. We found no significant platform-dependent differences between detectors deployed on mobile devices versus on desktop computers. Furthermore, there were no major differences or complexity increases observed in the benignware and malware evaluated for desktop-grade systems.

Table 1: Power-Based Malware Detection

Device Class	Target Device	Paper	Sample Rate	Parallel [a]	Benign Tasks	Malware	Malware Train[b]	Detection Approach
MCU	MCU	[30]	1.25 GHz	○	Square Root, Matrix Multiply, Bubble Sort	CPROP	○	HMM
	Arduino, Raspberry Pi, Siemans PLC	[38]	125 MHz	○	Normal Machine States	Botnet	●	LSTM, MLP Classifiers, Autoencoder
	PICDEM Z Demonstration Kit	[2]	500 MHz	○	Normal Mode, Turbo Mode	Code Modification	○	Neyman-Pearson
	Allen Bradley SLC 5/03 7 PLC	[16]	1 Hz	○	Turning On/Off PLC Lights	Code Modification, DOS, Replay Attack	N.A. [c]	N.A.
	Distribution Terminal Unit (ARM and DSP)	[42]	1 KHz	●	Power Grid Operation	Code Modification	☾	One-class-, Binary-, Multiclass-SVM
	Medical Devices	[11]	250 KHz	☾	Windows XP Embedded Running Device Software	Keylogger, Popup Launcher, Screen Grabber	●	Multiclass MLP
Mobile Device	HP iPAQ rx4200	[24]	10 Hz	●	Windows Media Player, Bluetooth and Wifi Transfers, User Behaviors	Mobile Worms, Battery Depletion	●	Chi-Squared Distance
	Galaxy S3	[9]	Missing[d]	☾	Idle	Covert-Channels	☾	Tree, MLP, Regression
	Nokia 5500 Sport	[29]	Missing	☾	User Behaviors	Mobile Worms, Spyware	☾	LR, MLP, DT
	HTC Nexus One, Galaxy Nexus	[18]	N.A.	☾	PowerTutor, MyPhoneExplorer, K-9 Mail	Data Heist Gone-in-60-Seconds	○	N.A.
	Galaxy S3, Galaxy Nexus, SCH-1535	[13]	Event Based	☾	User Behaviors	SMS Spam, Rootkit	○	Threshold
	Odroid XU3	[39]	200 Hz	☾	Facedetect, SHA-3, Room Navigation	Evasive uArch Attacks	○	ocSVM, LSTM
Desktop	Dell Optiplex	[8]	58 Hz	☾	Internet Explorer, Windows Registry	Rootkits	○	Stats Ensemble
	Dell 9020	[3]	1 Hz	☾	17 Unique User Applications	Custom Virus	○	F-Test, ARIMA
	Dell Optiplex	[31]	Random	☾	Stress Tests, User Applications	Rootkits, (KBeast, FUTo, Azazel)	●	DT, SVM, rgr, MLP
	Dell Optiplex	[12]	1 Hz	☾	Benchmark Script	Rootkits, (KBeast, FUTo, Azazel)	●	Beholder Project
	Dell Optiplex	[22]	100 Hz	☾	23 Applications and Benchmarks	Ransomware, Worm, Trojan, Backdoor, Rootkits, Virus	●	Supervised Classifiers

[a]○ refers to single-core systems, ☾ refers to multi-core systems executing one task at a time, ● refers to multicore systems with concurrent execution of multiple benign tasks.
[b]○ refers to no training on malware, ☾ indicates studies evaluating multiple detectors, some of which train on malware, ● refers to detectors that do train on malware.
[c]N.A. refers to entries that are not applicable to the featured work. Often applied to works showcasing approach feasibility or lack thereof.
[d]Missing indicates information not found in the featured work.

2.2 Application Complexity

Orthogonal to the target device, we categorize application complexity into two classes, single- and multi-task. Single-task applications exhibit only one behavior at all times, even when running on multi-core platforms. By contrast, multi-task applications increase the number of distinct behaviors. We define *operating modes* as all unique executable combinations of tasks. When multi-task applications run on single-core platforms, tasks will be time-multiplexed such that the number of modes is equivalent to the number of tasks. The detection challenge is exacerbated when multi-task applications run on multi-core systems. As multiple tasks execute in parallel, the of number modes and range of benign behavior exponentially increases. An additional challenge in multi-core systems is that malware can run in parallel with benign tasks. We define modes that include malware tasks executing alone or in parallel to benign ones as *infected modes*, and modes without as *benign modes*. In multi-core systems, where malware may execute in parallel to benign tasks, there is a corresponding infected mode for each benign mode. To effectively detect malware, a detector must be able to distinguish between all possible benign and infected mode pairs.

