LAPD: Lifecycle-Aware Power-Based Malware Detection

Alexander Cathis, Mulong Luo, Mohit Tiwari, and Andreas Gerstlauer
Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, USA
Email: {alexander.cathis, mulong, tiwari, gerstl}@utexas.edu

***Abstract*—Behavioral malware detectors that analyze power traces have proven to be a promising defense to secure low-complexity embedded systems. However, recent work shows that this approach struggles in more challenging contexts with multi-core processors, multithreaded software, and inherently stealthy or deliberately evasive attacks. These detectors consider the execution of an attack to be a single detection event, whereas real-world malware requires a lifecycle of sequential stages to prime an attack, execute, and perform post-exploit actions. In this work, we augment traditional detectors with a lifecycle-aware backend to collate detection events, generate more informed detection decisions, and thus significantly improve detection performance. Our lifecycle-aware detector is implemented via a hidden Markov model trained on general attack lifecycles. Leveraging signals across lifecycle stages, this approach enables detection of attacks even if certain stages are missed by a traditional detector.**

We evaluate this framework on an 8-core embedded-class Intel Xeon platform running drone-based workloads, using board-level power measurements. When tested against attack lifecycles derived from the MITRE ATT&CK matrix, our lifecycle-aware detector is able to achieve an ROC-AUC score of 0.98 on baseline attack lifecycles. Furthermore, it achieves a score of 0.70 on attack lifecycle specifications that a baseline density-based detector completely fails to detect, due to it's vulnerability to simple attack stalling strategies.

***Index Terms*—attack lifecycle, attack killchain, malware detection, power side-channel**

I. Introduction

Information leakage through physical side channels, such as electromagnetic emissions or power consumption, has been exploited by attackers to exfiltrate data and compromise computer systems [1–5]. However, side channels can also be leveraged defensively. By characterizing a system during normal operation, side channel-based malware detectors can identify anomalous behavior. These out-of-band detectors offer significant advantages over traditional in-band software-based detection methods. Because they operate externally, a compromised application or operating system (OS) cannot undermine the detector. Moreover, these detectors are non-intrusive, requiring no modifications to existing hardware or software, even as new attack vectors emerge. These features are particularly critical for embedded systems, which are often resource-constrained, have long lifetimes, and typically lack in-field update capabilities.

This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

Power-based malware detectors for embedded systems have garnered significant interest due to their cost-effectiveness and ease of deployment [6–10]. However, recent work has shown that existing approaches are often limited to simple systems relying on suboptimal machine learning (ML) techniques, thus performing poorly in noisy deployment contexts, such as on complex multi-core platforms running multi-tasking workloads [11]. The authors proposed a novel state-based ensemble detector that outperformed prior work in such deployment scenarios. However, their approach still failed to detect relatively stealthy or deliberately evasive attacks.

All prior studies focus solely on detecting a single, typically the execution phase of a larger attack lifecycle, neglecting other critical stages. Real-world attacks involve multiple stages within an *attack lifecycle*, designed to prime exploit execution and maximize post-execution persistence and reward. The MITRE ATT&CK® matrix [12] is widely used to categorize attack lifecycles, their sequential stages, and the associated actions or tactics, techniques, and procedures (TTPs). Prior works predominantly focus on the execution stage, as the surrounding stages generally have lower computational demands and thus produce a smaller power footprint. However, even the execution stage can have low overhead or hide behind other system noise. At the same time, knowledge of an attack's lifecycle can enhance detection confidence by contextualizing detections within the broader attack framework, rather than isolating individual actions or TTPs.

We propose a novel lifecycle-aware power-based malware detection (LAPD) framework designed to enhance performance against inherently stealthy attacks and evasive attack lifecycles. To the best of our knowledge, no prior power-based malware detector has incorporated such lifecycle awareness. Our framework integrates a traditional power-based detector, as used in prior works, with an LAPD backend combining a lightweight tree-based attack classifier with a hidden Markov model (HMM). The classifier categorizes malicious windows identified by the front-end detector into common actions corresponding to attack lifecycle stages. Subsequently, the HMM evaluates the observed sequence of actions, scoring them based on their likelihood of representing a genuine attack.

Our contributions in this paper are as follows:

1) We present a novel lifecycle-aware side-channel malware detector that enhances the performance of established detectors by classifying malicious actions and incorporating the likelihood of various attack sequences and strategies.

2) We evaluate the detector against multistage attacks inspired by the MITRE ATT&CK matrix, encompassing microarchitecture, software, network, and external device attack actions. Our LA detector is able to achieve an ROC-AUC score of 0.98 on baseline attack lifecycles and a score of 0.70 on attack lifecycle specifications that a baseline density-based detector completely fails to detect.
3) We analyze the performance of our LA detector and discuss critical considerations for future implementations.

The rest of the paper is organized as follows, in Section II, we introduce related work, in Section III, we give an overview of the LAPD system, in Section IV, we explain in detail the LAPD backend architecture, in Section V, we describe the experiments and results, in Section VI, we summarize this paper and discuss future directions.

II. Related Work

Power Side-Channel Based Malware Detection. Power side-channel is a very effective method for malware detection across different platforms. It has been demonstrated on microcontrollers [6, 10, 13–16], mobile devices [7, 9, 17–20], as well as on desktop computers [8, 21–24]. Depending on how these power are measured, power-based malware detection can be classified as in-band and out-of-band. In-band detection has been demonstrated in [9, 18] on smart phones. One of the limitations of the in-band detection is that the system is subject to the same threats similar to other software- and hardware-based defenses. This is because an attacker that has compromised the system will be able to access their in-band measurement directly and disable these in-band detectors. Out-of band detection, on the other hand, acquire the power traces outside the system, thus it will be able to detect the malware even if the system is already compromised. It has been demonstrated in SCADA systems [25], software-defined radios [26], and microcontroller units (MCU)[13]. For example, [13] take fine-grain power measurements to track code execution of an MCU. They use an HMM to recover the most likely executed instruction sequence and can detect a single instruction change. However, all of these out-of-band detections are limited to simple embedded microcontroller setups assuming single core or single task in a multicore system.

Aggregation of Measurements. To improve the detection accuracy, the measurements can be aggregated. Depending on the source of measurement, it can be either spatially aggregated or temporal aggregated. For spatial aggregation, [27, 28] form a graph to aggregate enterprise-level network data from multiple spatial sources. For temporal aggregation, there are density-based approaches, which compose a new user-level score from some sample of recent window-based detector scores. This work demonstrates that the density-based detectors can be easily evaded by interleaving benign actions with malicious actions.

[29] employ a bag-of-words scheme, where individual time windows are treated as words and larger time-to-detect windows as bags. An alarm is raised if a bag appears excessively anomalous. Similarly, [30] use a larger sliding window over smaller window samples. Our experimental results demonstrate that such density-based approaches can be evaded in the power domain by interleaving benign actions with malicious actions.

In contrast, our lifecycle-aware detector leverages temporal aggregation and an HMM to capture the causal relationships between different measurements, rather than relying on simple aggregation.

