Detecting Energy Attacks in the Battery-less Internet of Things

Luca Mottola[†+*] and Thiemo Voigt[†+]
[†]RI.SE Sweden, [+]Uppsala University (Sweden), [*]Politecnico di Milano (Italy)

Abstract. We present a technique to detect energy attacks in the battery-less Internet of Things (IoT). Battery-less IoT devices rely on ambient energy harvesting and are employed in a multitude of applications, including safety-critical ones such as biomedical implants. Due to scarce energy intakes and limited energy buffers, their executions become *intermittent*, alternating periods of active operation with periods of recharging energy buffers. Evidence exists that demonstrates how exerting limited control on ambient energy one can create situations of *livelock*, *denial of service*, and *priority inversion*, without physical device access. We call these situations *energy attacks*. Using concepts of approximate intermittent computing and machine learning, we design a technique that can detect energy attacks with 92%+ accuracy, that is, up to 37% better than the baselines, and with up to one fifth of their energy overhead. By design, our technique does not cause any additional energy failure compared to the regular intermittent processing.

1 Introduction

Ambient energy harvesting allows Internet of Things (IoT) devices to eliminate their dependency on traditional batteries [12]. This enables previously unattainable deployments, including safety-critical settings such as biomedical implants [1, 18, 23, 33]. Harvested energy is generally highly variable in time, yet energy buffers, such as capacitors, are generally limited. System shutdowns due to energy depletion are unavoidable and computing becomes *intermittent* [5].

Computing intermittently. Fig. 1 shows an example execution. The ambient charges the IoT device's onboard capacitor until voltage V_{on} is reached that causes the device to power on. The device senses, computes, and communicates as long as the capacitor charge remains above a threshold V_{off}. The device then switches off, waiting for the capacitor to reach V_{on} again. This pattern may occur on tiny time scales; for example, computing simple error correction codes on a battery-less IoT device may require as many as 16 energy cycles [11].

Due to resource constraints, applications run with no operating system support [5]. When the device powers off at V_{off}, the system state would normally be lost. Intermittently-computing IoT systems use checkpointing [4, 8, 11, 54, 61] or task-based programming [44, 62] to create persistent state on non-volatile memory (NVM). These systems operate as the device approaches V_{off}, allowing them to retain the application state across energy failures. Operations on NVM, however, are extremely energy hungry [46].

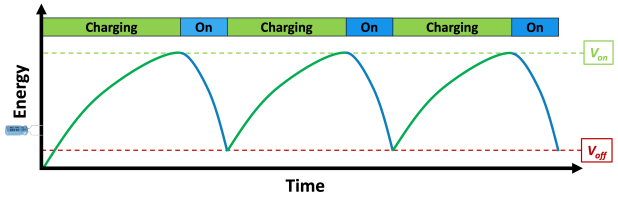


Fig. 1. Example intermittent execution.

Energy attacks. Evidence exists that shows how exerting limited control on ambient energy may steer intermittent executions in unintended ways [49]. We call these situations *energy attacks*. The simplest scenario consists, for example, in physically blocking a solar panel that powers the device, leading to a denial of service. Such an attack, however, would be straightforward to detect, as the system would suddenly and completely stop working.

Curically, much more subtle situations exist, where attackers may potentially do much more harm than with a complete denial of service situation that is immediately recognized and redressed. Energy attacks are indeed reported that that create situations of *livelock*, *priority inversion*, and *denial of service* [49]. Unlike the simple scenario above, these attacks create situations that are deceptively similar to legitimate executions.

Following a definition of the system and attack model in Sec. 3, in Sec. 4 we tackle the problem of *detecting* energy attacks. Intuitively, this means understanding when ambient energy provisioning does not follow the "natural" patterns. The problem appears as a case of *anomaly detection* [17]. Three peculiar requirements exist: detecting energy attacks *i)* accurately and *ii)* with low latency, while doing so *iii)* right on the IoT devices, as opposed to an external system, to spare the energy overhead of radio operations necessary to offload data to a third party. Our technique uses concepts of approximate intermittent computing [9, 59] and machine learning to ensure that, by design, the attack detection process does not cause additional energy failures compared to the regular intermittent processing, imposing minimal overhead.

Sec. 5 reports on the accuracy and overhead of our detection technique, based on 500K+ data points obtained using real-world energy traces, compared with two baselines. The results indicate that our technique is 92%+ accurate, which is up to 37% better than the baselines, and imposes an overhead that is up to one fifth of the baselines. Further, the detection performance is largely independent of the energy patterns and robust to previously unseen attacks.

