Using Power-Anomalies to Counter Evasive Micro-Architectural Attacks in Embedded Systems

Shijia Wei[1], Aydin Aysu[2], Michael Orshansky[1], Andreas Gerstlauer[1], and Mohit Tiwari[1]
[1]Department of Electrical and Computer Engineering, The University of Texas at Austin
[2]Department of Electrical and Computer Engineering, North Carolina State University
[1]{*shijiawei, orshansky, gerstl, tiwari*}*@utexas.edu* [2]*aaysu@ncsu.edu*

***Abstract*—High-assurance embedded systems are deployed for decades and expensive to re-certify – hence, each new attack is an unpatchable problem that can only be detected by monitoring *out-of-band* channels such as the system's power trace or electromagnetic emissions. Micro-Architectural attacks, for example, have recently come to prominence since they break all existing software-isolation based security – for example, by hammering memory rows to gain root privileges or by abusing speculative execution and shared hardware to leak secret data. This work is the first to use anomalies in an embedded system's power trace to detect *evasive* micro-architectural attacks. To this end, we introduce *power-mimicking* micro-architectural attacks – including DRAM-rowhammer attacks, side/covert-channel and speculation-driven attacks – to study their evasiveness. We then quantify the operating range of the power-anomalies detector using the Odroid XU3 board – showing that rowhammer attacks *cannot* evade detection while covert channel and speculation-driven attacks can evade detection but are forced to operate at a *36× and 7× lower bandwidth*. Our power-anomaly detector is efficient and can be embedded out-of-band into (e.g.,) programmable batteries. While rowhammer, side-channel, and speculation-driven attack defenses require invasive code- and hardware-changes in general-purpose systems, we show that power-anomalies are a simple and effective defense for embedded systems. Power-anomalies can help future-proof embedded systems against vulnerabilities that are likely to emerge as new hardware like phase-change memories and accelerators become mainstream.**

I. Introduction

Systems embedded in critical infrastructure, like robotics, manufacturing or hospitals, face many evolving threats over their decades of deployment. Such embedded systems have to pass stringent, time-consuming and costly compliance requirements before they can be deployed or upgraded. In many cases, users choose to not upgrade them [1], [2], leaving them vulnerable to threats that emerge post-deployment [3], [4]. As a result, there is considerable interest in placing *out-of-band* monitors that observe physical side-effects of computation to detect malicious activity in benign systems. For example, fine-grained monitoring of electromagnetic (EM) emissions [5]–[7] or power [8]–[11] has been shown to reliably detect tampering of small programs on micro-controllers.

Out-of-band detectors are, however, hard to scale to modern attacks on complex embedded systems such as robots and drones [12]. Benign systems include complex systems-on-chip (SoC) hardware [13] with dynamic behaviors due to advanced cache hierarchies, accelerators, and on-chip networks; running software such as computer vision and distributed control on systems like Linux or Robot Operating Systems (ROS). Furthermore, micro-architectural attacks [14]–[17] break all software defenses without tampering with existing code—instead, they reuse benign instructions with data specifically crafted to abuse speculative execution [16], [17] or DRAM characteristics [15]. Can power analysis observe the 'signal' inherent in an attack through the noise of benign execution? Can *evasive* attacks [18]–[20] hide behind noise? This paper provides *quantitative* answers to these questions.

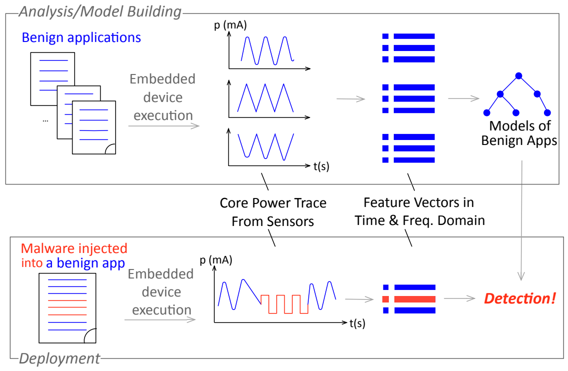

Analysis/Model Building
Benign applications
Embedded device execution
p (mA)
t(s)
Models of Benign Apps
Core Power Trace From Sensors
Feature Vectors in Time & Freq. Domain
Malware injected into a benign app
Embedded device execution
p (mA)
t(s)
Detection!
Deployment

Fig. 1: Overview of power-anomaly detector system.

We demonstrate that power-anomalies can reliably cut through the noise of diverse benign programs to detect micro-architectural attacks in complex embedded systems. First, we have to model evasive attackers – to this end, we introduce the first *power-mimicry* technique where an attacker shapes its power trace to mimic that of a benign program while continuing to execute its malicious task. In doing so, the attacker pays with efficiency (lower channel bandwidth, for example) to evade detection. We then test a range of power-anomaly detectors using benign programs from computer vision to room navigation on real hardware to quantify the *operating range* [21] of our anomaly detector. We find that, surprisingly, rowhammer attacks cannot evade even simple power-anomaly detectors. Further, power-mimicking covert-channels are forced to reduce bandwidth by 36× while speculation-attacks (Spectre) are reduced by 7×. The key takeaway is that, for embedded systems, power-anomaly is a simple, effective alternative to defenses that use new hardware designs (caches [22] and memory controllers [23]) or new OS-techniques (that hide `/proc`). We demonstrate through an FPGA-based prototype that power-anomalies are simple enough to be deployed inside (e.g.) software-defined batteries [24] and smart energy management systems [25].

Fig. 1 describes our approach: security engineers train machine-learning models using benign applications' power draw and deploy the models in off-chip logic. Detection-alerts from each detector are then forwarded to global detectors that use collaborative or ensemble methods with other types of per-device detectors [26]–[29] to weed out false positives and

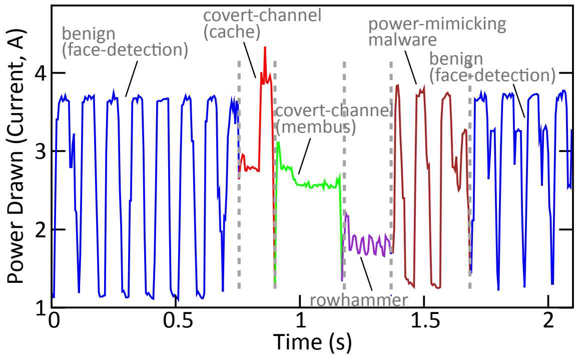


Fig. 2: Two seconds of CPU power consumption of an embedded processor. While baseline micro-architectural attacks generate anomalies on periodic power consumption, an adaptive malware (brown) can mimic benign behavior.

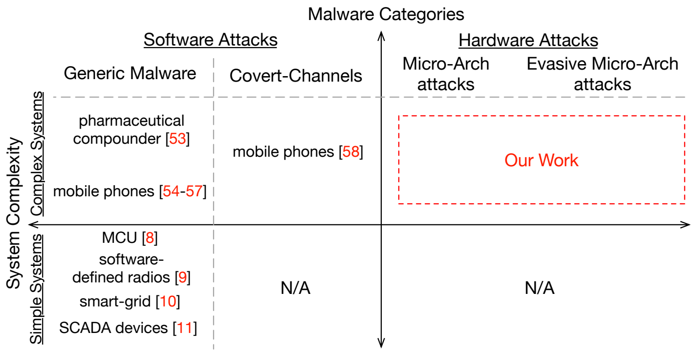


Fig. 3: Existing work in malware detection using *power traces*.

prioritize alerts for attention. The goal of per-device anomaly detectors, including power-anomalies, is not perfect detection with zero false-positives; it is to drive false-positives low enough that global detectors can identify true positives in aggregate. For reference, system-call [29] and hardware performance counter [30]–[32] detectors operate at true-positive rates of 80%–90% and false positives from 1%–20%.