MITRE ATT&CK and Attack Lifecyles. The MITRE ATT&CK framework is a valuable tool for characterizing attack lifecycles. Many attacks can be mapped to ATT&CK [31–33] using machine learning techniques. General knowledge bases of attack lifecycles, including those for generic and IoT malware, are provided by [12, 34]. The framework has also been applied to vulnerability identification [33] and risk assessment [32].

Earlier work on lifecycle-aware malware classification and analysis was introduced in the context of network intrusion detection [35]. Additionally, it has been used for taint-tracking in Android applications [36]. Several SoK and measurement papers have characterized the applications, use cases, and limitations of ATT&CK [37–39].

Prior works also utilize finite state machines and HMMs to detect sequences of attack or intrusion behavior [40–44]. However, these detectors rely on explicit signals, such as system call traces or host states, rather than power side-channels. Furthermore, their detection scope is narrower, focusing on specific intrusion patterns rather than generalized attack lifecycles.

To our knowledge, no prior work has proposed or analyzed a detection approach that leverages the MITRE ATT&CK framework in the context of out-of-band power-based detectors. This paper is the first to systematically design and evaluate a lifecycle-aware malware detection approach for this context.

III. System Overview

We first describe our threat model and deployment scenario and then provide an overview of our LAPD framework.

A. Threat Model and Deployment Scenario

We consider a scenario where a multi-core system concurrently runs multiple potentially multi-threaded embedded applications, such as autonomous drones [45]. At the beginning, the system is not compromised. The goal of the attacker is to deploy a malware that compromises the system through a chain of actions. For example, the attacker may do an SSH access to gain initial access of the system, then the malware can do a priviledge escalation to gain access to critical information, then perform exfiltration.

LAPD aims to detect such attacks, using an out-of-band detector, physically separated from the target system, sampling power exclusively from external, board-level power supply rails. This is similar to the power-based malware detector setup in existing works such as [11], and it introduced no additional overhead. LAPD can operate as software on a microcontroller or a Raspberry Pi-class device, equipped with a current sensing

and analog-to-digital signal processing and conversion front-end.

As LAPD operates out-of-band, its power readings cannot be tampered with by any attacker within the monitored system. We further assume that LAPD is deployed on a tamper-resistant device. Apart from that, we model a strong adversary with both network and physical access to the victim embedded device. The attacker is assumed to be aware of LAPD's presence and actively motivated to evade it.

LAPD leverages attack lifecycles and assumes that a security practitioner can configure it to model relevant scenarios. In our evaluations, we configure LAPD to model attack lifecycles based on the MITRE ATT&CK matrix. We assume this matrix accurately reflects real-world attack scenarios.

Specifically, we treat the generalized *stages* of the MITRE matrix as covering common malicious *actions* that adversaries may employ. For instance, malware cannot simply appear on a victim device. An attacker must first execute an *injection action* to introduce new malware or an *activation action* to trigger an existing trojan, both of which belong to the *initial access stage* defined by the MITRE model.

We emphasize that the LAPD framework is not restricted to the MITRE model; extensions, modifications, or alternative lifecycle models can be seamlessly incorporated.

B. LAPD Framework Overview

Fig. 1 provides an overview of the LAPD framework architecture. LAPD consists of two main parts: the front-end black-box window detector, and the back-end lifecyle-aware detector. Power samples extracted from the target system first go through a sliding window extraction, and a black-box front-end detector is used to score and filter out potential malicious windows of power samples, which are then forwarded to the lifecycle-aware detector. LAPD requires very little overhead compared to a traditional out-of-band power-detector implementation. Target device, power sampling, and black-box detector all incur no overheads. In addition to Lifecycle-Aware Detector, all that is needed is to pipe the front-end scores to a window-forwarding function feeding the LA Detector. Afterwards, an end-user can simply monitor the top level threat score emited by the LA detector.

C. Black-Box Front-End Detector

We do not restrict the specific implementation of the front-end detector. It is treated as a black box, which allows the framework to incorporate any existing detector from prior works, including newly developed ones. In this paper, we use a detector that uses ensembles of one-class classifiers (OCCs), which has demonstrated superior performance in complex detection contexts involving parallel execution of multithreaded software on advanced hardware. The key idea behind this detector is to train each OCC on known benign combinations of executing applications. If no OCC identifies a given window as benign, the window is classified as malicious. This approach eliminates the need for malware-specific training, enabling the detection of zero-day attacks.

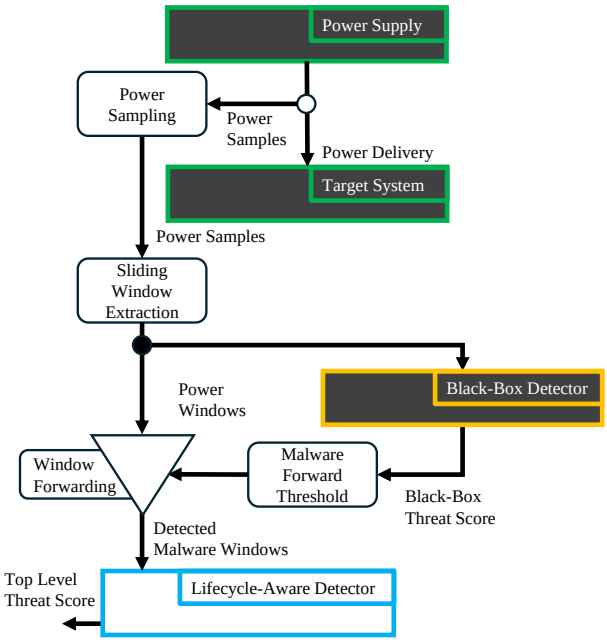


Fig. 1: LAPD framework. Power sample windows are first analyzed by a traditional black-box power-based detector. Windows identified as malicious are then passed to the lifecycle-aware detector, which generates a continuous threat score reflecting the entire power trace.

IV. LAPD Backend Architecture

The front-end Black-Box Detector takes sliding window samples from the input power trace and outputs a maliciousness score for each window. The maliciousness score is used to determine whether to forward a window to the backend LA detector for further investigation. The output of LAPD is a threat score that estimates the likelihood of a sequence of windows representing a valid attack according to a lifecycle model.

Figure 2 provides a detailed overview of the LAPD backend, consisting of the Action Classifier, the Sequence Processor, the Hidden Markov Model (HMM), and the Threat Score Function. The Action Classifier categorizes malicious windows into common actions associated with an attack lifecycle. The Sequence Processor then processes these action classifications, filtering them into a sequence of stage emissions. Next, the HMM analyzes the stage emission sequence and outputs a threat score reflecting its confidence that the input sequence corresponds to a real attack. Finally, the Threat Score Function integrates window classifications, processed emission sequences, and the HMM score to produce a continuous, trace-level threat score.