All studies of the MCU class only consider the single-core context, except for Zhang et al. [42] who evaluate a single operating mode where a task executes on an ARM core and DSP simultaneously. Within the mobile class, Kim et al. [24] focus on detecting multiple malware tasks running concurrently but do not experiment with scenarios involving multiple benign tasks or simultaneous execution of benign and malicious tasks. All other studies within the mobile and desktop class evaluate malware running alongside typical user operations like making phone calls or playing music, yet they not test scenarios with multiple concurrent user behaviors. Overall, we did not see from prior works a multi-core, multi-thread evaluation of all pairwise comparisons between benign and infected modes.

One could consider the individual cores of a multi-core processor as unique systems and apply single-core malware detection techniques to each of them simultaneously. However, this requires on-chip, i.e. in-band power monitors, incurs software overheads, and ignores core interactions from resource or work sharing. We have not seen this approach in practice.

2.3 Detector Context Takeaways

In summary, previous works have not fully evaluated detectors on systems that execute tasks concurrently. To reliably detect malware in this context, a detector must distinguish all possible benign operating modes from all infected operating modes, with a potentially large number of modes and mode combinations. However, existing studies primarily focus on detecting malware running in isolation or distinguishing a single benign mode from its infected counterpart, where malware runs alongside the benign mode of interest. Some studies [9, 27] suggest that detecting malware becomes more challenging when a system is idle, as malware can manipulate performance to evade detection by users or detectors. However, experimental results supporting this claim are currently lacking. We identify a research gap for detectors targeted at multi-core systems: a lack of *evaluation on parallel task sets*.

3 Detector ML Pipelines

In the following, we further investigate the two main stages of the detector pipeline, the ML algorithm and feature selection. Table 2 summarizes existing works along those axes.

3.1 ML Formulation

At the highest interpretation, malware detection is a binary classification problem where behavior stemming from benignware and malware must be distinguished. However, threat model and feature constraints can limit the effectiveness of a binary classifier. Thus, other ML formulations have been used, where prior works utilize various forms of classification, regression, signature matching, and state-transition approaches.

3.1.1 Classification. Malware detection can be framed as a one-class, binary, or multiclass classification problem. In one-class classification (OCC), a classifier trained on benign data aims to classify new data points as either inliers (benign) or outliers (malware). A key advantage of one-class classification is its ability to detect zero-day attacks. Wei et al. [39] illustrate this approach using data from single task executions, while Zhang et al. [42] aggregate data from multiple benign tasks into a single benign class. However, when labeling multiple benign classes as inliers, or even a single class with high variance, the classification function may be expanded enough to misclassify outliers. One-class Support Vector Machines (ocSVM) are a popular model, although an ensemble of single-feature detectors have also been used [8]. While effective in prior works, a single-feature ensemble cannot consider feature interaction or importance without human intervention, and does not offer a continuous decision function. Thus, it may perform worse than a true OCC.

In binary classification, all benign data is grouped into a benign class and all malware into a malware class. This method relies on training with representative malware samples, posing a significant challenge. Another limitation is the lack of detailed interpretation in predictions; while the classifier can classify a test input as benign or malicious, it doesn't specify which specific benign task or malware variant from the training set it most resembles. To address this, it is beneficial to keep the scope of benign inputs well-defined. One method is to adopt a single operating mode for the benign class [42]. Other studies execute a scripted sequence of benign tasks and treat the entire trace as a single benign input [12, 31].

In multiclass classification, benign tasks and malware are typically assigned to their own class during training. During test, inputs are classified as a specific benign task or malware variant. Unlike binary classification, multiclass classifiers provide deeper insights. During training, this can indicate which classes are frequently confused and reveal similarities between benign tasks and malware. However, the challenge of constructing a representative training set persists. Some studies have examined the generalization of trained models to detect zero-day malware. Some works randomly allocate malware samples for training and testing, but specifics about the diversity and representativeness of their training sets remain unclear [11, 12]. Wei et al. [39] avoid training on malware altogether, instead relying on lower confidence predictions to flag anomalies. However, reduced confidence for unseen malware is not guaranteed—a classifier can confidentially label unseen data as in-class.