Fig. 2 demonstrates baseline and evasive malware on a real example. It shows the CPU power consumption of an ARM SoC device executing a real-time face detection application [33] as part of a video-surveillance pipeline. A malicious program ('malware') runs on the system for a split second, performing a rowhammer exploit, a baseline covert-channel, and a power-mimicry covert-channel attack, before releasing the system to its normal operations. Despite application phase variations and OS noise, baseline malware generate observable anomalies on the regular power consumption of face detection. In contrast, an evasive power-mimicking attack (brown) replicates the power behavior of face detection and may evade detection.

We construct three power-anomaly detectors to explore detection performance vs. training effort and implementation complexity: (1) a radius-distance based nearest neighbor (RBNN) classifier, (2) a one-class Support Vector Machine (ocSVM), and (3) a recurrent neural networks using Long Short Term Memory (LSTM). We explore the design space of the detectors using a variety of features and parameters. Our results show that, on face detection, while RBNN performs poorly, both ocSVM- and LSTM- based classifiers achieve close to perfect detection performance on *baseline* rowhammer, Spectre, and covert-channel attacks—at 99% true positives, attacks are detected with less than 0.7%, 0.6% and 0.8% false positive rate, respectively.

Against evasive power-mimicking attackers, we show that a rowhammer attack does not succeed after mimicking benign behavior because power-mimicking slows down rowhammer attack's memory accesses to a rate at which it can no longer flip a bit. Covert-channel and Spectre attacks, by contrast, can shape their power signature while continuing to leak information, but doing so forces the channel to lower bandwidths. For example, a mimicry covert-channel attack needs to lower its bandwidth to 2.8% of its baseline—a 36× reduction—while a mimicry Spectre, similarly, loses bandwidth by 7×.

We conduct all experiments on an Odroid-XU3 running embedded Linux detailed in § V-A1. Finally, we prototype the ocSVM based detector on an NIOS-II micro-controller synthesized to a low-cost FPGA. The prototype occupies only 5046 Logic Cells (23% of a small 30-USD Altera Cyclone-IV FPGA) and performs the detection in real-time.

Our key contributions include:

- We propose power-anomaly detectors as a simple, out-of-band, and effective way to detect a range of micro-architectural attacks – including speculative side-channels, covert-channels, and rowhammer – and to drive down effectiveness of evasive attacks.
- We introduce evasive, power-mimicking micro-architectural attacks that replicate the power behavior of benign applications while also executing malicious tasks. We compute the bandwidth cost of mimicry for side- and covert-channels.
- We construct machine learning classifiers that observe *coarse-grained* power traces (i.e., that are available to smart batteries and power-management units), and quantify the detectors' *operating range* – specifically, (a) rowhammer attacks *cannot* evade detection, and (b) covert-channel and Spectre attacks are forced to lose 36× and 7× bandwidth respectively to evade detection in our setup.

In summary, our detection results and hardware implementation suggest that a particularly damaging and sophisticated class of attacks can be subverted through a simple mechanism – power-anomaly detection – that is easy to deploy, does not require expensive upgrades, and can be augmented to handle the slew of attacks that will likely emerge to exploit write-endurance, memory remanence, and similar benign-looking side-effects of new hardware.

II. Background and Related Work

We discuss micro-architectural attacks and prior work in power-based (and other) detectors. We address the gap (Fig. 3) that *power-anomaly detectors have not been studied against micro-architectural attacks – we do so for both baseline and evasive power-mimicking attacks.*

Micro-Architectural Attacks. Malware can abuse micro-architectural behaviors to break *confidentiality* or *integrity*.

Covert-channel and Speculation-driven attacks are two representative types of attacks that break system *confidentiality*. Covert-channel attacks enable two isolated processes running on a system to communicate through a channel that is not designed to transmit information. For micro-architectural covert-channels, the covert communication is achieved by leveraging contentions on shared hardware resources like the last-level cache. An example protocol looks like following: a *busy state* on the shared resource correspond to '1' while an *idle state* refers to '0'. Processes communicate by obtaining/releasing and checking the shared resource's availability. A number of covert-channel attacks have been demonstrated on x86 architectures based on micro-architectural resources such as data and control path [34], caches [35], SMT [36], specialized hardware units [37], and memory buses and controllers [38], [39]. On ARM cores, Lipp *et al.* [14] recently demonstrated the first cache covert-channel.

Unlike covert-channel attacks in which two processes deliberately communicate information, in speculation-driven attacks, e.g. Spectre [16], an attacker tricks the victim into speculatively executing a gadget which reads a (victim's) secret into a micro-architectural structure. The attacker subsequently learns the secret through a micro-architectural side-channel. Many variants have additionally leveraged deffered exceptions [17], speculative stores [40], page-walks [41], and system register reads [42] to break isolation boundaries like hardware enclaves. Various software and hardware mitigation have been proposed, introducing up to 25% performance overhead [42], [43].

Rowhammer, introduced by Kim *et al.* [44], breaks system integrity by modifying the contents of the memory. This vulnerability causes unintentional flips within a row of DRAMs when there are rapid memory accesses to its adjacent rows. Seaborn *et al.* exploited rowhammer vulnerability by flipping critical bits containing kernel privilege information to gain root access to the system [45]. Various attack vectors have been proposed, such as leveraging cache flush instructions [46], non-temporal store instructions [47], and cache eviction strategies [14] to subvert the entire system through the rowhammer attack. Recently, rowhammer attacks have been implemented on ARM embedded devices leveraging DMA interface [15].

Traditional (In-Band) Digital Defenses. Mitigations for micro-architectural attacks is an active area of research. One approach is to identify and eliminate the causes of micro-architectural vulnerabilities. Examples include restricting unprivileged usage of cache-flush instructions [45], removing userland Direct Memory Access (DMA) interface [15], enforcing time-multiplexing or fixed-service memory controllers [39], [48], shared-caches partitioning [35], and disabling hyper-threading [49]. These mitigations, however, require processor architecture modifications or operating system updates. An alternative approach is to detect malicious processes as they execute and raise alerts to a security service. However, traditional detectors like static signature based analysis [50], dynamic behavioral detectors based on permissions, API and system calls [29], [51] do not fully capture micro-architectural behaviors. Hence, they fail to detect the attacks exploiting hardware 'features'. Hardware-based Malware Detectors (HMD) using performance counters [30]–[32], [52] and dedicated hardware units [30] can capture micro-architectural attacks. Our work, which proposes the use of *out-of-band* power-anomaly detectors to detect micro-architectural attacks and quantifies their operating ranges, is complementary to HMD and other in-band detectors.

Anomaly Detection From Power Traces. Fig. 3 summarizes existing work on *power-anomaly* detectors in two dimensions: malware categories and system complexity. As shown in Fig. 3, a number of power-based malware detectors have been investigated on *low-complexity* embedded systems such as 8-bit microcontrollers with simple programs [8]–[11]. These devices do not have micro-architectural features that enable the attacks we analyze. Anomaly detection on more complex embedded devices, such as a pharmaceutical compounder [53] or mobile phones [54]–[58] through power footprints has also been studied for identifying various types of malware. However, to our best knowledge, none of them have ever been evaluated with advanced micro-architectural attacks and attacks that *actively* mimic the power behavior of benign applications.

The work of Caviglione *et al.* is relatively close to our context. It conducts a preliminary study of covert-channel detection based on the energy consumption of a mobile phone [58]. Compared to our work, both the capability of covert-channels and the detection mechanism is relatively low. They implement several *event-based* covert-channels and try to detect them. Attacks are detected, only compared to the *IDLE* state, through the large difference in energy-consumption. By contrast, this work studies and detects, for the first time, evasive covert-channels (and rowhammer attacks) that replicate the power behavior of benign applications. Only with effective attacks and power-mimicking malware introduced in this work, we quantify the operating range of the power-anomaly detectors.