Following detection of an energy attack, the system should apply countermeasures. In Sec. 6, we explore the multiple dimensions of the problem, articulate the related trade-offs, and inspire the design of defense techniques.

2 Background and Related Work

Our work is an example of the many ongoing efforts at running inference processes of machine learning models on resource-constrained devices [6]. We specif-

ically tailor an existing machine learning technique for detecting energy attacks on a batter-less IoT device running intermittently. In the following, we provide background information and survey related works in closely related areas.

Power attacks in data centers. Energy attacks resemble similarities with power attacks in data centers. Malicious workloads may generate power spikes on multiple servers at the same time, which causes branch circuit breakers to trip, leading to power outages [29, 40]. Detection techniques include machine learning applied to performance logs [19] and modeling user behaviors that may indicate the infrastructure is under-performing [40].

Common with our problem is that energy is part of the attack vector. However, the technology is extremely different, for example, in terms of workloads and hardware platforms. Moreover, in contrast to the attack model we describe in Sec. 3, attackers do not directly manipulate the energy provisioning channel and need access to the target data center or must be informed of its layout. In contrast, the attacks described by Mottola et al. that we consider [49], do not require physical access to the target device.

Security in battery-powered IoT. IoT devices are difficult to secure due to resource constraints, which complicates the use of mainstream security mechanisms and protocols [57]. Battery-powered IoT devices enable peculiar attacks, for example, in an attempt to drain batteries [36, 51, 48]. Low-power radios make IoT devices vulnerable to denial of service attacks, for example, due to intentional jamming [35]. Multi-hop networks require specialized network stacks that open to new kinds of attacks, in particular at the routing layer, motivating new security mechanisms ranging from hardware-based solutions [52] to methods for attack detection and mitigation that rely on machine learning [21, 56].

These approaches, unfortunately, fall short of expectations for battery-less IoT devices, where energy constraints are way more severe. Intermittent executions add a new dimension to the problem that requires specialized solutions.

Security in intermittent computing. The prevailing intermittent computing architecture includes a mixed-volatile MCU with built-in NVM and one or multiple capacitors to tame fluctuations of energy intake [20]. This configuration is seen in available platforms [31, 34] and deployments [1, 18, 23, 33].

The few existing security solutions in intermittent computing focus on securing persistent state. As an example, Krishnan et al. [37] demonstrate that persistent state is vulnerable to sniffing, spoofing, or replay attacks. In other works, Asad et al. [7] experimentally evaluate the use of different encryption algorithms and ARM TrustZone protection. Krishnan et al. [38] build on this and propose a configurable checkpoint security setting that leverages application properties to reduce overhead. Ghodsi et al. [28] use lightweight algorithms [13] for securing checkpoints. Valea et al. [60] propose a SECure Context Saving hardware module inside the MCU. In contrast, Grisafi et al. [30] present a hypervisor to manage and protect checkpoints.

Unlike these works, we study how to detect new types of attacks realized by exerting control on ambient energy provisioning.

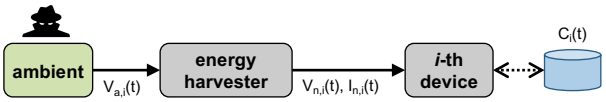


Fig. 2. System and attack model.

3 Energy Attacks

We describe first the system and attack model we adopt; next, we show evidence of example vulnerabilities.

System and attack model. Fig. 2 illustrates the system and attack model. Energy coming from the ambient a and arriving at the energy harvester of node i is modeled as a continuous signal of voltage $V_{a,i}(t)$. This describes the energy content made available by the ambient to node i at time t. For simplicity, our description here considers a single energy source. The corresponding analysis, however, applies no matter the number of energy sources, as long as the attack model is applicable to each of them. Relying on multiple energy sources may be, nonetheless, a way to defend against energy attacks, as we discuss in Sec. 6.

The energy harvester of node i takes $V_{a,i}(t)$ as input and transforms it into an energy signal of voltage $V_{n,i}(t)$ and current $I_{n,i}(t)$. The latter is a function of $V_{n,i}(t)$ and of the equivalent resistance offered by the charging circuitry at node i. The energy signal described by $V_{n,i}(t)$ and $I_{n,i}(t)$ charges the local energy buffer, eventually discharged while sensing, computing, or communicating. We model the charge available in the energy buffer of node i as $C_i(t)$.