Table 2: ML Pipelines of Power-Based Malware Detectors

ML Formulation	Training	Features	Detector Algorithms	Papers
One-Class Classification	Benign	BoW, Time-Domain Stats, Frequency-Domain Stats,	ocSVM	[39, 42]
Binary Classification	Benign & Malware	Time-Domain Stats, Frequency-Domain Stats, Delay-Embedded Graph of Discretized Timeseries	SVM, Decision Tree, Linear Regression, Boosted Trees, Beholder Project	[12, 31, 42]
Multiclass Classification	Benign & Malware	Spectrogram, Time-Domain Stats, Frequency-Domain Stats, Time-Domain Stats with Network Packet Features, Delay-Embedded Graph	CNN, GradientBoosting, DecisionTree, ExtraTrees, RandomForest, GaussianNB, SVM, KNeighbors, MLP, LSTM, J48, RandomTree, OneR, JRip, PART, SMO, DecisionTable, ANFIS,	[6, 38, 42, 43], [9, 11, 22, 29], [12]
	Benign	DWT Wavelets	LSTM	[39]
Ensemble of One-Class Classifiers	Benign	Time-Domain Stats, L2 Error, Data-Smashing Distance	Ensemble of Single-Feature Detectors	[8]
Regression	Benign	Timeseries, Hierachical Timeseries,	MLP, LSTM, Autoencoder, DecisionTree, Linear Regression,	[9, 20, 21, 38], [29]
Signature Matching, Statistical Tests	Benign & Malware	Compressed Timeseries, Compressed Spectrogram	Chi2 Distance	[24]
	Benign	Timeseries, Dimensionally Reduced Timeseries, Spikes of Spectrogram, Timeseries with Location Data	SAD, XCORR, Neyman-Pearson Criteria, Komogorov-Smirnov Test, Threshold Function	[2, 13, 33]
State-Transition	Benign	Timeseries, Dimensionally Reduced Timesries	HMM	[30, 37]

3.1.2 Regression. For a regression-based detector, a raw timeseries can provide separability through regression error. The simplest approach is to segment the timeseries into windows and utilize the error betweeen predicted and true future values as a proxy for malicious behavior. The fundamental assumption is that inputs from benign tasks, on which the regressor is trained, will produce lower errors compared to inputs generated by unseen malware. Typically, regressors are trained on a single benign task, but timeseries composed of up to four benign tasks have also been utilized [38].

While regression error can serve as a proxy for class label, it's also susceptible to scenarios where a benign task with highly variable and noisy power traces might produce a higher regression error than a simple trace generated by malware.

3.1.3 Signature Matching. Signature matching refers to the comparison of known patterns or signatures against those from previously extracted benignware or malware. Users have the flexibility to choose whether to train on malware or not. An advantage of this approach is its minimal reliance on feature engineering; many implementations compare raw traces against a known good signature [3, 16]. However, the lack of feature engineering can be a drawback when raw power traces contain too much noise [18].

3.1.4 State-Transition Approach. A few works adopt a state-transition methodology, where a probabilistic state-machine is fitted to benign behavior and the inverse likelihood of state-transition sequences represents anomalousness. Liu et al. [30] exemplify this by initially employing fine-grained sampling to categorize instructions or states within a given task. Once instructions are identified, a control flow graph (CFG) is constructed. Subsequently, a hidden Markov model (HMM) is trained either on the CFG itself or on the instruction stream. During test, the HMM assesses the likelihood that a specific sequence of CFG transitions or instructions originated from the training process. This approach, when applied at the instruction level, has shown efficacy in detecting code modifications, including zero-day vulnerabilities. However, its practicality is constrained by the necessity for a relatively simple benign task where instructions can be clearly identified. Additionally, the instruction sequence or CFG must be sufficiently uncomplicated for an HMM to effectively learn, thus limiting its current application to basic MCU-grade systems.