Besides power consumption, electromagnetic radiation of a device collected by a near-field probe is shown to be a promising source to enforce control flow integrity of micro-processors [5]–[7]. In their setting, the fine-grained measurements (of up to seven orders-of-magnitude finer than ours) is critical to detection. The proposed detectors, however, can cost up to several thousands of dollars and is not yet practical for embedded systems. By contrast, our work uses existing *out-of-band* power sensors on embedded devices, which provides mobility, resistance to environmental noise and cost-effectiveness.

Evasive Malware. A line of research, similar in spirit to our work, focuses on evasive malware analysis [19]–[21], [59]. These works target different channels than ours – e.g., hardware performance counters (HPC) [19]–[21] and network traffic inspectors [59] – as well as different attacks (generic malware instead of micro-architectural attacks). Furthermore, we study evasive malware that *actively* mimics benign behaviors by both padding logical NOPs and grafting variants onto benign applications – techniques that prior work has presumed to not be feasible for their case-studies [19]. RHMDs' [20] mimicry is conceptually close to us since the authors reverse-engineer HPC detectors to insert minimal dummy instructions into malware that successfully evade detection. However, we target power-mimicry instead of HPC mimicry; beyond this, we inject entire micro-architectural malware tasks into benign programs, while RHMD inserts dummy instructions into malware.

III. Power-Anomaly Preliminaries

In this section, we discuss the deployment and threat model for a power-anomaly detector (§ III-A), and our implementation of a representative set of micro-architecture attacks (§ III-B).

A. Power-Anomaly Deployment

Constraints. In a long-term deployment, we expect power-anomaly detectors will be required to identify many unknown (post-deployment, '0-day') attacks. Hence we eschew supervised training with known malware and instead train models using *only* the power traces of benign applications (hence the 'anomaly' detector). In addition, detectors will be deployed under limited budget and off-chip pin access to deployed systems; therefore, power-anomaly detectors are constrained to use sensors with coarse-grained sampling (200Hz in our prototype).

Application-Specific Detectors. We evaluate power-anomaly detectors in both *tailored* or *generic* settings. A tailored detector guards one application at a time—any behavior unknown to the detector will be reported as malicious. A generic detector, however, has been trained with execution traces of the complete set of benign applications running in the system. A tailored detector can be more accurate in identifying execution deviations from expected behaviors, while generic detectors provide flexibility in protecting multiple applications at the same time. In high-assurance systems, in which schedule of each application is deterministic, tailored detectors are preferred for lower complexity [60]. However for systems with dynamic scheduling, a generic detector is needed.

Threat Model. Consider a drone that runs untrusted applications separated by an OS; we introduce a detector (e.g., on a smart battery) that monitors CPU power and runs a light-weight anomaly detector on a micro-controller or accelerator. Applications can be tampered with by a malicious employee, an untrusted motherboard component, or be actively malicious [61]—i.e., in the field, an application can run rowhammer to escalate privileges, or use covert-channels to leak secrets from secret (e.g.) path planning or computer vision compartments to an unclassified (e.g.) diagnostics compartment.

The detector aims to ensure that applications act in the field as they did in testing; while the attacker can buy the same device and train mimicry-based evasive attacks. We want to find the operating range of power-anomaly detectors against such evasive adversaries. Note that the attacker has *off-line* access to benign power draw–i.e., they can mimic power, but cannot tamper with or run hamming-distance based (DPA) attacks on the power trace in the field.

While a defense can use software, HPC, or turn off clflush, once the devices is exploited (as we show, using flush-less rowhammer), all software stack becomes untrusted. In contrast, our detector is deployed on a tamper-resistant device (e.g., on logic in a smart-battery) and managed by a security team out of reach from employees and untrusted on-board hardware. The detector observes CPU-only power trace (without noise from its own power-draw) and can run on a micro-controller (like our prototype) or an accelerator (for efficiency). To construct power-mimicry attacks that evade *any* power-anomaly detector, we do not customize the mimicry to specific classifier and its parameters. Instead, we apply a generic method to replicate the power behavior so that such evasive malware can stay effective when detector are updated. As long as the above conditions are met, we do not impose further constraints on applications or OS running in the device once the detector is deployed.

Deployment Platforms. Although we use an ARM-based system (§ V-A1) for our evaluation, this platform—including a triple-issue out-of-order quad-core processor with a 15-stage pipeline and two-level cache hierarchy—uses design and fabrication techniques similar to x86-based embedded systems. In addition, recent advances in cross-platform power models [62] suggest that power traces indeed carry information about program execution that can be modeled with $\sim$99% accuracy – our insight is to use this information to detect attacks. We believe that our methodology used to quantify the operating range and the use of power-anomaly as an *out-of-band* detector is thus applicable to other (e.g., x86) platforms.

B. Implementing Representative Micro-Architectural Attacks

Covert-Channel Attacks. Cache- and memory-bus- based covert-channel attacks are two high-performance attacks that break *confidentiality* [32].

In general, covert-channel attacks have two phases, a *Synchronization* phase, *synch* for short, where two processes determines when to start communication; and a *Communication* phase, *comm* for short, where two processes transmit bits after *synch*. In *synch* phase, for both cache-based and memory-bus-based covert-channel attacks, the trojan and spy have to initiate communication at the same time so that integrity of the transferred information is preserved. In our implementation, this is achieved by reading core local time stamp counter (TSC), and waiting until a pre-agreed time point, e.g, both trojan and spy wait until lower bits of TSC overflows, to start communication.

We implement a cache-based covert-channel attack *without Hyper-Threading* or cache bypassing instructions like *clflush*. Without *Hyper-Threading*, the *comm* phase of a cache-based covert-channel attack has to use contentions on the LLC shared among all cores—the trojan process sends information to the spy process running on another core. This *comm* phase works in *Prime-Access-Probe* fashion [32]. In each iteration, by evicting spy's data in LLC or not, trojan process transmits one bit of information to spy process. The spy process uses HPC to count number of cache misses encountered during probing. Larger number of cache misses indicates data invalidated. Although an alternative is using a timing channel, using HPC results in a less noisy channel and higher bandwidth in our setup.

We implement the memory-bus-based covert-channel *without* the bus locking instructions like *xchg*. For this covert-channel, the *comm* phase consists a trojan process transmitting a single bit each time through memory bus by creating bus congestion to transmit a bit '1'. The spy process then uses its own execution time to determine the bit transmitted. For both trojan and spy processes, we extend 'cache eviction set' [14] access strategy which generates maximum memory bus traffic on our system.

Spectre Attacks. We implement an intra-process Spectre attack *without* cache flush instructions like *clflush*. In an intra-process Spectre (bounds check bypass), the victim has valid access to confidential data. Attacker exploits a conditional branch residing in victim's code by feeding the branch a malicious input sequence to mistrain the branch predictor. We apply a similar method elaborated by Kocher *et al.* [16]. The wrong path of the branch accesses an out-of-bound, confidential memory location and subsequently leaks the value loaded. In addition, to construct a reliable cache side-channel, we perform an offline timing side-channel analysis to reverse engineer congruent address mappings. We then try techniques introduced by Lipp *et al.* [14] to find a deterministic 'strategy' for cache line evictions. However, we observe that, because of the random replacement policy of ARM cores, none of the 10,000 'eviction strategies' that we evaluated evict cache lines deterministically. Hence, we pick the best two strategies (among 10,000) that evict a line with high likelihood and are fast. Each time we want to evict a target line (the arrays bounds variable and the cache lines used to infer the secret), we make two attempts with both strategies to make eviction closer to being deterministic. Our Spectre thus has two phases – evict and reload, with a short speculative access by the victim in between. That is, the attacker evicts its own cache lines (the *evict* phase); it then tries to mislead the victim program to speculatively read a classified memory location; and finally it infers the secret byte by reloading the the cache line.