The attacker has no physical access to the devices and no knowledge of the relation between $V_{a,i}(t)$ and $V_{n,i}(t)$ or $I_{n,i}(t)$. She can sniff and inspect packets, as well as intervene along the path from the energy source to the energy harvester attached to the device, including directly controlling the energy source. This means the attacker can alter the value of $V_{a,i}(t)$ taken as input at node i. We model this as a function $a_i(V_{a,i}(t))$, that is, a transformation a from the voltage domain to the same domain, specific to node i.

A straightforward example of function a that causes a denial of service at node i from t' onwards is $a_i(V_{a,i}(t)) = 0, t > t'$, that is, the harvester at node i receives no energy after t'. The energy buffer at node i progressively discharges because of application processing and capacitor leakage, until the device persists the state before entering the charging phase, as shown in Fig. 1. However, because $a_i(V_{a,i}(t)) = 0, t > t'$, that is, there is no energy arriving at node i later than t', $C_i(t)$ never reaches V_{on} again, and node i never resumes.

The attacker can access to the application's source code or reverse-engineer from binaries [58]. Codebases for battery-less IoT systems, including those used in real deployments [18, 1, 23], are often public, including operating system layers [5] and hardware drivers [15], while compilers [26] often require the entire source code to perform full-program optimizations.

Example attacks. Existing literature reports three specific instances of energy attacks [49]. These create situations of livelock, denial of service, and priority inversion. For illustration, we show how an existing vulnerability leads to to the

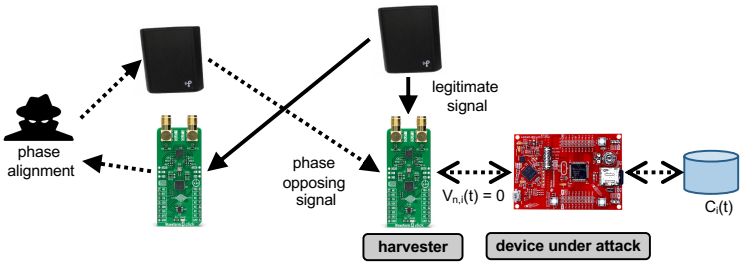


Fig. 3. Generating an opposing RF energy signal.

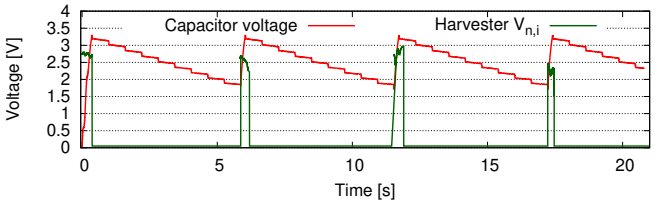


Fig. 4. Livelock situation caused by an energy attack.

former situation, using a TI MSP430FR5969 Launchpad running HarvOS [11], an existing checkpointing system for intermittent computing, and two Powercast transmitter-receiver pairs. The same situation can be achieved, for example, by controlling the incident solar radiation [12].

Existing intermittent systems recommend setting the activation threshold V_{on} by striking a balance between charging times and energy content when the system is at V_{on} [4, 11, 20, 31, 54]. The former suggests a lower V_{on}, whereas the latter pushes for a higher V_{on}. In most existing systems [4, 11, 54], V_{on} is statically set before deployment and does not necessarily guarantee that the energy content is sufficient to make progress in the application logic and persist the state whenever necessary. The ambient is indeed supposed to provide some energy also during the active times [12].

An attacker may systematically block the energy source at a node i while the device is computing, that is, he creates a transformation a such that $a_i(V_{a,i}(t)) = 0, t' > t > t''$, where t' and t'' are the points in time where the system reaches V_{on} and V_{off}, respectively. Concretely, while the first Powercast transmitter normally powers the device, the attacker implements function a by generating an opposing signal with a second Powercast receiver-transmitter pair, as shown in Fig. 3. This may be achieved by generating a signal in phase opposition [49] with the regular one, yielding destructive interference [43, 50]. The attacker thus cancels out the energy contribution of the legitimate Powercast transmitter.