3.2 Feature Selection

ML algorithms used for malware detection rely on selecting the right features that enable differentiation between benign and malicious inputs. When employing similarity or distance measures for detection, extensive feature extraction is often unnecessary. Aguayo et al. [2] utilize Neyman-Pearson correlation as a distance metric between known good traces and test inputs. Kim et al. [24]

compress time windows and spectrograms using one-way compression via Fast Fourier Transform. Almshari et al. [3] employ F-Test of sample variances and 2-way ANOVA. Sehatbakhsh et al. [33] sample spectra and use spikes as features. Similarity measures can also serve as features for other ML algorithms; Bridges et al. [8] use L2-norm error and Data Smashing Distances alongside time-domain statistical features.

In classification-based detectors, most studies utilize time-domain statistics such as mean, variance, skewness, kurtosis, and interquartile range [3, 6, 8, 11, 18, 22, 31, 39]. Others employ spectrograms derived from FFT of time-series windows [38, 43]. Frequency-domain features such as zero crossing rate, energy, energy entropy, spectral centroid and more may be used [42]. Clark et al. [11] utilize energy from specific frequency bands. Dawson et al. [12] discretize time-series data, construct a delay-embedded graph, and employ the Beholder Project phase-space classifier [17]. In regression-based detectors, modifications to the standard approach include applying discrete wavelet transform to the timeseries and employing a weighted hierarchical timeseries structure [21, 39].

3.3 Machine-Learning Takeaways

Systematizing the ML methodologies from previous studies reveals a reliance on optimistic assumptions. Some detectors train on malware with the expectation that this data will generalize to detect out-of-sample malware in real-world scenarios, yet other works show cases where this assumption does not hold [11, 12, 24]. Many works utilize regression-based detectors, which rely on the assumption that a malware will yield greater error than benignware [3, 9, 20, 21, 29]. Wei et al. [39] propose using classification prediction confidence as a metric for anomaly detection, which, while intuitive, may not reliably detect all types of malware. We consider these as examples of incorrect framing between ML problems and malware detection. In a security context, these assumptions and methodologies should be approached with caution. We highlight a research gap in *inappropriate utilization of tools borrowed from the ML community* [34].

4 Attacks and Datasets

In Table 3, we categorize attacks from prior studies by grouping them into stages derived from the MITRE ATT&CK matrix [35]. The stages that have received the most attention are *Execution* and *Collection/Exfiltration/Impact*. This focus can be attributed to two main reasons. First, these stages highlight critical vulnerabilities and often result in significant consequences. Second, such attacks typically exhibit high power consumption or induce noticeable changes in system state, making them easier for side-channel detectors to identify.

As mentioned in Section 1, few, if any, papers release their power traces, and flaws have been discovered in popular public datasets used in timeseries anomaly detection. The power side-channel community lacks established public datasets for testing. Wu et al. [41] suggest that the timeseries anomaly detection community should address dataset limitations, study implications, and collaborate to develop rigorous and realistic datasets. We advocate for similar efforts for power side-channel research. Currently, the *lack of rigorous public datasets* remains a significant research gap.

Table 3: Attacks Evaluated in Power-Based Malware Detection.

Stage	Instance/Family	Papers
Initial Access	Replay Attack	[16]
Discovery/ Resource Development	Botnet	[38]
Execution	Code Modification	[2, 16, 42]
	Control Flow Hijack	[30, 33]
	Cause Spam	[11, 13]
	Virus	[22]
	Microarchitecture Attacks	[39, 43]
	Evasive μ-Arch Attacks	[39]
	Covert-Channels	[9, 39]
Persistence/ Defence Evasion	Rootkit	[8, 13, 42], [12, 22, 31]
	Backdoor	[22]
Lateral Movement	Worm	[22, 24, 29]
Collection/ Exfiltration/ Impact	DDOS	[16]
	Ransomware	[18, 22]
	Spyware	[11, 29]
	Battery Depletion/ Electrical Theft	[6, 24]
	Data Deletion	[18]
Other	Fabricated Virus	[3, 20]

5 Case Study

To assess the impact of research gaps in deployment contexts and ML formulations of prior works we conducted a case study using an advanced high-performance embedded multi-core development platform executing drone-based workloads. Our setup consists of a Portwell PCOM-C700 Type VII carrier board with a Portwell PCOM-B700G processor module. This module features an 8-core Intel Xeon D-1539 embedded-class processor. Power consumption was monitored using a Hall-effect current sensor clamped around the 12V CPU power cable. Current sensor readings were sampled at 2 KHz. Sliding window extraction was employed to generate model inputs.