Rowhammer Attacks. While the hierarchical memory system reduces the probability of rapid accesses to the same DRAM row, purposefully crafted memory access patterns can still cause bit-flips. We exploit an efficient attack vector, similar to Van Der Veel *et al.* [15]: Our attack exploits Direct Memory Access (DMA) buffers to bypass caches and issues rapid memory accesses. Our attack also applies the double-sided hammering to induce bit flips in DRAM rows. To achieve this attack, we reverse engineer the DRAM structure and determine the row size. Upon obtaining the row size, our attack chooses two physical pages on adjacent rows to the victim row where target physical page resides and then start issuing rapid DRAM access until the bit-flip occurs. For the power analysis, we only capture power trace of the bit-flipping phase of rowhammer attacks, which turns out to be sufficient in identifying rowhammer.

IV. Designing Power Mimicking Micro-Architectural Attacks

In this section, we discuss our methodology to construct an evasive malware given a benign application's power profile.

A. Designing Power-Mimicking Malware

Power signature of a program on a given micro-architecture is primarily determined by its source code. Intuitively, in order to *ideally* mimic the power trace, which is a physical characteristic of the target application, the malware has to replicate all instructions—such a malware, however, will no longer be a malware but instead be a replica of the benign application. In a realistic setting, modern embedded processors execute instructions much faster than existing on-board power sampling frequency, which provides malware the opportunity to mimic coarse-grained power behavior of benign applications while still be able to complete some malicious tasks. The key idea behind power mimicking is that a malware can shape its power behavior by inserting a precise amount of extra operations during execution at a finer granularity than power sampling. For example, in one setup where the detector is sampling power at 5-ms granularity, the malware can execute a sequence of power-hungry instructions every 200us to increase the power consumption. To lower power, the malware can insert *nop* instructions or simply sleep.

We define the terms below to describe our methodology.

- **Atomic task:** In order to make a progress towards completing a certain malicious task, a malware has to execute a sequence of instructions that cannot be stopped and resumed later. We define this sequence of instructions as *atomic task*. A malware may have multiple atomic tasks. Each of these tasks take a certain amount of *time* and *power* to successfully execute.
- **Atomic task period:** *Atomic task period* is the time needed for malware to execute an *atomic task*. For malware that has multiple *atomic tasks*, *atomic task period* is the longest one.
- **Atomic task power:** During the execution of an *atomic task*, the amount of energy (i.e. average power) consumed cannot be further reduced by inserting *nop* instructions or sleeping. Otherwise, the malicious task will fail, which violates malware's original goal. We define this amount of required power as *atomic task power*. In a malware that has multiple *atomic tasks*, *atomic task power* is the minimum one.

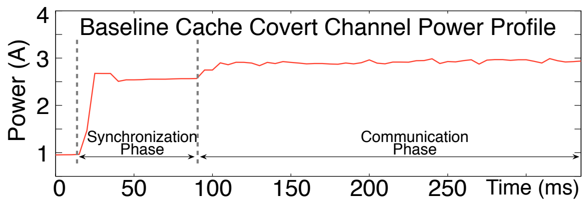


Fig. 4: Atomic task breakdown of baseline cache-based covert-channel attack. Two phase of the attack consists of two types of atomic tasks, *synch* and *comm* (§ III-B).

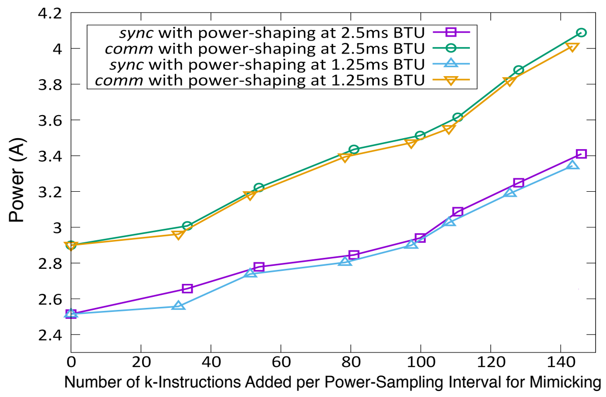


Fig. 5: Power shaping capability of evasive covert-channel attacker by inserting power hungry instructions. After executing an atomic task (*sync* or *comm*), the malware issues a sequence of instructions and records the corresponding power consumption.

- **Power sampling interval:** The power monitor takes samples with a fixed period, which we call *power sampling interval*. Each sample is the average power since the previous sample.
- **Basic time unit:** Malware partitions a *power sampling interval* into *basic time units* within which it can execute a fixed number of *atomic tasks* and then shape its power signature in between. *Basic time unit* is a design-time constant for the malware and is a divisor of *power sampling interval*. *Basic time unit* is the finest time granularity at which the malware uses to shape its power. For example, for a *basic time unit* of 1 millisecond, malware can use the first 200us to execute an *atomic task*, and the rest for power shaping.

We make two key observations: *(1) If atomic task period is longer than power sampling interval, the malware cannot shape its power within this period; (2) a malware cannot perform an atomic task when the atomic task's power consumption is higher than the one of benign application for a given basic time unit.*

Fig. 4 shows the two phases of the baseline covert-channel attack as described in § III-B: *Synchronization* and *Communication*. The *atomic task period* in *Synch* phase is the time to read the timer and determine whether to start communication, which takes less than 1us. In *Comm* phase, *atomic task period* is the time required to send/receive a single bit, which can take from 125us to 1ms in our experimental setup depending on malware's design choice on maximizing channel bandwidth. The *atomic task period* in this covert-channel is shorter than the 5ms *power sampling interval*. If there were no power channel detector in the system, a malware would take the entire *basic time unit* for *Synch* and *Comm* without any power mimicking. However, in order to evade a power channel detector, the safest course of action a malware can take is to shape its power within every *power sampling interval* to mimic a benign application for that power sample. Hence, malware breaks down each *power*

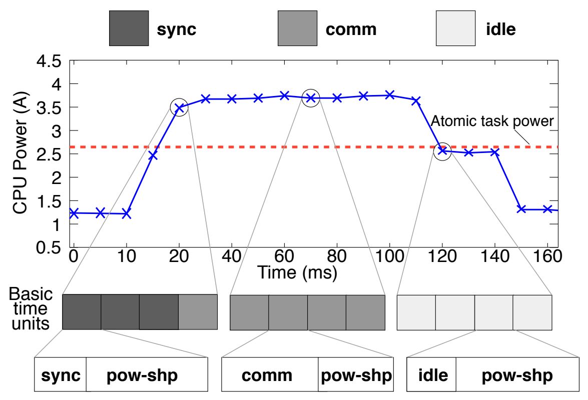


Fig. 6: Evasive covert-channel attacker design.

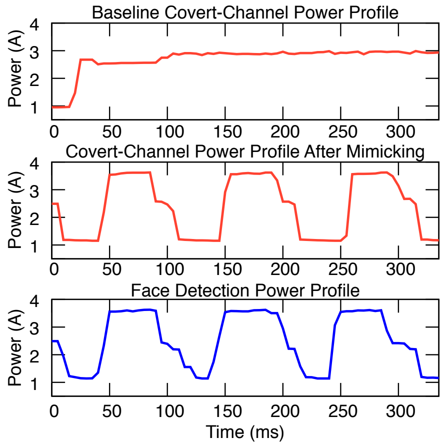


Fig. 7: Power profile of baseline attack, power-mimicking cache-based covert-channel attack, and face detection.

sampling interval into smaller *basic time units* within which the malware can first potentially execute one type of *atomic task* and then shape its power. Within that 5ms *power sampling interval*, power consumed in all *basic time units*, aggregated, mimics the power draw of the targeted benign application. In our method, mimicking malware distributes the total power uniformly to *basic time units*, hence, malware aims to consume the same amount of power at each *basic time units*.