At its core, LAPD incorporates a classifier trained on common attack actions and an HMM with encoded attack flows through its start probabilities and transition weights. The HMM assigns a high likelihood score when a sequence of actions aligns with an expected attack flow and a lower score

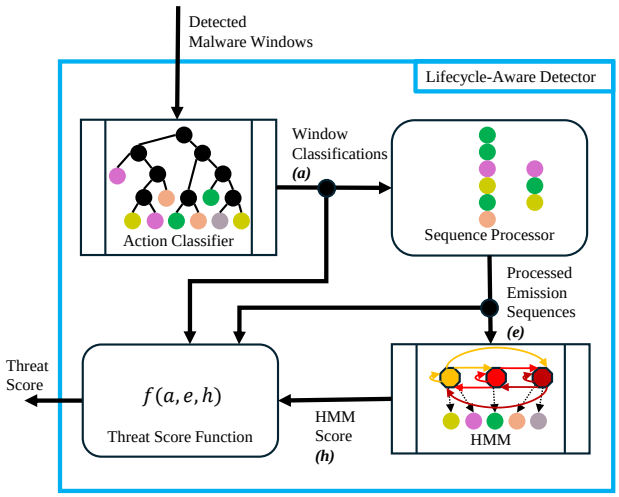


Fig. 2: Detailed architecture of LAPD detector. Key compoents include an Action Classifier, Sequence Processor, HMM, and Threat Score Function.

when the sequence deviates from the attack lifecycle. This lifecycle-aware detector offers three key advantages over prior state-of-the-art detectors:

1) Sequence-awareness of an attack (e.g. cannot run exfiltraion before initial access) is encoded into model weights to reduce false positives.
2) Detection-awareness of attack actions is incorporated to reduce false negatives, allowing the HMM to infer that emission sequences with some missing actions are not necessarily less malicious.
3) Time-agnostic analysis eliminates reliance on attack speed or predefined time ranges for composing window predictions, ensuring that an attacker stalling or spreading actions over long periods of benign behavior does not lower the chance of detection.

A. Action Classifier

The first component within the LAPD backend is the Action Classifier, which categorizes windows labeled as malicious by the front-end Black-Box detector into common actions within an attack lifecycle. In our evaluations, we train an XGBoost tree classifier on common actions defined by the MITRE ATT&CK matrix, as listed in Table I.

The Action Classifier is a supervised model trained exclusively on malicious actions. Exploring semi-supervised classifiers capable of generalizing to other actions is left for future work. However, note that the front-end malware detector is trained solely on benign data, and the HMM weights are set to provide robustness against misclassifications by the Action Classifier. Furthermore, when classification confidence does meet a user-set threshold, the classification is not forwarded to the Sequence Analyzer. We found a threshold value $C_{\text{clf\_conf}} = 0.85$ to give reasonable results.

TABLE I: Attack Actions considered in this work. Actions commonly observed in attack lifecycles form a core component of the LAPD backend. These actions are used to train the Action Classifier and are modeled as hidden state emissions in the HMM formulation.

Action	***Description***	***Abbr.***
SSH	SSH's into system	ssh
UFW	UFW configuration chang	ufw
External Device	Insert external device into system	fd
Enum Configuration	Gather config files for common applications and services	enum cfg
Enum System	Gather system information	enum sys
Enum Network	Gather network information	enum net
Meltdown	Microarchitectural attack	m
L1 Covert-Channel	Microarchitectural attack	cc
Spectre	Microarchitectural attack	s
Change Permissions	Change permissions of files/logs	perm
Rootkit	Execute a rootkit	rk
Delete Bash History	Delete Bash History	hist
Create Account	Create a privileged account	user
SCP	Exfiltrate data via SCP	scp

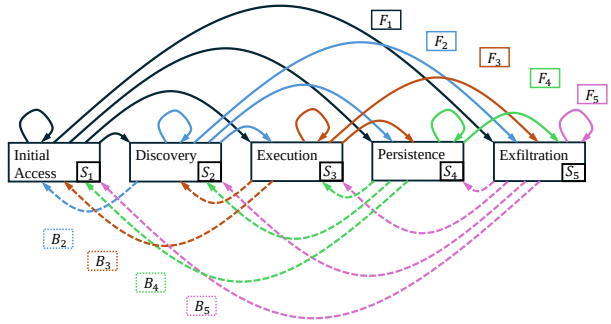
B. Sequence Analyzer

The Sequence Analyzer processes actions classified by the Action Classifier, treating them as emissions for the HMM. First, to reduce noise of misclassifications, the Sequence Analyzer performs a filtering function on the action sequence, deleting subsequences of identical actions if they are under a user set length. In our experiments, we used the length $C_{\text{uniform\_subseq}}$ to be 2. Next, to better align with the problem framing, the Sequence Processor replaces consecutive repeated emissions with a single emission, as they are unlikely to provide additional insight. Additionally, the Sequence Analyzer computes and stores emission subsequences, which are later utilized by the Trace Memory function described in Section IV-D.

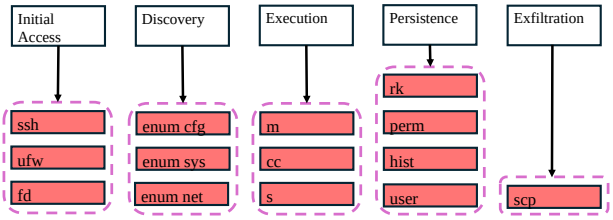
C. HMM

We model the sequence of stages in an attack lifecycle using a hidden Markov model (HMM). Lifecycle stages are defined as hidden states, and actions are grouped as emissions corresponding to their respective stages. Since each stages in an attack may emit several actions which may not be unique, the state and the corresponding probability cannot be directly inferred by looking at the actions. For example, Figure 3b shows, each state can correspond to several actions. However, HMM is a tool for inferring the states in this case. An HMM provides a natural and intuitive framework for attack lifecycle detection. While the lifecycle-aware detector cannot directly observe an attacker's flow, it can infer the sequence based on actions detected by the front-end classifier.

The HMM formulation is inherently tolerant to missed detections. Weights can be configured to reduce, but not eliminate, the warning signal in the event of a missed detection. Additionally, the framework does not require training on every possible action. Undetected actions are simply treated as absent. As long as the overall sequence aligns with an attack lifecycle, the HMM-based lifecycle-aware detector remains



(a) HMM states, start probabilities, and transition probabilities. Start probabilities are labeled S_i, forward transition probabilities as F_i, and backward transition probabilities as B_i.



(b) Action-to-emission and emission-to-state mapping. Actions are grouped as a single emission exclusive to each state. Emission weights are all set to 1.

Fig. 3: Lifecycle-aware HMM with weights based on ATT&CK knowledge.

TABLE II: Attack lifecycle stages. The most critical and or common stages inspired by ATT&CK and used for our LA detector.

Stage	***Description***
Initial Access	Gain initial attacker foothold or malware delivery within a victim device.
Discovery	Actively or passively gather information that can be used to support or enhance the attack.
Execution	Utilize some exploit to execute adversary-controlled code on victim device.
Persistence	Perform post-exploit actions to escalate privileges, evade detection, and persist within a victim.
Exfiltration	A common key purpose of an attack; to steal confidential data, typically over a network.

robust against detection gaps. This robustness is particularly important for noisy detectors, which are common in power-based detection systems. While both finite state machines and HMMs have been used to detect specific exploits by monitoring system calls, network information, and machine state [40–44], noisy power signals and generalized attack lifecycles make full detection coverage and performance unrealistic. Recent studies demonstrate that even enterprise-wide SIEM tools, which claim broad coverage of the ATT&CK matrix, often fail to deliver on these claims [39].