For regression-based detectors, windows were divided into a 1000-sample input sequence and a 3-sample prediction sequence with a stride of 1 and 2, respectively. For other machine learning formulations, we transformed each sliding window into a feature vector. The feature vector comprised of statistical features including mean, min, max, amplitude, negative amplitude, and range, and bag-of-words (BoW) features following the methodology of [36]. We created a dictionary by discretizing a trace to 11 levels, forming n-grams and adding the k most popular n-grams to the dictionary, with $n = k = 5$. A bash script running on a separate desktop PC was used to automate the data collection, sending commands to ensure that the target board executed in a specified state before triggering the sampling of data on the oscilloscope. Power traces

for all possible modes were collected and divided using a 50-50 train-test split.

We executed three benign applications representing typical drone tasks: a SHA-3 implementation from the Extended Keccak Code Package [23], a face detection application using the OpenCV library running a video benchmark [40], and an autonomous drone package delivery benchmark from MAVBench [7]. We use Meltdown [28], Spectre [25] and L1 covert-channel [19] microarchitectural attacks as best-case malware for detectability due to their noisy power signatures. Although microarchitectual attacks only form a subset of threats evaluated by prior works and seen in the real world, we focus on their evaluation to test the limits of all evaluated detectors across different operating modes. A detector that cannot detect noisy microarchitectural attacks will not be able to detect less power-hungry software exploits or an evasive power-mimicking attack as described by Wei et al. [39].

5.1 ML Detectors

We select a representative approach for each ML formulation detailed in Table 2. Due to our use of coarse-grained sampling, we omit the state-transition approach, which requires fine-grain sampling for classifying individual instructions. Detectors were implemented using Python, leveraging Scikit-Learn for shallow models and Keras for deep learning models. Each approach is optimized by selecting preprocessing parameters and model hyperparameters via a random parameter search [5]. Parameters yielding best results for each approach's inherent scoring function were selected. Performance estimations were made by blocked cross-validation [4].

We use Support Vector Classification (SVC) as an example of binary classification, Random Forest (RF) for multiclass classification, and both RF and Long Short-Term Memory (LSTM) for regression. We also implement an ensemble of single-feature ensembles following Bridges et al. [8].

In addition to emulating existing methodologies, we propose an alternative malware detector that employs an ensemble of OCCs. This method is similar to [8], but in contrast to fitting an ensemble to each benign task our detector employs an OCC for each benign mode. By training on modes instead of tasks, our detector effectively handles scenarios where multiple benign tasks execute concurrently. Our detector is trained by fitting a one-class pipeline, composed of a OCC and z-score normalizer, to each benign mode. We then ensemble the scores of the one-class pipelines and take the minimum as benign samples should be classified as in-class by at least one pipeline while malicious samples should be classified as outliers by all. We assess our one-class ensemble detector in two implementations, one which uses Isolation Forests (IFs) and the other which uses ocSVMs as the OCC.

5.2 Detector Results

We test detectors for both single- and multi-task contexts and deployment scenarios involving 1, 2, or 3 benign applications running in the system. For the 1-benign and 2-benign scenarios, we evaluated all possible deployment instances depending on the subset of applications included in the system. Within each deployment instance, we report ROC-AUC between all possible pairs of benign and infected modes.

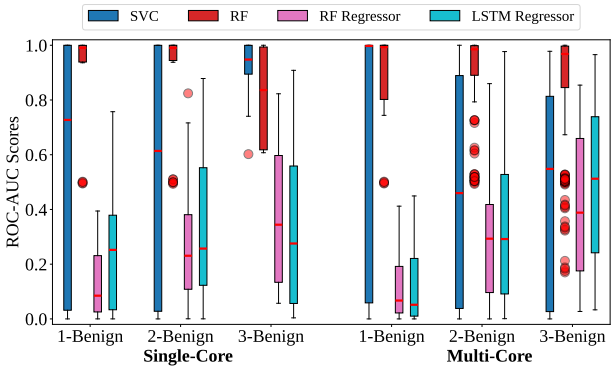

Figure 1: Detector ROC-AUC for single- and multi-core settings in 1-, 2-, and 3-benign contexts.