Given these constrains, we modify our cache-based covert-channel to mimic a face detection application, under our platform's 5-ms power sampling constraint. To successfully construct a power mimicking attack, we first study the malware's ability to shape its power. Fig. 5 shows the range of power a malware can achieve. In this experiment, we first execute an atomic task and then apply power mimicking. Hence we show the *atomic task power* (with zero instructions inserted) as well as the maximum power a malware can achieve by inserting power-hungry instructions, e.g. SIMD floating-point subnormal operations. We vary the total number of power-hungry instructions inserted in each *basic time units*, a 5-ms time window in Fig. 5, to plot this malware's power shaping range. Essentially, malware forms a map for the number of instructions inserted vs. the resulting power. As long as a benign application's power trace falls within this range, it is possible for a malware to shape its power to mimic this application. However, if in some time window, the benign application's power consumption is lower than malware's *atomic task power*, malware must stop malicious tasks completely to remain undetected.

Fig. 6 visualizes our methodology on cache covert-channels. It shows the power mimicry for a repeated period in the face detection application that consists of 19 power samples (depicted in blue). The figure shows the example where each data point in the trace is sampled at 5-ms intervals and highlights malware actions at 3 power samples. Malware breaks down a 5-ms window into four 1.25-ms *basic time units* within which only one atomic task executes. The malware mimics the benign application at each sample by first comparing the expected, benign power with the *atomic task power*; if this value is higher than the *atomic task power*, malware then executes an atomic task (*sync* or *comm*) and uses the remaining time in *basic time unit* to shape the power (*pow-shp*). Otherwise, malware waits idly and then executes power shaping operations. Note that the amount of instructions used for power shaping in a basic time unit is determined from the map generated beforehand (Fig. 5). Fig. 7 shows the result of our methodology. While the baseline covert-channel attack has a visually distinct power signature, power-mimicking malware exhibits a similar behavior to the benign face detection application.

In addition to cache covert-channels, we modify our Spectre attack with power shaping capability. First, two *atomic tasks* are identified: (a) *Evicting cache lines*, and (b) *Attempting to infer the secret value that is loaded into cache if the vulnerable branch is mispredicted*. The baseline Spectre tries to recover one bit at a time, which requires a side-channel consisting of two cache lines from different sets. Evicting both two lines on our platform takes around 140us. We empirically determine that the misprediction rate of the vulnerable conditional branch is around 0.1% with malicious training, resulting in an attack bandwidth of 7.13 bps. Profiling results show that each attempt to mistrain the branch predictor takes less than 1us, with marginal effect on a power draw sampling up to 200Hz on our platform. The computational complexity of side-channel data extraction, however, is the key to crafting a power-mimicry Spectre.

Note that we visualize our mimicry method on the face detection application, while it can be extended to mimic other applications. Mimicking any application in the benign suite is sufficient to evade generic detectors that are trained on the suite.

B. Effectiveness of Evasive Attacks

We measure micro-architectural covert-channel attacks' effectiveness using channel capacity as the metric [32], [63], [64]—modeling each covert channel as a discrete memoryless channel, constructing channels with different trade-offs between bit-duration and reliability against noise, and using the Blahut-Arimoto algorithm to compute channel capacity. Fig. 8 compares the bandwidth of the baseline cache-based covert-channel to the power-mimicking variants with different *basic time units*—the window of time within which a fixed number of *atomic tasks* are executed. Note that y-axis shows the bandwidth in logarithmic scale. Evasive malware have a minimum requirement of 125us for each *atomic task*, but beyond this, evasive malware bandwidth benefits from a shorter *basic time unit*, as it can execute *atomic tasks* more frequently. Hence as the *basic time unit* increases, bandwidth goes down. Baseline attacks execute one *sync* task for each *basic time unit* followed by *comm* tasks continuously—i.e., an overhead of starting a new *basic time unit*. Hence, bandwidth increases with longer *basic time unit*. The baseline attack can achieve a peak bandwidth of 7574 bps (taking synchronization into account, which is necessary whenever the malware resumes from the idle state). However,

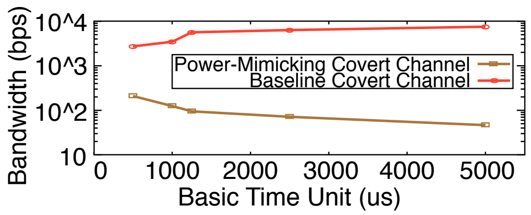


Fig. 8: Evasive covert-channel bandwidth comparison. Increasing basic time unit negatively effects the bandwidth of mimicking attacks since it reduces the number of executed atomic tasks. By contrast, it increases the bandwidth of baseline attacks that constantly execute atomic tasks.

with full power shaping capability, malware bandwidth degrades by over 36×, resulting in a maximum bandwidth of 209 bps.

For Spectre, similar to covert-channel attacks showed in Fig. 8, a larger *basic time unit* makes attacks unreliable and reduces bandwidth. While maintaining the capability of power shaping, attacker can extract up to 6 bits of secrets in each attempt, which requires evicting 64 lines, taking up to 5ms. However, due to the fact that eviction strategies may fail to evict a cache line, this approach of exfiltrating more bits each time results in a noisier channel (i.e., the probability of a failed eviction is higher than that of a misprediction). Therefore, our Spectre can only shape its power by extending *basic time unit* up to 5ms—we observe that further reduction in leak bandwidth results in the failure of the attack. We attribute this behavior to TLB and cache pollution between *atomic tasks*. The resulting latency of the access to the cache channel is longer than the time of a misspeculation resolution. In summary, mimicking in Spectre results in a bandwidth range of 7.13 down to 1.0 bps.

We also modify the rowhammer attack to mimic the power signature of face detection. However, we found that, if the interval between two read operations to DRAM exceeds 200 ns, it is unlikely for this attack to trigger bit-flips in our experimental platform, which complies with the findings of van der Veen *et al.* [15]. Hence, in order to issue DRAM accesses that cause bit flips, the *atomic task* of rowhammer is issuing a sequence of rapid DRAM accesses instead of issuing one. This results in an 64ms *atomic task period* (DRAM refresh interval), which is significantly longer than the power sampling interval. We therefore conclude that, in our setup, it is infeasible to craft a power-mimicking rowhammer that evades the detector.

V. Detecting Attacks

Intuitively, a detector using power channel as defense should be effective since each application and malware has its own power signature. However, § IV-A demonstrates that malware can shape their power signatures. Therefore, evaluating power-anomaly detectors requires quantifying the operating range of detectors against a range of differently effective attacks. We must apply a white-box evaluation, i.e. to evaluate against both baseline and evasive malware.

A. Experimental Setup

1) Target Embedded Device: For our white-box evaluation, we use an Odroid-XU3 board with a built-in energy monitor. It is powered by a Exynos SoC with a big-core and a little-core clusters. Specifically, we run all our experiments on the big-core cluster and use the little-cores to emulate an *out-of-band* detector. The big-core cluster consists of OoO Cortex-A15 quad cores with 32KB private L1I/D caches and a 2MB shared L2. The board runs ubuntu-`14.04` with gcc-`4.8` with a 2GB LPDDR3 running at 933MHz. The on-board energy monitor, INA231, can measure the power of the big-core cluster every 1.25ms. We configure the system with DVFS enabled, specifically the CPU frequency governor is set to *interactive* for the big-cluster, which dynamically scales processor frequency between 200MHz and 2GHz. In addition, for more controlled experiments, we conduct the same tests with CPU frequency fixed at 2GHz.