Fig. 4 shows an example execution once the phase alignment is achieved. Without any contribution of energy during active times and a V_{on} setting that does not account for this, the system approaches V_{off} with insufficient energy to persist state, that is, no new checkpoint is created. When the system reaches V_{on} again, HarvOS resorts to the previous checkpoint, that is, the one that does not include the progress achieved between t' and t''. The previous operations are

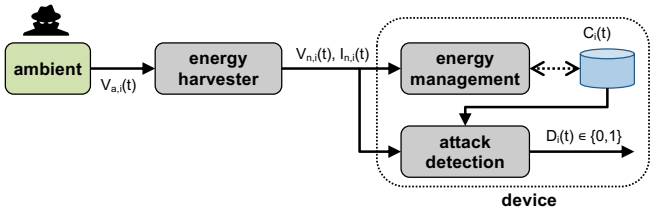


Fig. 5. Detection system architecture.

then executed again, and with the attacker replaying the same function a once more, the system approaches V_{off} again with insufficient energy to persist the state. As long as the attacker keeps doing so, the system continues to restart from the same checkpoint, making no progress in the long run.

Note that in the case of RF energy harvesting, this attack may occur not just without physical access to the device, but also without being anywhere close to it. As long as the attacker is in the (wireless) range of both the legitimate Powercast transmitter and of the device under attack, as shown in Fig. 3, she may detect both the incoming energy wave and regular network packets.

4 Attack Detection

We study the problem of *detecting* energy attacks when they are not as simple as completely and suddenly stopping the system. This means understanding when ambient energy does not follow the expected patterns, which is a case of anomaly detection [17]. Most deployments of IoT battery-less devices operate in areas with little to no opportunities for external instrumentation [1, 18, 23, 33]; for example, to install systems to detect physical intrusion of the attacker using surveillance cameras. We are, therefore, to meet specific requirements:

R1 *be accurate*, that is, minimizing false positives and false negatives so a device can rely on factual information;
R2 *operate online with low latency*, as countermeasures may be effective only in the short term;
R3 *run locally on the IoT device* due to the excessive energy consumption and additional latency that offloading the process to a third party would incur.

4.1 Design Space

Fig. 5 shows the detection system architecture. The detection process runs periodically, every T. We indicate with $D_i(t) \in \{0, 1\}$ a binary indicator that corresponds to whether the attack detection system thinks that an energy attack is ongoing at time t, with $t = kT$, $k \in \mathbb{N}$.

Regardless of how to detect energy attacks, the inputs to the detection system at node i may only be voltage $V_{n,i}(t)$, current $I_{n,i}(t)$, and capacitor charge $C_i(t)$. This is because the device has no information about ambient energy at node i, indicated as $V_{a,i}(t)$ in Fig. 4, before its transformation into usable energy by

the harvesting mechanism. This means that whenever an attack described as a function $a_i(V_{a,i}(t))$ occurs, the detection system cannot observe the attack directly, but it only has access to its effects after the harvesting processing.

Of the requirements above, **R1** is common [17]; **R2** excludes techniques that require the anomaly to stabilize in the long term or even to stop occurring before they can return an indication that it did happen. Most difficult, however, is **R3**. It rules out anomaly detection methods that demand large memories [24], significant processing power [39], or based on sharing information among nodes. It demands reducing energy consumption. The ideal solution would be one that does not cause additional energy failures, which may require persisting state and indirectly cause an overhead that would not be present otherwise.

We base our solution on *approximate intermittent computing* [9, 59] and develop an approximate support vector machine (ASVM) for detecting energy attacks. Approximate intermittent computing provides a knob to trade energy consumption for accuracy. We elect to use support vector machines because of their accuracy for binary classification [14, 53] and their optimal trade-off between accuracy of classification and resource consumption [10, 22].

4.2 ASVM for Attack Detection

An ASVM is a support vector machine trained as a regular SVM, but capable of using a subset of the signal features for inference [59, 9].

We train the ASVM using a total of 52 features by sampling every T the input signals $V_{n,i}(t)$, $I_{n,i}(t)$, $C_i(t)$ for a given ambient energy source, and combinations thereof. Example features include statistical parameters like averages and standard variations, up to various maximum likelihood estimators. The features are computed over different sliding windows over the past $wT, w \in \mathbb{N}$ time instants, which we tune based on the energy source. For example, in case of solar radiation, we use different windows to account for the behavior of the energy source over the last 24 h and the last 10 min. Training happens on a regular machine using the SVM library of the `ScyPy` package.

If we were to run the resulting model as a regular SVM, it may happen that running the energy attack detection step becomes the cause of an energy failure, prompting the system to dump the state on NVM to resume the work once energy is newly available. The energy required for NVM operations is significant and even if it is not directly required for detecting energy attacks, its overhead is in fact indirectly caused by that process exceeding the available energy.