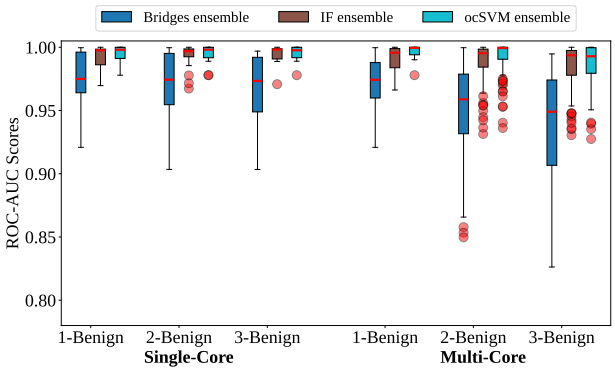

Figure 2: Ensemble detector ROC-AUC for single- and multi-core settings in 1-, 2-, and 3-benign contexts.

Figures 1 and 2 show ROC-AUC distributions of non-ensemble and ensemble detectors across all instances and mode pairs in each deployment scenario. Figure 1 shows that no previous non-ensemble approach exhibits good detection rates across all cases. Moreover, all prior works demonstrate notably weak detection performance in the multi-core context. Underperformance can be attributed to improper problem formulations and the increased complexity of our setup.

Figure 2 shows that for all ensemble models and cases, the ROC-AUC remains above 0.80, greatly outperforming prior works of Figure 1. The Bridges ensemble [8] is consistently outperformed by our ensembles utilizing OCCs, likely due to limitations discussed in Section 3.1.1.

6 Discussion

Our empirical evidence confirms that multi-core presents significant challenges as it increases detection complexity and does not scale well. All detectors perform worse in the multi-core context. As a result, we advise limiting deployments to targets with a restricted number of applications and benign modes. Furthermore,

we recommend the use of applications with periodic power signatures and minimal variance. This approach helps to narrow down the range of benign behavior, thereby reducing the likelihood of misclassification.

Within this context, it is crucial to evaluate each deployment scenario and pairing of benign and infected mode individually, taking into account the worst-case performance, since an attacker can invariably exploit the worst-case scenario to carry out their attack. Worst-case behavior is poor for most existing works as improper formulations can catastrophically fail. It is not difficult to find instances of poor performance when an improper problem formulation is applied against a challenging benchmark. Reliance on average-case performance can lead to misleading conclusions.

Our results also underscore that malware detection hinges significantly on the appropriate formulation of the detection problem, rather than on the application of the most advanced ML models. Our ensemble detector, which utilizes lightweight OCCs, surpasses deep learning approaches due to its design specifically tailored for malware detection. While different threat models may necessitate distinct problem formulations, the fundamental principle remains: the chosen ML tools should be aptly suited to the specific context of malware detection. Basic models can outperform complex ones when applied in the appropriate problem formulation.

Detection outcomes were significantly influenced by factors such as the sampling rate, preprocessing methods, feature engineering, and parameter selection. A well-structured model may fail to deliver effective performance if the data mining process is compromised. Although feature engineering may not be intellecutally novel, it proved to be a critical component for achieving our detection results. Thus, it is imperative to place a strong emphasis on understanding the data and techniques used in its preparation. Advanced ML cannot make up for deficiencies in domain expertise.

While it is vital to achieve good performance from a detector, understanding the detector's limitations is equally, if not more, important. This understanding enables researchers and practitioners to make informed decisions regarding the feasibility of a specific approach. With our platform and detector, we can anticipate robust detection results against microarchitectural attacks. However, we have not evaluated against software-exploiting attacks and recommend caution when considering our approach for threat models that encompass such attacks.

7 Summary and Conclusions

In this paper, we systematically examined power side-channel detectors and pinpointed three significant research gaps: lack of *evaluation on parallel task sets*, *inappropriate utilization of ML tools*, and *lack of rigorous public datasets*. To investigate these gaps, we carried out a case study on a contemporary, complex embedded system running multiple parallel tasks. We found that effective strategies in single-core settings do not necessarily succeed in multi-core environments. We proposed an ensemble-based detection scheme that has been demonstrated to be effective in such settings, and we discussed guidelines and limitations of power-based malware detection. To encourage further development, we publicly released our datasets at [1]. In future work, we aim to further characterize the operating range of our proposed detector and explore alternative approaches for more complex detection scenarios, including heterogeneous hardware platforms, software-based attacks, and power-mimicking malware.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant No. CCF-1901446.