Fig. 3a visualizes the states, start probabilities, and state transition probabilities of the HMM used in our work. In our implementation, we condensed the MITRE ATT&CK matrix to focus on the most important stages present in almost all attacks (Table II), and set the number of hidden HMM states to $n = 5$. Start probabilities, denoted as S_i for each state i, are configured based on the end-user's confidence in the components preceding the HMM to detect the Initial Access stage. Given the starting state probability S_1, the subsequent start probabilities can be set as follows:

$$S_i = (1 - s_1) \cdot \frac{(n-i) \cdot c_{\text{start\_weight}}}{\sum_{i=2}^{n} (n-i) \cdot c_{\text{start\_weight}}}, i \in [2, n]$$

We used $c_{\text{start\_weight}} = 1.5$ to control the start weight ratio among states. An observed emission sequence beginning at the execution stage implies that the preceding components may have missed emissions from the *Initial Access* and *Discovery* stages. Therefore, later stages are assigned lower start weights compared to earlier stages.

The state transitions of the HMM are illustrated in Fig. 3a, where forward transitions are shown as solid edges and backward transitions as dotted edges. Self-edges are treated as forward transitions, reflecting cases where an attacker may attempt different actions within the same stage to advance through the attack lifecycle.

The transition matrix produces a fully connected HMM. While we make no assumptions about the relative likelihood of specific attack flows, we consider forward transitions more probable and consistent with the attack lifecycle than backward transitions. Therefore, all forward outgoing edges within a stage are weighted equally, as are all backward outgoing edges. We prioritize forward transitions over backward transitions using $c_{\text{FB\_ratio}} = 2$ and compute forward F_i and backward B_i edge weights for each state using:

$$F_i = \frac{c_{\text{FB\_ratio}}}{(n+1-i) \cdot c_{\text{FB\_ratio}} + (i-1)}, i \in [1, n]$$

$$B_i = \frac{1}{(n+1-i) \cdot c_{\text{FB\_ratio}} + (i-1)}, i \in [2, n]$$

Fig. 3b illustrates the action-to-emission and emission-to-hidden-state mappings in our HMM. Actions associated with a given stage are grouped within purple dashed boxes and mapped to a single emission for that stage. Initially, we considered setting emission weights based on the detection performance of preceding components and introducing a dummy emission to account for cases where no action was detected. However, we later observed that, from an external perspective, it is impossible to distinguish between a missed emission and a skipped transition. For example, observing an emission exclusive to $state_1$ followed by an emission exclusive to $state_3$ does not reveal whether an emission from $state_2$ was missed or whether the hidden process transitioned directly from $state_1$ to $state_3$. To address this ambiguity, we simplify our model by assigning equal weights (set to 1) for all emissions. In other words, the likelihood of missing an emission from $state_2$ and the likelihood of a direct transition from $state_1$ to $state_3$ are combined into the transition weight between $state_1$ and $state_3$.

The HMM returns a log likelihood from a given emission sequence e of length T. We normalize this likelihood by the sequence length and define the HMM score as:

$$\mathrm{f}_{\text{HMM}} = \exp\left[\frac{1}{T}\log\left(\sum_{X} P(e|X)P(X)\right)\right]$$

where X represents all possible hidden node sequences, which correspond to sequences of attack stages. Note that Fig. 3 illustrates one implementation of a lifecycle-aware HMM based on the ATT&CK framework. Our approach is general and can be adapted to other attack scenarios by modifying the HMM to suit specific deployment and threat models.

D. Threat Score Function

The HMM score alone is insufficient for malware detection, as it is overly sensitive to the length of the emission sequence. Additionally, a subsequence's HMM score is always greater than or equal to that of its supersequence, regardless of the HMM weight configuration. In other words, for a fixed-length emission sequence e of length T, there is no sequence e^+ composed of T identical emissions and one subsequent emission such that:

$$\mathrm{f}_{\text{HMM}}(e) < \mathrm{f}_{\text{HMM}}(e^+)$$

To mitigate these limitations, we combine the HMM score with two intuitive penalties to generate a top-level threat score.

First is the *Duration Penalty*. A trace with more detected actions should be considered more suspicious. Thus, with an emission sequence e of length T emitted by the Action Classifier and a duration penalty constant C_{duration} (set to $C_{\text{duration}} = 500$ in our work), we define a duration score $\mathrm{f}_{\text{duration}}$ as:

$$\mathrm{f}_{\text{duration}}(e) = len(e) \cdot C_{\text{duration}}$$

Next, we incorporate a *Forward Propagation Penalty*. For a given emission sequence, we evaluate how many forward transitions between hidden states were used to generate it. In our HMM framework, more forward transitions correspond to greater progression through an attack lifecycle, which is inherently more dangerous. To account for this, we introduce a penalty. Using a stage propagation constant $C_{\text{propagation}} = 0.4$ and the unique inferred forward transition count $u()$ for an emission sequence e processed by the Sequence Analyzer, we define a propagation score $\mathrm{f}_{\text{propagation}}$ as:

$$\mathrm{f}_{\text{propagation}}(e) = u(e) \cdot C_{\text{propagation}}$$

Finally, a simple top-level threat score combines the penalties and HMM score as:

$$\mathrm{f}_{\text{top}}(e) = \mathrm{f}_{\text{HMM}}(e) + \mathrm{f}_{\text{duration}}(e) + \mathrm{f}_{\text{propagation}}(e)$$

However, motivated attackers may execute nonsensical attack flows with backward edges to intentionally confuse the LA detector. We leverage a memory function to store worst-case emission subsequences in order to enhance robustness against such scenarios. The Sequence Analyzer includes a memory function to generate and store subsequences e' of e. We then define the LAPD output as the worst-case subsequence score within a trace:

$$\mathrm{f}_{\text{LAPD}}(e) = \arg\max_{e' \subseteq e}[f_{\text{top}}(e')]$$

V. Experiments and Results

We envision our LAPD framework to be deployed in complex cyber-physical systems. We therefore evaluate LAPD on an embedded multi-core development platform running drone-based workloads. We assess multiple LAPD variants across varying lifecycle flows, attacker capabilities, and detector avoidance modifications. Additionally, we investigate notable findings and analyze their underlying causes.

A. Experimental Setup

Our trace collection setup uses a Portwell PCOM-C700 Type VII carrier board equipped 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, with readings sampled at 2 kHz. Traces were expanded into sliding windows of size 1000 with a stride of 100. A bash script running on a separate desktop PC automated data collection, ensuring the target board operated in a specified state before triggering data sampling on the oscilloscope. Power traces for all modes were collected and split into 50-50 training and testing datasets.