To mitigate these occurrences, we profile the energy required for computing each of the 52 features, including the cost for hardware interaction, and analytically study the contribution to the overall accuracy each of them provides [3]. Next, we order the 52 features according to the ratio between their contribution to the resulting accuracy and their energy cost. This means that features that come first are those with the lowest cost per unit of contributed accuracy.

When the detection system runs, we probe the capacitor for the energy level and determine the first m features, out of the total 52 features, we can afford for classification without causing an energy failure. As probing the capacitor is

required anyways, the energy cost for doing so is not considered in the energy cost profile of any of the features. We then compute the classification incrementally and return the classification determined using $m \leq 52$ features. By placing the call to the energy attack detection system as last in an application period, we ensure that the attack detection process only runs with the energy "leftovers", without ever causing an energy failure that would not be there already.

5 Evaluation

Our evaluation is entirely based on real-world energy traces and accounts for 500K+ data points. The analysis is three-pronged. In Sec. 5.2, we evaluate the accuracy and system performance of the ASVM by generating energy attacks in a synthetic manner, that is, by varying the statistical parameters representing how an attacker manipulates the energy signal. This is instrumental to assess the general behavior of our design in a multitude of conditions. In Sec. 5.3, we concentrate on three specific attacks with existing experimental evidence [49]. Sec. 5.4 measures the performance against attacks whose patterns are never seen before, testing our design's ability to tackle previously unseen scenarios.

5.1 Setting

Because of the highly non-deterministic behavior of energy sources, achieving perfect reproducibility when evaluating energy-harvesting systems is challenging using real devices [5]. We thus opt for system emulation over hardware-based experimentation. Still, in the specific case of Sec. 5.3, we manage to run experiments with a *real device and energy harvester*.

Setup. We use the custom Siren MSP430 emulator [25]. We extend the tool with a model of 64 Kbyte of FRAM NVM next to a 2 Kbyte SRAM space, corresponding to the memory layout of the MSP430-FR5969, often employed in intermittent computing [44, 54, 61]. We account for the energy consumption per clock cycle of various operating modes of the MSP430-FR5969, such as regular computation, non-volatile/volatile memory operations, and I/O.

We simulate a paradigmantic sense-process-transmit IoT application [5] and use HarvOS [11] for checkpointing. We do not emulate sensors and radio, but the time and energy overhead for both are synthetically accounted for at the end of every application round. Note that the application processing is orthogonal to the energy attack detection and useful only to quantify the relative overhead of the energy attack detection system. In our case, we insert a call to the energy attack detection system at the end of every application round, thus generating the highest relative overhead.

Metrics and traces. We compute three key metrics: *i) the percentage of false negatives (positives)* measures the accuracy of the detection process as the fraction of energy attacks that are definitely missed (wrongly detected); *ii)* the *time to detection* is the number of application rounds between attack injection and when the detection system signals the attack, which measures how rapidly a

given technique realizes that the signal represents an anomaly[1]; and *iii)* the relative *energy overhead* as the additional energy consumption due to the energy attack detection system, compared with the application alone.

We feed the emulator with twelve diverse energy traces, offering a mixture of energy source, harvesting technology, and setting. In the following, the terms "indoor/outdoor" coupled with "static/mobile" refer to the location and mobility of the harvester unit, respectively. The M-RF trace is from Mementos [54] and is recorded using a Powercast transmitter [55]. Four traces are from EPIC [3] and are recorded using a mono-crystalline solar cell in settings including outdoor mobile (E-SOM), indoor mobile (E-SIM), outdoor static (E-SOR), and indoor static (E-SIR). The M-VIB and M-TEG traces are obtained using a ReVibe modelD kinetic energy harvester and a Thermalforce 254-150-36 thermoelectric energy harvester [1], respectively. The remaining traces are from the development of the Bonito protocol [27] and account for scenarios involving solar cells (B-JOG and B-OFI) and piezoelectric (B-STA, B-CAR, B-WAS) harvesters.

We use a 75/25 split between training and test data for every trace [47]. During each experiment, we inject no attack for 0.5 h of simulated time to let estimators stabilize. Experiments last between 12 h and 24 h of simulated time.

Baselines. We compare the performance of the ASVM in detecting energy attacks against an equivalent SVM that uses the same model with the regular inference step, called Regular SVM, as well as two additional baselines.