References

[1] 2024. Power-Based Malware Detection Dataset. https://github.com/SLAM-Lab/PMD-Dataset.
[2] Carlos R Aguayo González and Jeffrey H Reed. 2011. Power Fingerprinting in SDR Integrity Assessment for Security and Regulatory Compliance. *AICSP* 69, 2 (2011), 307–327.
[3] Mohammed Almshari, Georgios Tsaramirsis, Adil Omar Khadidos, Seyed Mohammed Buhari, Fazal Qudus Khan, and Alaa Omar Khadidos. 2020. Detection of Potentially Compromised Computer Nodes and Clusters Connected on a Smart Grid, Using Power Consumption Data. *Sensors* 20, 18 (2020), 5075.
[4] Christoph Bergmeir and Jose M Benitez. 2012. On the Use of Cross-Validation for Time Series Predictor Evaluation. *Information Sciences* 191 (2012), 192–213.
[5] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *JMLR* 13, 2 (2012), 281–305.
[6] Konstantinos V Blazakis, Theodoros N Kapetanakis, and George S Stavrakakis. 2020. Effective Electricity Theft Detection in Power Distribution Grids Using an Adaptive Neuro Fuzzy Inference System. *Energies* 13, 12 (2020), 3110.
[7] Behzad Boroujerdian et al. 2018. Mavbench: Micro Aerial Vehicle Benchmarking. In *MICRO*. Fukuoka.
[8] Robert Bridges et al. 2018. Towards Malware Detection via CPU Power Consumption: Data Collection Design and Analytics. In *TrustCom/BigDataSE*. New York.
[9] Luca Caviglione et al. 2015. Seeing the Unseen: Revealing Mobile Malware Hidden Communications via Energy Consumption and Artificial Intelligence. *IEEE Trans. Inf. Forensics Security* 11, 4 (2015), 799–810.
[10] Luca Caviglione, Michał Choraś, Igino Corona, Artur Janicki, Wojciech Mazurczyk, Marek Pawlicki, and Katarzyna Wasielewska. 2020. Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection. *IEEE Access* 9 (2020), 5371–5396.
[11] Shane S Clark et al. 2013. Wattsupdoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices. In *HealthTech*. Washington D.C.
[12] Joel A Dawson, J Todd McDonald, Jordan Shropshire, Todd R Andel, Patrick Luckett, and Lee Hively. 2017. Rootkit Detection Through Phase-Space Analysis of Power Voltage Measurements. In *MALWARE*. Fajardo.
[13] Bryan Dixon, Shivakant Mishra, and Jeannette Pepin. 2014. Time and Location Power Based Malicious Code Detection Techniques for Smartphones. In *RAID*. Gothemburg.
[14] Navyata Gattu, Mohammad Nasim Imtiaz Khan, Asmit De, and Swaroop Ghosh. 2020. Power Side Channel Attack Analysis and Detection. In *ICCAD*. Virtual.
[15] Anupam Golder, Debayan Das, Josef Danial, Santosh Ghosh, Shreyas Sen, and Arijit Raychowdhury. 2019. Practical Approaches Toward Deep-Learning-Based Cross-Device Power Side-Channel Attack. *IEEE TVLSI* 27, 12 (2019), 2720–2733.
[16] Jarilyn Hernández Jiménez, Qian Chen, Jeffrey Nichols, Chelsea Calhoun, and Summer Sykes. 2017. Towards a Cyber Defense Framework for SCADA Systems Based on Power Consumption Monitoring. In *HICSS*. Waikoloa Village.
[17] Lee M Hively and J Todd McDonald. 2013. Theorem-based, data-driven, cyber event detection. In *CSIIRW*. Oak Ridge.
[18] Johannes Hoffmann, Stephan Neumann, and Thorsten Holz. 2013. Mobile Malware Detection Based on Energy Fingerprints—a Dead End?. In *RAID*. Gros Islet.
[19] Casen Hunger et al. 2015. Understanding Contention-Based Channels and Using Them for Defense. In *HPCA*. Burlingame.
[20] Andrei Ioaneş and Radu Tirnovan. 2019. Power Grid Health Assessment Using Machine Learning Algorithms. In *ATEE*. Bucharest.
[21] Halldór Janetzko, Florian Stoffel, Sebastian Mittelstädt, and Daniel A Keim. 2014. Anomaly Detectan for Visual Analytics of Power Consumption Data. *Computers & Graphics* 38 (2014), 27–37.
[22] Jarilyn Hernandez Jimenez and Katerina Goseva-Popstojanova. 2019. Malware Detection Using Power Consumption and Network Traffic Data. In *ICDIS*. South Padre Island.
[23] KECCAK Team. 2020. *XKCP*. XKCP. https://github.com/XKCP/XKCP.
[24] Hahnsang Kim et al. 2008. Detecting Energy-Greedy Anomalies and Mobile Malware Variants. In *Mobisys*. Seoul.
[25] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2020.