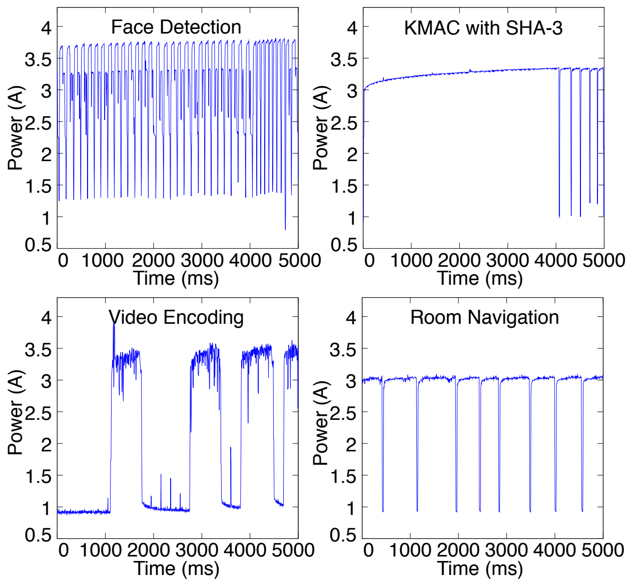


Fig. 9: Power signatures of benign workloads.

2) Benign Apps and Their Power Profile: For benign applications under test, prior work [6]–[8] use parts of existing embedded system benchmarks [65], [66] or synthetic programs [7], [9] that have fairly short (up to 20 ms) execution time. These applications, in the best case, run for half a second on our platform. While this trace length is reasonable for prior detectors using external, fine-grained measurements, given the realistic embedded evaluation of our work with coarse-grained sampling and relatively faster system, we use applications that are representative of realistic scenarios to stress test our detectors. These applications—*cognition*, *cryptography*, *video coding*, and *decision making*—have been intensively studied and used in real-world settings like robotic and drone systems on ARM processors [67]–[69]. Specifically, a *multithreaded* face detection from the OpenCV with dataset from a video surveillance benchmark [33], a KMAC hashing (SHA-3) [70], a *multithreaded* video encoding application from FFmpeg and a room navigation running on the *multiprocess* Robot Operating System (ROS) [71] are used.

Fig. 9 illustrates the power signatures of the big-core while executing these applications. The figures demonstrate phases in the benign applications. While KMAC and room navigation have minor variations in power, video encoding and face detection have significant variations in their power consumption. For example, video encoding does more computations (higher power) if the objects in the video have many motions or scene changes. Likewise, face detection has a periodic power behavior with the period determined by the frame rate of inputs.

B. Methodology and Detector Design Space

1) Methodology and Metrics: Figs. 5 and 8 show that attacks compromise bandwidth for power-mimicking capabilities. When

TABLE I: Summary of detection results (AUC) for baseline attacks with *interactive* DVFS governor

	One-Class SVM					LSTM					
Benign / Attacks	Face Detection	Video Encoder	SHA-3 KMAC	Room Navigation	GeoMean	Face Detection	Video Encoder	SHA-3 KMAC	Room Navigation	All Benign Apps	GeoMean
Cache Covert Channel	0.9964	0.9996	0.9880	0.9558	**0.9848**	0.9985	0.9847	1.0000	0.9615	**0.9917**	**0.9861**
Memory-bus Covert Channel	0.9938	1.0000	0.9814	0.9179	**0.9727**	0.9969	0.9816	0.9999	0.9589	**0.9898**	**0.9842**
Spectre	0.9980	0.9879	0.9763	0.9919	**0.9885**	0.9974	0.9821	0.9939	0.9956	**0.9962**	**0.9822**
Rowhammer	0.9928	0.9999	0.9777	0.9850	**0.9863**	0.9972	0.9828	1.0000	0.9823	**0.9898**	**0.9906**
GeoMean	**0.9927**	**0.9968**	**0.9828**	**0.9622**		**0.9975**	**0.9828**	**0.9984**	**0.9745**	**0.9919**	

evaluating power-anomaly detectors against evasive malware, we study detection results against evasive malware with varying bandwidth (hence, varying ability to mimic power). This way, our work reveals the operating range of power-anomaly detectors. Our threat model (§ II) requires a detector to identify anomalies from expected behaviors and make a binary decision on power traces: benign or malicious. Power-anomaly detectors classify a given input power trace as benign or malicious by producing a score indicating the likelihood of the input trace being malicious. Since our threat model aims to detect 0-day architectural attacks and malware, it is important for us to perform training *only* on the power traces of benign applications. Hence each malware is presented to our detectors as a 0-day exploit. When training different machine learning algorithms, we use 5-fold cross-validation to determine the best model specific parameters for each classifier. Cross-validation prevents detectors from over-fitting the benign samples in the training set.

We evaluate trained detectors using Receiver Operating Characteristic (ROC) Curve and Area Under Curve (AUC). ROC Curve illustrates the effectiveness (true positive vs. false positive) of a binary classifier when varying its discrimination threshold. AUC reflects the probability that a classifier would assign a higher score to a randomly chosen true positive input than to a randomly chosen true negative input. In malware classification, a malicious input is considered positive.

2) Machine Learning Classifiers: We explore three different ML algorithms that are commonly used in our setting: radius-distance based nearest neighbor (RBNN) classifier, one class support vector machine (ocSVM), and vanilla long short term memory (LSTM). Feature extraction and selection in frequency-domain is widely used to reduce measurement noise. we use Discrete Wavelet Transform (DWT) for features extraction to capture both time- and frequency-domain information. Detecting anomalies using power trace is a time-series classification problem. However, for classifiers that are not initially designed for time-series classification (e.g., RBNN and ocSVM), it is useful to build a model (selecting features) to capture both short and long term similarity in traces. To that end, we use Bag-of-Words model [72]. Bag-of-Words model treats feature vectors in each time window as a document of words and forms a codebook from them and assigns code for each feature vector.

Radius-Distance Based Nearest Neighbor Classifier: RBNN is trained on code representation of Bag-of-Words model for all possible benign applications in the system (§ V-A), and predicts a power trace to be either one of the benign applications or an outlier based on a radius-distance threshold.

One-Class Support Vector Machine: ocSVM is a commonly used unsupervised ML algorithm for anomaly classification. It is also trained atop the Bag-of-Words model, except that each ocSVM instance is only trained on power traces from a single benign application and classifies if a power trace indeed shows expected behavior. In practice, it is possible to deploy multiple ocSVM detectors to a mixed workload system.

Long Short Term Memory: LSTM is a recurrent neural network (RNN) that is specifically designed for time-series classification. It captures the dependency information in the trace. We refer readers to the manuscript [73] for further technical details. We train a small vanilla LSTM on extracted feature vectors from DWT for all benign applications, without the Bag-of-Words model, during which we use our best effort to determine parameters yielding effective classification results.

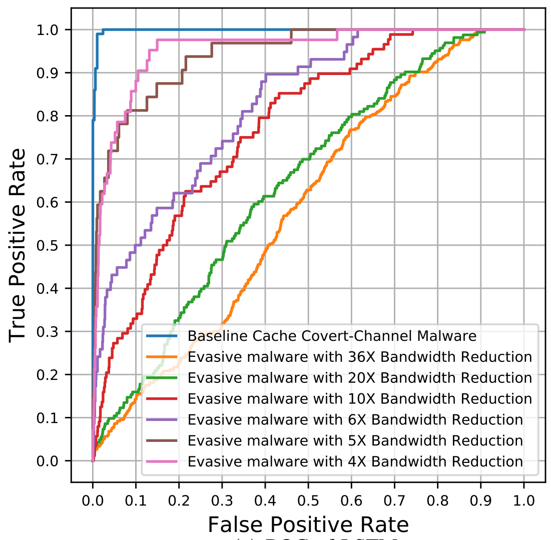
3) Parameters: We use Bag-of-Words model for RBNN and ocSVM. Prior work [72] suggests that size of codebook in the Bag-Of-Words model is an important parameter. A larger codebook might capture more patterns, however, it might also degrade detection performance due to sparsity in the resulting code representation of power traces. Large codebook size requires more computation and has longer detection latency.