We utilized three benign applications representing typical drone tasks: a SHA-3 implementation from the Extended Keccak Code Package [46], a face detection application using the OpenCV library with a video benchmark [47], and an autonomous drone package delivery benchmark from MAVBench [45]. For *Execution Stage* malware, we selected Meltdown [48], Spectre [49], and L1 covert-channel [50] microarchitectural attacks as best-case scenarios for detectability due to their noisy power signatures. Additionally, we executed various ATT&CK actions listed in Table I, using bash scripts and the Metasploit Framework [51].

For our LAPD framework implementation, we use an ensemble-based detector from [11] as the Black-Box Detector. In preprocessing for the Black-Box Detector and Action Classifier, each sliding window was transformed into a feature vector comprised of basic statistical features as well as Bag-of-Words (BoW) features[52]. The Black-Box outputs were normalized such that benign outputs formed a standard normal distribution, with lower scores indicating more malicious behavior. We set $C_{\text{forward\_threshold}} = -2$ to determine window forwarding. The Action Classifier is implemented as a XGBoost Classifier [53] with default hyperparameters. The classes are specified in Table I. Other parameters for the Action Classifier, Sequence Analyzer, HMM, and Threat Score were configured as specified in Section IV-D.

We tested multiple LAPD variants, each distinguished by unique Threat Score Functions, as described in Section IV-D.

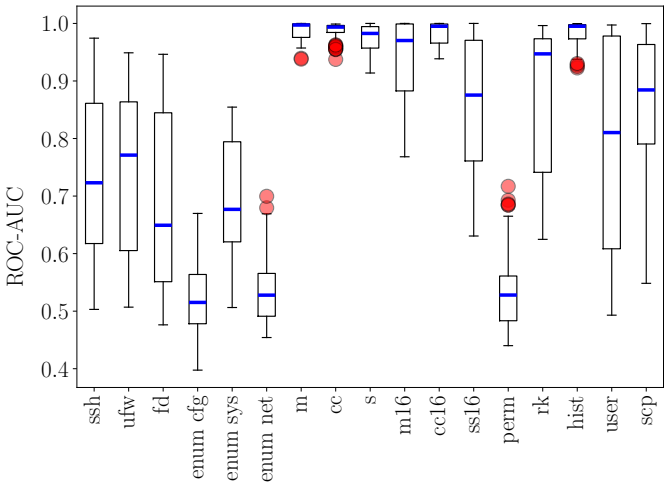


Fig. 4: Black-Box Front-End Detector results against lifecycle actions. Some good results are achieved against the execution stage actions, but performance greatly suffers in other stages. Furthermore, when Meltdown and Spectre are slowed down by 16x (*s16* and *m16*), they can avoid detection.

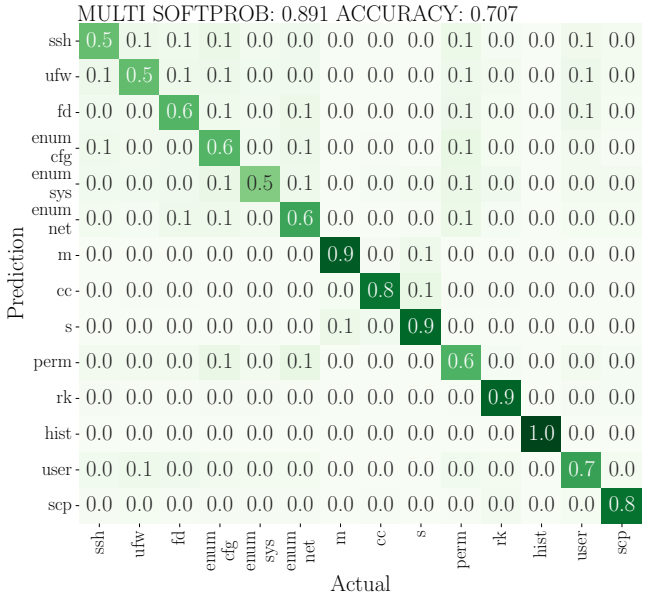


Fig. 5: Normalized confusion matrix of action classifier.

LA-B represents the most basic LA detector, which uses a top-level threat score:

$$\mathrm{f}_{\text{LA-B}}(e) = \mathrm{f}_{\text{HMM}}(e)$$

LA-D incorporates a Duration Penalty, producing:

$$\mathrm{f}_{\text{LA-D}}(e) = \mathrm{f}_{\text{HMM}}(e) + f_{\text{duration}}(e)$$

LA-P applies the Propagation Penalty and leverages the Sequence Analyzer's worst-case subsequence memory function:

$$\mathrm{f}_{\text{LA-P}}(e) = \arg\max_{e'}[f_{\text{HMM}}(e') + f_{\text{propagation}}(e')]$$

Finally, LA-PD employs the full Threat Score Function, f_{LAPD}, defined in Section IV-D.

We compare LAPD against a baseline approach, where the front-end detector is combined with a density-based detector commonly used in prior intrusion detection studies [29, 30]. This baseline density-based detector maintains a buffer of size d (unless stated otherwise, we use $d = 40$ in our experiments) to sum a series of scores, b, produced by the Black-Box Detector. It outputs the following top-level threat score:

$$\mathrm{f}_{\text{Density}} = \arg\max_{j}\left(\frac{1}{d}\sum_{j}^{j+d} b_j\right)$$

B. Black-Box Front-End Evaluation

We first evaluate a standalone Black-Box Front-End Detector on all actions of our described attack lifecycle. We also test against evasive attacks, which we implemented by slowing down the execution rates of Meltdown, L1 Covert-Channel, and Spectre by 16x (*m16*, *cc16* and *s16*)

Fig. 4 shows ROC-AUC scores across all combinations of benign and infected states for each type of action as mentioned in Section V-A. Results show that despite good performance against actions in the execution stage, the performance across the rest of the lifecycle suffers greatly. Furthermore, we validate that slowed-down, evasive attacks can easily avoid the front-end detector. Thus, detection against attack flows cannot be guaranteed by a Black-Box Front-End Detector alone.

C. Action Classifier Evaluation

We present the normalized confusion matrix of the Action Classifier in Figure 5. Although the total accuracy score is not particularly high, later full-system evaluations of LAPD demonstrate that an accuracy of 70% is sufficient for effective performance. This can be attributed to the fact that subsequent components of LAPD are designed to tolerate noisy or lossy results from the Black-Box Detector and Action Classifier.

We also highlight an important tradeoff in the Action Classification design: Increasing the number of classes improves lifecycle coverage but comes at the cost of reduced accuracy. While multiclass classifiers with hundreds of classes exist, they typically operate on image data rather than noisy power traces. Despite this limitation, we anticipate that LAPD performance will improve with a more accurate Action Classifier.