Spectre attacks: Exploiting speculative execution. *Commun. ACM* 63, 7 (2020), 93–101.
[26] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *CRYPTO*. Santa Barbara.
[27] Jean-Francois Lalande and Steffen Wendzel. 2013. Hiding Privacy Leaks in Android Applications Using Low-Attention Raising Covert Channels. In *ARES*. Regensburg.
[28] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, et al. 2020. Meltdown: Reading kernel memory from user space. *Commun. ACM* 63, 6 (2020), 46–56.
[29] Lei Liu, Guanhua Yan, Xinwen Zhang, and Songqing Chen. 2009. Virusmeter: Preventing Your Cellphone From Spies. In *RAID*. Saint-Malo.
[30] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. 2016. On Code Execution Tracking via Power Side-Channel. In *CCS*. Vienna.
[31] Patrick Luckett, J Todd McDonald, William B Glisson, Ryan Benton, Joel Dawson, and Blair A Doyle. 2018. Identifying Stealth Malware Using CPU Power Consumption and Learning Algorithms. *Journal of Computer Security* 26, 5 (2018), 589–613.
[32] Valentino Merlino and Dario Allegra. 2024. Energy-based approach for attack detection in IoT devices: A survey. *Internet of Things* 27, 2024 (2024), 101306.
[33] Nader Sehatbakhsh, Monjur Alam, Alireza Nazari, Alenka Zajic, and Milos Prvulovic. 2018. Syndrome: Spectral Analysis for Anomaly Detection on Medical IoT and Embedded Devices. In *HOST*. Washington D.C.
[34] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *SP*. Oakland.
[35] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. 2018. *MITRE ATT&CK: Design and Philosophy*. Technical Report. MITRE.
[36] Jin Wang, Ping Liu, Mary FH She, Saeid Nahavandi, and Abbas Kouzani. 2013. Bag-Of-Words Representation for Biomedical Time Series Classification. *Biomed. Signal Process. Control.* 8, 6 (2013), 634–644.
[37] Xinlin Wang, Insoon Yang, and Sung-Hoon Ahn. 2019. Sample Efficient Home Power Anomaly Detection in Real Time Using Semi-Supervised Learning. *IEEE Access* 7 (2019), 139712–139725.
[38] Xiao Wang, Quan Zhou, Jacob Harer, Gavin Brown, Shangran Qiu, Zhi Dou, John Wang, Alan Hinton, Carlos Aguayo Gonzalez, and Peter Chin. 2018. Deep Learning-Based Classification and Anomaly Detection of Side-Channel Signals. In *Cyber Sensing*. Orlando.
[39] Shijia Wei, Aydin Aysu, Michael Orshansky, Andreas Gerstlauer, and Mohit Tiwari. 2019. Using Power-Anomalies to Counter Evasive Micro-Architectural Attacks in Embedded Systems. In *HOST*. Mclean.
[40] Yongkang Wong, Shaokang Chen, Sandra Mau, Conrad Sanderson, and Brian C Lovell. 2011. Patch-Based Probabilistic Image Quality Assessment for Face Selection and Improved Video-Based Face Recognition. In *CVPR*. Colorado Springs.
[41] Renjie Wu and Eamonn J Keogh. 2021. Current Time Series Anomaly Detection Benchmarks Are Flawed and Are Creating the Illusion of Progress. *IEEE TKDE* 35, 3 (2021), 2421–2429.
[42] Guoming Zhang, Xiaoyu Ji, Yanjie Li, and Wenyuan Xu. 2020. Power-Based Non-Intrusive Condition Monitoring for Terminal Device in Smart Grid. *Sensors* 20, 13 (2020), 3635.
[43] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Bo Li, Peter Volgyesi, and Xenofon Koutsoukos. 2018. Leveraging Em Side-Channel Information to Detect Rowhammer Attacks. In *SP*. San Francisco.