Besides the specific algorithms used and their corresponding parameters, sampling related parameters must be considered when designing power-anomaly detectors. First, sampling granularity plays an important role. Indeed, prior work [6]–[8] all use relatively fine grained sampling method to gain insights from program executions. In our experiments, by contrast, we aim to evaluate effectiveness of embedded detectors that leverage realistic, coarser-grained sampling methods, i.e. sub-Kilo Hz sampling frequency while system operates at GigaHz. We sweep the sampling period from 5ms, 10ms, 20ms, 40ms, to 80ms and found that at a practical sub-Kilo Hz range, finer grained sampling provides better detection performance. However with further fine-grained sampling capability, the cost of the power sensor increases beyond practicality for the embedded deployment. Second, the length of the time window used for each detection decision, Time-to-Detection (TTD), is also crucial to a detector's performance. While increasing detection latency, longer TTD provides more insights of program behaviors.

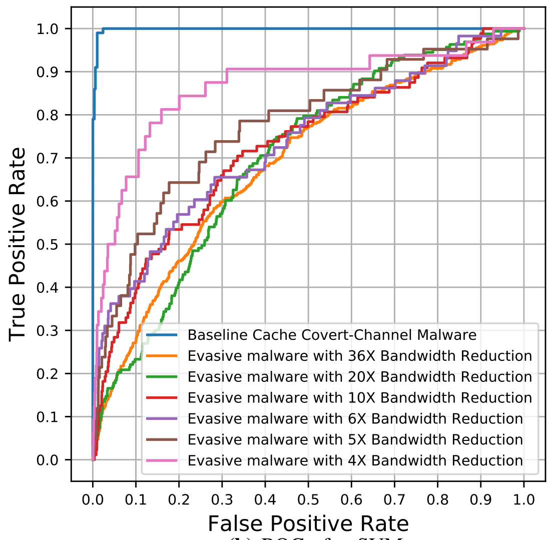
C. Operating Ranges of Detectors

Among the three algorithms, RBNN fails to detect baseline malware. We explore the design space for RBNN by varying feature extraction algorithms, codebook sizes, and distance functions. RBNN produces results with AUCs lower than 0.7 in most cases. The underlying reason is that each feature contributes equally in the distance function to decisions made by RBNN, while different features usually characterize a class differently.

Our results show that the rest two (ocSVM and LSTM) detectors have near perfect detection performance against baseline malware. Table I summarizes the detection results for these two effective classifiers. Each column in the table gives the AUC values of the best performing parameter configurations of detectors trained on a certain benign application to capture several micro-architectural attacks. The AUC results presented in Table I validate that the baseline malware are indeed almost perfectly identified. Note that each entry shows the result of



(a) ROC of LSTM



(b) ROC of ocSVM

Fig. 10: Both detectors achieve better Area Under Curve against less evasive covert-channel attacks.

AUC for classification between a benign application and a malware—manner. The single benign application columns show detection performance when the power-anomaly detector is used as a tailored detector. The penultimate column (from the right), in addition, presents the results when power-anomaly detector uses LSTM classifier as a generic detector, classifying between all benign applications and malware. Across all different benign settings, both classification algorithms detect baseline Spectre and rowhammer attacks with high AUC. This is because of the simple and regular power signatures of baseline Spectre and rowhammer attacks. For both classification algorithms, the detection results drop slightly when detectors are tailored to protect the room navigation application. This is due to the fact that the room navigation runs on top of the *multiprocess* ROS framework. Indeed, besides the benign room navigation application, ROS runs separate processes that perform system status logging and inter-process message passing. *Experiments performed with DVFS off show close and sometimes better detection performance and similar trend with room navigation.*

We further show the operating range of detectors by evaluating them against different evasive malware. Fig. 10 illustrates the detection performance of ocSVM and LSTM with corresponding best parameters derived at training time. Specifically, TTD is set to 500ms, sampling granularity is set at 5ms for both classifiers. In addition, results for ocSVM showed in Fig. 10b is trained with Bag-of-Word codebook size of 200. For LSTM, a network of 3 layers with 20 hidden states is used. These classifier configurations give the best performance against all baseline malware, hence we further evaluate these parameters of ocSVM and LSTM against evasive malware. We stop tweaking evasiveness of attacks when either they no longer *succeed* or AUC (detection performance) of both detectors drop below 0.7. The AUC values for LSTM detector against evasive malware ranges from 0.5859 to 0.9538 for power-mimicking covert-channel attacks that reduce bandwidth from 36× to 4×. Meanwhile, the detector using ocSVM produces AUC of 0.6882 when cache-based covert-channel reduces its effectiveness by 36×. Similarly, when the Spectre attack drops its bandwidth from 7.1 to 1.0 bps, AUC produced by LSTM and ocSVM drop to 0.9124 and 0.8671 respectively.

TABLE II: Resource utilization by hierarchy[1]

Module	Logic Cells	Memory (9kb)	DSP (9x9)
NIOS II microcontroller	3018	58	4
SDRAM burst controller	1539	0	0
Hardware timer	150	0	0
Others[2]	303	2	0
Total	**5046**	**60**	**4**

[1] Resource on Cyclone-IV EP4CE22F17C6.
[2] Others include UART controller, PLL and debug interface, and sld hub infrastructure.

D. Detector Prototype on a Low-Cost FPGA

We prototype the *out-of-band* detector using ocSVM on a low-cost FPGA to demonstrate its simplicity. We use a DE0-Nano with Cyclone-IV FPGA. We configure a NIOS-II softcore to execute the detection software. This processor uses the NIOS-II/f configuration with a 32-bit RISC architecture with 16kB of instruction cache, 32kB of data cache, and a 32MB of SDRAM. Table II shows the FPGA resource utilization. While we prototype on a small FPGA, further energy-constrained environments can leverage the significant research into low-power and real-time ML classifiers (SVM and LSTM specifically) [74], [75].

We port the detection software with a 500ms TTD. The entire detection software has a code size of 216Kb. The total cycle count is approximately 48M, which corresponds to 480 milliseconds with the operating clock frequency of 100MHz. Note that, this value is smaller than the target TTD of 500ms. As expected, due to the large number of multiplications, codeword assignment is the timing bottleneck of the detection, which takes around 82.21% of the total cycles.

VI. Conclusions

Rowhammer attacks on DRAM and side- and covert-channel attacks on shared micro-architecture are here today. Forthcoming memory technologies and hardware substrates will no doubt motivate further instruction-driven abuse of hardware. In this setting, long-term embedded systems will find continually adding in-band defenses and getting the system re-certified to be extremely expensive. An out-of-band detector is thus critical for high-assurance embedded systems. We demonstrate that micro-architectural attacks are extremely noisy when observed through the power channel – rowhammer does not succeed and covert/speculative channels have to slow down considerably even if the attacks attempt evasion by hiding behind benign programs. Power-anomaly is thus a simple and effective defense compared to ones that rely on new hardware or OS techniques, and introduces smart power management units and software defined batteries as trustworthy out-of-band security monitors.

VII. Acknowledgement

This work was supported in part by Lockheed Martin, and by the National Science Foundation under Grants No. 1850373 and 1527888.

References

[1] S. Gollakota *et al.*, "They can hear your heartbeats: Non-invasive security for implantable medical devices," *SIGCOMM*, 2011.

[2] D. Benjamin, "TLS ecosystem woes–why your crypto isn't real world yet," 2018, Real World Crypto Symposium.

[3] Intel, "Microcode revision guidance," Tech. Rep., 2018.

[4] S. Grover *et al.*, "The internet of unpatched things," FTC PrivacyCon.

[5] N. Sehatbakhsh *et al.*, "Syndrome: Spectral analysis for anomaly detection on medical iot and embedded devices," in *HOST '18*, 2018.

[6] A. Nazari *et al.*, "EDDIE: Em-based detection of deviations in program execution," in *ISCA '17*. ACM, 2017.

[7] Y. Han *et al.*, "Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations," in *CCS '17*. ACM, 2017.

[8] Y. Liu *et al.*, "On code execution tracking via power side-channel," in *CCS '16*. ACM, 2016.

[9] J. Hernández Jiménez *et al.*, "Towards a cyber defense framework for scada systems based on power consumption monitoring," in *HICSS '17*.