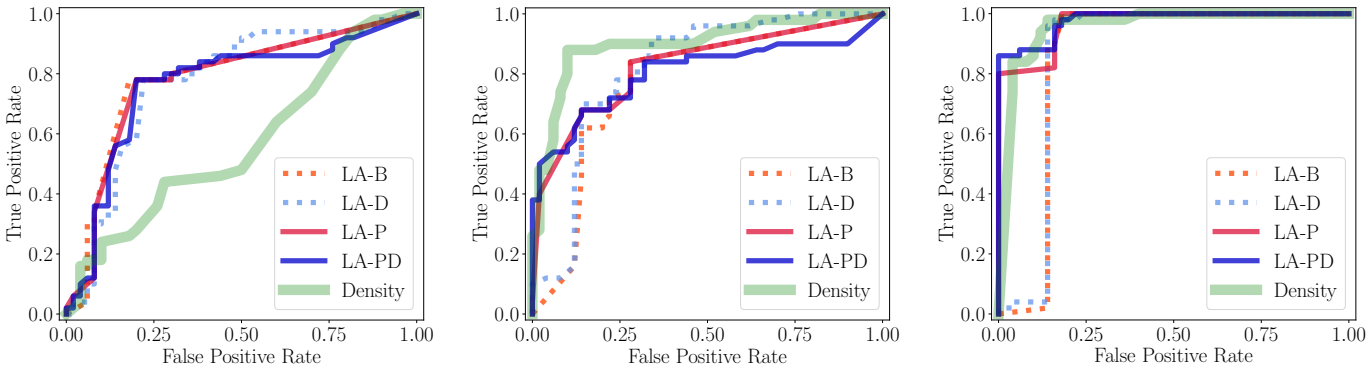
D. Attack Lifecycle Evaluation

We evaluate the framework against multiple potential flows through the attack lifecycle. The top-level flows are summarized in Table III. The *End-to-End* flow represents a full lifecycle execution and is expected to be the easiest to detect. Next, we test a *Focused* flow, which models an insider threat bypassing the discovery stage. Finally, we assess a *Barebones* flow with the minimal sequence required to execute an attack.

We also model attackers with varying capabilities and adapt malware usage within the execution stage accordingly. For a *Baseline Attack*, the execution stage contains only known

TABLE III: Modeled attack flows.

Name	Stages	Motivation
End-to-End	Initial Access, Discovery, Execution, Persistence, Exfiltration	Full attack lifecycle
Focused	Initial Access, Execution, Persistence, Exfiltration	No Discovery needed
Barebones	Initial Access, Execution, Exfiltration	Minimum required stages



(a) Stalled Baseline Barebones (b) Backedge Zero-Day Focused traces (c) Backedge Baseline Focused traces

Fig. 6: ROC curves of interesting detection results as described in Table IV.

malware. For an *Evasive Attack*, the execution stage includes power-evasive malware [7]. To generate traces, we used attacks intentionally slowed down to reduce their power signature. To model *Zero-day Attacks*, we created traces that completely omit the execution stage.

Additionally, we consider attackers aware of density-based or lifecycle-aware detectors and capable of modifying their attack lifecycles to evade detection. *Stalled* traces introduce long periods of benign activity to mislead density-based detectors. *Backedge* traces add extraneous actions resembling backward edges in lifecycle flows to confuse lifecycle-aware detectors. Finally, we evaluate traces combining both *stalled* and *backedge* modifications.

For full-system evaluation, each detector is provided with identical, randomly generated benign and malicious window sequences to distinguish. Malicious sequences follow the attack flows described in Table III. For each evaluated attack flow, we sample 50 malicious and 50 benign sequences. Actions are randomly selected within each stage, with all actions from Table I included in the sampling process. We define a benign state as a unique combination of executing benign applications, and an infected state as one where malware executes in parallel with benign applications. Both benign and infected states in trace generation are randomly selected.

We evaluate lifecycle awareness by varying lifecycle flows, attacker capabilities, and detector avoidance modifications. The complete ROC-AUC results are presented in Table IV, with the best scores for each trace specification noted in bold. A key observation is that the Density detector is not consistently outperformed. In many cases, it achieves the highest ROC-AUC, particularly for traces without *Stalled* modifications. However, even in these cases, its performance is often only marginally better than that of the LAPD detectors. Another observation is that LA-B rarely achieves the best performance and, when it does, only slightly outperforms other LAPD detectors. This reinforces the value of the Threat Score Function and its HMM extensions. While LA-P seldom performs best, in combination with LA-D, the full LA-PD frequently outperforms others, suggesting that the Duration Penalty remains valuable. The LA-D function also demonstrates standalone benefits, especially for traces with *Stalled* and *Backedge Stalled* modifications, where it often achieves the best performance. One final observation is that LAPD typically performs better against *Barebones* flows than against *Focused* flows. This seems counterintuitive, as the *Focused* flow more closely resembles the full attack lifecycle flow that the HMM is trained on. This warrants further investigation, but a potential cause could be the Action Classifier producing unreliable classifications on actions in the persistence stage, leading to additional detection issues later in the LAPD architecture.

E. ROC Curve Analysis

We further analyze the results by examining trace specifications from Table III and their corresponding ROC curves in Figure 6. Figure 6a illustrates a best-case scenario for LA detectors, which achieve a significantly better true-positive/false-positive tradeoff than the Density detector for *Stalled Baseline Barebones* traces. In this case, the density detector's tradeoff results in an AUC close to 0.5, equivalent to random guessing.

Figure 6b highlights a scenario where the Density detector outperforms the LA detectors. While the density detector performs better, the curves do not indicate dramatic outperformance or suggest any major underlying issue. For *Backedge Zero-Day Focused* traces, which include only the initial access, persistence, and exfiltration stages, LA detectors without Propagation Penalty and worst-case memory exhibit a sharp drop-off around 0.2 FPR. This suggests a subset

TABLE IV: ROC-AUC of detectors for different attack flows and variants.

Trace Specification	Density	LA-B	LA-D	LA-P	LA-PD
Baseline Attack					
End-to-End	**0.988**	0.947	0.955	0.963	0.981
Focused	**0.975**	0.955	0.971	0.963	0.969
Barebones	0.981	0.959	0.961	0.982	**0.988**
Evasive Attack					
End-to-End	**0.978**	0.818	0.887	0.870	0.909
Focused	0.919	0.876	**0.920**	0.877	0.887
Barebones	**0.970**	0.848	0.926	0.858	0.906
Zero-Day Attack					
End-to-End	**0.922**	0.838	0.857	0.850	0.883
Focused	**0.935**	0.864	0.923	0.877	0.908
Barebones	0.877	0.804	**0.885**	0.805	0.859
(Default Average)	0.949	0.879	0.920	0.894	0.921
Stalled Baseline Attack					
End-to-End	0.698	0.780	0.792	**0.806**	0.770
Focused	0.591	0.736	0.718	**0.740**	0.711
Barebones	0.567	**0.787**	0.774	0.781	0.765
Stalled Evasive Attack					
End-to-End	0.519	0.600	0.570	**0.638**	0.586
Focused	0.495	0.632	0.644	0.648	**0.678**
Barebones	0.489	0.629	**0.645**	0.632	0.609
Stalled Zero-Day Attack					
End-to-End	0.493	0.673	0.680	0.674	**0.696**
Focused	0.530	**0.669**	0.658	0.668	0.603
Barebones	**0.614**	0.551	0.611	0.559	0.561
(Stalled Average)	0.555	0.673	0.677	0.683	0.664
Backedge Baseline Attack					
End-to-End	0.970	0.901	0.910	0.973	**0.982**
Focused	0.958	0.858	0.863	0.969	**0.978**
Barebones	0.988	0.939	0.952	0.978	**0.992**
Backedge Evasive Attack					
End-to-End	0.972	0.796	0.838	0.939	**0.973**
Focused	**0.953**	0.864	0.876	0.912	0.937
Barebones	**0.986**	0.904	0.937	0.937	0.948
Backedge Zero-Day Attack					
End-to-End	**0.959**	0.894	0.924	0.922	0.943
Focused	**0.903**	0.772	0.817	0.827	0.805
Barebones	**0.922**	0.840	0.899	0.857	0.872
(Backedge Average)	0.957	0.863	0.891	0.924	0.937
Backedge Stalled Baseline Attack					
End-to-End	0.565	0.763	0.715	**0.823**	0.804
Focused	0.569	0.728	0.716	**0.774**	0.745
Barebones	0.681	0.803	0.783	**0.826**	**0.826**
Backedge Stalled Evasive Attack					
End-to-End	0.504	0.708	0.664	**0.712**	0.667
Focused	0.525	0.706	0.664	**0.735**	0.706
Barebones	0.484	**0.680**	0.610	**0.680**	0.634
Backedge Stalled Zero-Day Attack					
End-to-End	0.523	0.697	0.653	**0.724**	0.693
Focused	0.518	0.662	0.621	**0.671**	0.606
Barebones	0.556	0.679	0.658	0.682	**0.685**
(Backedge Stalled) Average	0.547	0.714	0.676	0.736	0.707
(Global Average)	0.752	0.782	0.791	0.809	0.807

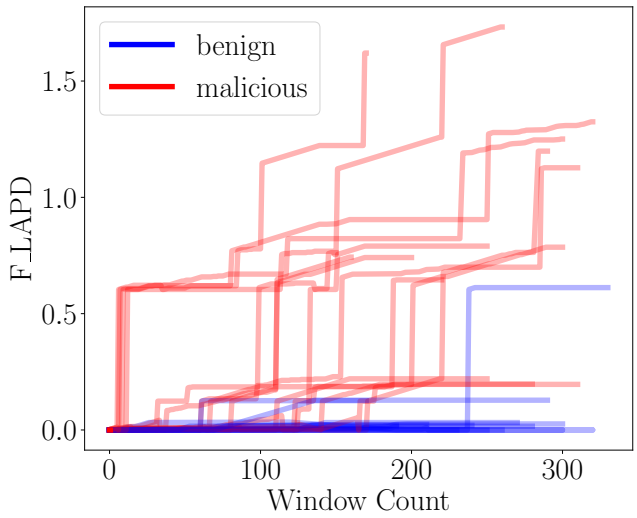


Fig. 7: Scores of benign and malicious trace streamed across windows.

of benign traces within this specification consistently scores above the lowest-scored malicious traces. A similar trend appears in Figure 6c, which examines the *Backedge Baseline Focused* trace specification. In several cases, LA detectors with Propagation Penalty and memory outperform those without. Figures 6a-6c suggest that the Propagation Penalty and worst-case subsequence scoring improve performance by enabling LA detectors to achieve 0 FPR at higher TPR levels.

F. Streaming Score Analysis

The lifecycle detector is time-agnostic and does not require tuning a window length, unlike the a density-based detector. This capability is demonstrated when streaming f_{LAPD} across input windows as shown in Fig.7. We observe that benign traces rarely elevate their threat scores, whereas malicious traces cause increasing scores as they progress through the attack lifecycle.

VI. Summary and Conclusions

In this paper, we present LAPD, a lifecycle-aware power-based malware detector framework that models the sequential stages of real-world attacks, including prime, execute, and post-exploit actions. We leverage hidden Markov models (HMMs) to capture signals across lifecycle stages. LAPD is evaluated on multicore processors with realistic workloads using board-level power measurements. Compared to the baseline method, LAPD achieves a ROC-AUC score of 0.98. Moreover, it achieves a score of 0.70 on attacks employing stalling strategies, which the baseline detector fails to detect.

These results demonstrate that lifecycle-aware detection is a powerful approach for power-based malware detection. We believe lifecycle awareness can extend to malware detection using other signals, such as API calls or microarchitectural timing characteristics. Additionally, emerging model architectures, such as transformers, show promising capabilities for

modeling event sequences beyond HMMs. We plan to explore these alternative signals and architectures in future work.

References

[1] M. Lipp *et al.*, "Platypus: Software-based power side-channel attacks on x86," in *Symposium on Security and Privacy (SP)*, IEEE, 2021.

[2] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side—channel (s)," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Springer, 2002, pp. 29–45.

[3] A. Golder, D. Das, J. Danial, S. Ghosh, S. Sen, and A. Raychowdhury, "Practical approaches toward deep-learning-based cross-device power side-channel attack," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2720–2733, 2019.

[4] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference (Crypto)*, Springer, 1999.

[5] L. Goubin and J. Patarin, "DES and differential power analysis the "Duplication" method," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Springer, 1999.

[6] S. S. Clark *et al.*, "Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *USENIX Workshop on Health Information Technologies (HealthTech)*, 2013.

[7] S. Wei, A. Aysu, M. Orshansky, A. Gerstlauer, and M. Tiwari, "Using power-anomalies to counter evasive microarchitectural attacks in embedded systems," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2019, pp. 111–120.

[8] R. Bridges *et al.*, "Towards malware detection via cpu power consumption: Data collection design and analytics," in *TrustCom/BigDataSE*, New York, 2018.

[9] H. Kim *et al.*, "Detecting Energy-Greedy Anomalies and Mobile Malware Variants," in *Mobisys*, Seoul, 2008.

[10] G. Zhang, X. Ji, Y. Li, and W. Xu, "Power-based nonintrusive condition monitoring for terminal device in smart grid," *Sensors*, vol. 20, no. 13, p. 3635, 2020.

[11] A. Cathis, G. Li, S. Wei, M. Orshansky, M. Tiwari, and A. Gerstlauer, "Sok paper: Power side-channel malware detection," in *Proceedings of the International Workshop on Hardware and Architectural Support for Security and Privacy*, 2024, pp. 1–9.

[12] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "MITRE ATT&CK: Design and philosophy," Tech. Rep., 2018.

[13] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On code execution tracking via power side-channel," in *Conference on Computer and Communications Security (CCS)*, 2016, pp. 1019–1031.

[14] X. Wang *et al.*, "Deep learning-based classification and anomaly detection of side-channel signals," in *Cyber Sensing*, Orlando, 2018.

[15] C. R. Aguayo González and J. H. Reed, "Power fingerprinting in sdr integrity assessment for security and regulatory compliance," *AICSP*, vol. 69, no. 2, pp. 307–327, 2011.

[16] J. Hernández Jiménez, Q. Chen, J. Nichols, C. Calhoun, and S. Sykes, "Towards a cyber defense framework for scada systems based on power consumption monitoring," in *HICSS*, Waikoloa Village, 2017.