[10] C. R. Aguayo González *et al.*, "Power fingerprinting in sdr integrity assessment for security and regulatory compliance," *Analog Integrated Circuits and Signal Processing*, 2011.

[11] J. H. Reed *et al.*, "Enhancing smart grid cyber security using power fingerprinting: Integrity assessment and intrusion detection," in *FIIW '12*.

[12] H. Genc *et al.*, "Flying IoT: Toward low-power vision in the sky," *IEEE Micro*, 2017.

[13] M. P. Singh *et al.*, "Evolution of processor architecture in mobile phones," *International Journal of Computer Applications*, 2014.

[14] M. Lipp *et al.*, "Armageddon: Cache attacks on mobile devices," in *USENIX Security '16*. USENIX, 2016.

[15] V. van der Veen *et al.*, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *CCS '16*. ACM, 2016.

[16] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *S&P'19*, 2019.

[17] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *USENIX Security'18*, 2018.

[18] D. Wagner *et al.*, "Mimicry attacks on host-based intrusion detection systems," in *CCS '02*. ACM, 2002.

[19] A. Tang *et al.*, "Unsupervised anomaly-based malware detection using hardware features," in *RAID '14*. Springer, 2014.

[20] K. N. Khasawneh *et al.*, "Rhmd: Evasion-resilient hardware malware detectors," in *MICRO '17*. ACM, 2017.

[21] M. Kazdagli *et al.*, "Quantifying and improving the efficiency of hardware-based mobile malware detectors," in *MICRO '16*, 2016.

[22] T. Zhang *et al.*, "New models of cache architectures characterizing information leakage from cache side channels," in *ACSAC '14*.

[23] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser," in *DATE '17*, 2017.

[24] A. Badam *et al.*, "Software defined batteries," in *SOSP '15*. ACM.

[25] J. Han *et al.*, "Smart home energy management system including renewable energy based on ZigBee and PLC," *IEEE Transactions on Consumer Electronics*, 2014.

[26] D. Dash *et al.*, "When gossip is good: Distributed probabilistic inference for detection of slow network intrusions," in *AAAI'06*. AAAI Press.

[27] C. V. Zhou *et al.*, "A survey of coordinated attacks and collaborative intrusion detection," *Comput. Secur.*, 2010.

[28] E. Vasilomanolakis *et al.*, "Taxonomy and survey of collaborative intrusion detection," *ACM Comput. Surv.*, 2015.

[29] A. Reina *et al.*, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," *EuroSec*, 2013.

[30] M. Ozsoy *et al.*, "Hardware-based malware detection using low-level architectural features," *IEEE Transactions on Computers*, 2016.

[31] K. N. Khasawneh *et al.*, "Ensemble learning for low-level hardware-supported malware detection," in *RAID '15*. Springer, 2015.

[32] C. Hunger *et al.*, "Understanding contention-based channels and using them for defense," in *HPCA '15*. IEEE, 2015.

[34] O. Aciiçmez *et al.*, "On the power of simple branch prediction analysis," in *ASIACCS '07*. ACM, 2007.

[33] Y. Wong *et al.*, "Patch-based probabilistic image quality assessment for face selection and improved video-based face recognition," in *IEEE Biometrics Workshop, CVPR Workshops*. IEEE, 2011.

[35] F. Liu *et al.*, "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *HPCA '16*, 2016.

[36] Z. Wang *et al.*, "Covert and side channels due to processor architecture," in *ACSAC '06*, 2006.

[37] D. Evtyushkin *et al.*, "Covert channels through random number generator: Mechanisms, capacity estimation and mitigations," in *CCS '16*.

[38] Z. Wu *et al.*, "Whispers in the hyper-space: High-speed covert channel attacks in the cloud," in *USENIX Security '12*. USENIX, 2012.

[39] A. Shafiee *et al.*, "Avoiding information leakage in the memory controller with fixed service policies," in *MICRO '15*. ACM, 2015.

[40] NIST, "CVE-2018-3639," 2018.

[41] J. V. Bulck *et al.*, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *USENIX Security 18*.

[42] ARM, "Speculative processor vulnerability," Tech. Rep., 2018.

[43] Intel, "Initial performance data for data center systems," Tech. Rep., 2018.

[44] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *ISCA'14*, 2014.

[45] M. Seaborn *et al.*, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, 2015.

[46] Y. Xiao *et al.*, "One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation," in *USENIX Security 16*. USENIX.

[47] R. Qiao *et al.*, "A new approach for rowhammer attacks," in *HOST '16*.

[48] Y. Wang *et al.*, "Timing channel protection for a shared memory controller," in *HPCA '14*, 2014.

[49] Y. Zhang *et al.*, "Homealone: Co-residency detection in the cloud via side-channel analysis," in *SP '11*, 2011.

[50] A. Moser *et al.*, "Limits of static analysis for malware detection," in *ACSAC '07*, 2007.

[51] D. Arp *et al.*, "Drebin: Effective and explainable detection of android malware in your pocket." in *NDSS*, 2014.

[52] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *ISCA '13*. ACM, 2013.

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

[54] L. Liu *et al.*, *VirusMeter: Preventing Your Cellphone from Spies*. Springer, 2009.

[55] J. Hoffmann *et al.*, *Mobile Malware Detection Based on Energy Fingerprints – A Dead End?* Springer, 2013.

[56] H. Kim *et al.*, "Detecting energy-greedy anomalies and mobile malware variants," in *MobiSys '08*. ACM, 2008.

[57] J. P. B. Dixon, S. Mishra, "Time and location power based malicious code detection techniques for smartphones," in *IEEE NCA '14*, 2014.

[58] L. Caviglione *et al.*, "Seeing the unseen: Revealing mobile malware hidden communications via energy consumption and artificial intelligence," *IEEE Transactions on Information Forensics and Security*, 2016.

[59] K. Wang *et al.*, "Anagram: A content anomaly detector resistant to mimicry attack," in *RAID '06*. Springer, 2006.

[60] A. Zimek *et al.*, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5.

[61] "The big hack: How china used a tiny chip to infiltrate u.s. companies - bloomberg," https://www.bloomberg.com, (Accessed on 10/17/2018).

[62] X. Zheng *et al.*, "Accurate phase-level cross-platform power and performance estimation," in *DAC'16*, 2016.

[63] S. Arimoto, "An algorithm for computing the capacity of arbitrary discrete memoryless channels," *IEEE Transactions on Information Theory*, 1972.

[64] R. Blahut, "Computation of channel capacity and rate-distortion functions," *IEEE transactions on Information Theory*, 1972.

[65] F. Vahid *et al.*, "Platform tuning for embedded systems design," *Computer*, 2001.

[66] M. R. Guthaus *et al.*, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE WWC-4*, 2001.

[67] T. Reed *et al.*, "Skynet: A 3g-enabled mobile attack drone and stealth botmaster." in *WOOT*, 2011.

[68] Q. Wu *et al.*, "Cognitive internet of things: A new paradigm beyond connection," *IEEE Internet of Things Journal*, 2014.

[69] D. Floreano *et al.*, "Science, technology and the future of small autonomous drones," *Nature*, 2015.

[70] J. M. Kelsey *et al.*, "Sha-3 derived functions: cshake, kmac, tuplehash and parallelhash," *Special Publication (NIST SP)-800-185*, 2016.

[71] M. Quigley *et al.*, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[72] J. Wang *et al.*, "Bag-of-words representation for biomedical time series classification," *Biomedical Signal Processing and Control*, 2013.

[73] F. A. Gers *et al.*, "Learning to forget: Continual prediction with lstm," *Neural Comput.*, 2000.

[74] K. H. Lee *et al.*, "Low-energy formulations of support vector machine kernel functions for biomedical sensor applications," *Journal of Signal Processing Systems*, 2012.

[75] S. Han *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *FPGA '17*. ACM, 2017.