[17] L. Caviglione *et al.*, "Seeing the unseen: Revealing mobile malware hidden communications via energy consumption and artificial intelligence," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 799–810, 2015.

[18] L. Liu, G. Yan, X. Zhang, and S. Chen, "Virusmeter: Preventing your cellphone from spies," in *International Workshop on Recent Advances in Intrusion Detection (RAID)*, Springer, 2009, pp. 244–264.

[19] J. Hoffmann, S. Neumann, and T. Holz, "Mobile malware detection based on energy fingerprints—a dead end?" In *RAID*, Gros Islet, 2013.

[20] B. Dixon, S. Mishra, and J. Pepin, "Time and location power based malicious code detection techniques for smartphones," in *RAID*, Gothemburg, 2014.

[21] M. Almshari, G. Tsaramirsis, A. O. Khadidos, S. M. Buhari, F. Q. Khan, and A. O. Khadidos, "Detection of potentially compromised computer nodes and clusters connected on a smart grid, using power consumption data," *Sensors*, vol. 20, no. 18, p. 5075, 2020.

[22] P. Luckett, J. T. McDonald, W. B. Glisson, R. Benton, J. Dawson, and B. A. Doyle, "Identifying Stealth Malware Using CPU Power Consumption and Learning Algorithms," *Journal of Computer Security*, vol. 26, no. 5, pp. 589–613, 2018.

[23] J. A. Dawson, J. T. McDonald, J. Shropshire, T. R. Andel, P. Luckett, and L. Hively, "Rootkit detection through phase-space analysis of power voltage measurements," in *MALWARE*, Fajardo, 2017.

[24] J. H. Jimenez and K. Goseva-Popstojanova, "Malware detection using power consumption and network traffic data," in *ICDIS*, South Padre Island, 2019.

[25] J. H. Jiménez, Q. (Chen, J. Nichols, C. Calhoun, and S. Sykes, "Towards a cyber defense framework for SCADA systems based on power consumption monitoring.," in *Hawaii International Conference on System Sciences (HICSS)*, 2017, pp. 1–7.

[26] C. R. A. González and J. H. Reed, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance," *Analog Integrated Circuits and Signal Processing*, vol. 69, no. 2, pp. 307–327, 2011.

[27] L. Invernizzi *et al.*, "Nazca: Detecting malware distribution in large-scale networks.," in *Network and Distributed System Security Symposium (NDSS)*, Citeseer, vol. 14, 2014, pp. 23–26.

[28] C. R. Harshaw, R. A. Bridges, M. D. Iannacone, J. W. Reed, and J. R. Goodall, "Graphprints: Towards a graph analytic method for network anomaly detection," in *Cyber and Information Security Research Conference (CISR)*, 2016, pp. 1–4.

[29] M. Kazdagli, V. J. Reddi, and M. Tiwari, "Quantifying and improving the efficiency of hardware-based mobile malware detectors," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–13.

[30] I. Onat and A. Miri, "An intrusion detection system for wireless sensor networks," in *International Conference on Wireless And Mobile Computing, Networking And Communications (WiMob)*, IEEE, vol. 3, 2005, pp. 253–259.

[31] A. Kuppa, L. Aouad, and N.-A. Le-Khac, "Linking cve's to mitre att&ck techniques," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021, pp. 1–12.

[32] M. S. I. Sajid *et al.*, "Soda: A system for cyber deception orchestration and automation," in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 675–689.

[33] C. Hankin, P. Malacaria, *et al.*, "Attack dynamics: An automatic attack graph generation framework based on system topology, capec, cwe, and cve databases," *Computers & Security*, vol. 123, p. 102 938, 2022.

[34] O. Alrawi *et al.*, "The circle of life: A large-scale study of the iot malware lifecycle," in *USENIX Security Symposium (USENIX)*, USENIX Association, 2021.

[35] S. K. Pandey and B. Mehtre, "A lifecycle based approach for malware analysis," in *International Conference on Communication Systems and Network Technologies (CSNT)*, 2014.

[36] S. Arzt *et al.*, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *ACM sigplan notices*, vol. 49, no. 6, pp. 259–269, 2014.

[37] K. Oosthoek and C. Doerr, "Sok: Att&ck techniques and trends in windows malware," in *Security and Privacy in Communication Networks: 15th EAI International Conference, SecureComm 2019, Orlando, FL, USA, October 23-25, 2019, Proceedings, Part I 15*, Springer, 2019, pp. 406–425.

[38] S. Roy, E. Panaousis, C. Noakes, A. Laszka, S. Panda, and G. Loukas, "Sok: The mitre att&ck framework in research and practice," *arXiv preprint arXiv:2304.07411*, 2023.

[39] A. Virkud, M. A. Inam, A. Riddle, J. Liu, G. Wang, and A. Bates, "How does endpoint detection use the {mitre}{att&ck} framework?" In *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 3891–3908.

[40] F. Wilkens, F. Ortmann, S. Haas, M. Vallentin, and M. Fischer, "Multi-stage attack detection via kill chain state machines," in *Proceedings of the 3rd Workshop on Cyber-Security Arms Race*, 2021, pp. 13–24.

[41] R. A. Kemmerer, "Nstat: A model-based real-time network intrusion detection system," *Computer Science Department, University of California, Santa Barbara, Report TRCS97-18, http://www. cs. ucsb. edu/TRs/TRCS97-18. html*, 1997.

[42] W. K. Zegeye, R. A. Dean, and F. Moazzami, "Multi-layer hidden markov model based intrusion detection system," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 265–286, 2018.

[43] J. Byrnes, T. Hoang, N. N. Mehta, and Y. Cheng, "A modern implementation of system call sequence based host-based intrusion detection systems," in *2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 2020, pp. 218–225.

[44] M. Liu, Z. Xue, X. Xu, C. Zhong, and J. Chen, "Host-based intrusion detection system with system calls: Review and future trends," *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[45] B. Boroujerdian *et al.*, "Mavbench: Micro aerial vehicle benchmarking," in *2016 51th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Fukuoka, 2018.

[46] KECCAK Team. "XKCP." https://github.com/XKCP/XKCP, XKCP. (Apr. 5, 2020).

[47] Y. Wong, S. Chen, S. Mau, C. Sanderson, and B. C. Lovell, "Patch-based probabilistic image quality assessment for face selection and improved video-based face recognition," in *CVPR*, Colorado Springs, 2011.

[48] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, vol. 63, no. 6, pp. 46–56, 2020.

[49] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[50] C. Hunger *et al.*, "Understanding Contention-Based Channels and Using Them for Defense," in *HPCA*, Burlingame, 2015.

[51] Rapid7. "Metasploit-framework." https://github.com/rapid7/metasploit-framework. (Apr. 17, 2024).

[52] J. Wang, P. Liu, M. F. She, S. Nahavandi, and A. Kouzani, "Bag-of-words representation for biomedical time series classification," *Biomed. Signal Process. Control.*, vol. 8, no. 6, pp. 634–644, 2013.

[53] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, ser. KDD '16, San Francisco, California, USA, 2016, pp. 785–794.