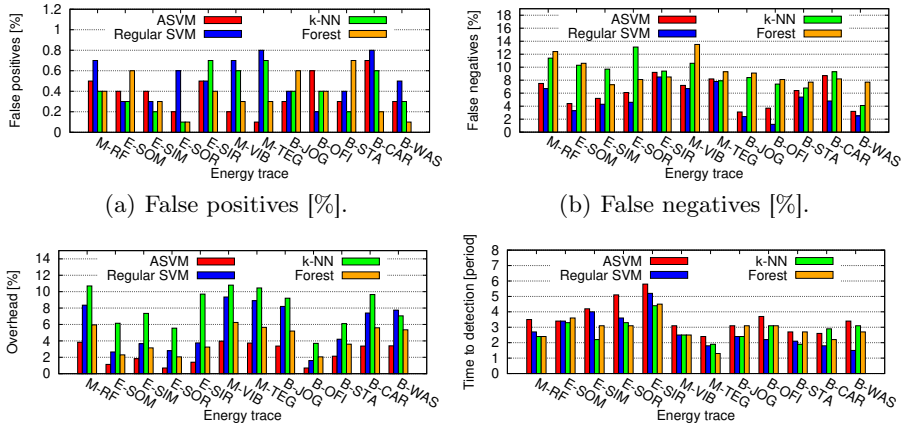
One baseline is the k-nearest neighbors algorithm (k-NN) [47], together with hyperparameter optimization to select k. k-NN is often used for anomaly detection of time series [17]. As in existing work [17], we obtain the classification by majority vote across the 52 features. The other baseline is called isolation forest (Forest) and is known to provide accurate anomaly detection with a linear time complexity and very limited memory consumption [41].

5.2 Synthetic Attacks

We inject energy attacks by considering the original trace as a signal $V_{n,i}(t)$ with mean μ and variance δ and by manipulating the latter two, changing the mean as $\mu' = k\mu$ and/or the variance as $\delta' = h\delta$, with $0 \leq k, h; k < 1; h < 2$; $h, k \in \mathbb{Q}$. Intuitively, $h > 1$ means increasing the randomness in the signal. Attack occurrence is drawn from a Poisson distribution with arrival rate λ of one every 10 minutes; the duration is drawn from a Normal distribution with mean 3 min. These parameters model existing energy attacks [49], which are proven to be successful. The application period is one minute.

Results. Fig. 6 shows the results we obtain as a function of the energy trace. The percentage of false positives, shown in Fig. 6(a), is limited across all designs and energy traces, never even reaching 1% of the cases. This is reassuring: in the presence of a mitigation technique, the price to pay for counteracting an energy attack would rarely be paid unnecessarily.

[1] False negatives are excluded from aggregate statistics.



(a) False positives [%].

(b) False negatives [%].

(c) Energy overhead [%].

(d) Time to detection [periods].

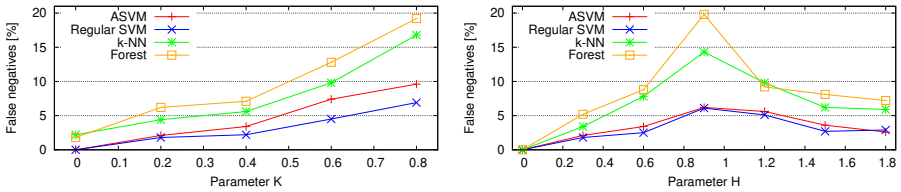
Fig. 6. Performance as a function of energy trace.

Different considerations apply to the results for false negatives, show in Fig. 6(b). The performance of both SVM-based solutions is markedly better than the other two baselines for most of the energy traces. The only exceptions are the E-SOR and B-STA traces, obtained using solar cells in an indoor setting, where the performance is comparable among all designs. The chances that temporary occlusions of the solar cells occur in this setting, for example, because of people passing by, is much higher than elsewhere. It is difficult to separate these legitimate energy variations from short-lived energy attacks.

Crucially, Fig. 6(b) demonstrates that the loss of accuracy, measured in terms of false negatives, due to using fewer features in ASVM is extremely limited compared to the Regular SVM. In the worst case, shown by the B-OFI trace, the ASVM is only 2.4% less accurate than the Regular SVM. In return, the ASVM imposes between half and one third of the energy overhead of the Regular SVM, as shown in Fig. 6(c). Two factors concur to this result: *i)* the ASVM uses first the features that contribute the most to an accurate result; and *ii)* the ASVM is inherently adaptive: if sufficient energy is available, it uses all available features and the performance is the same as the Regular SVM. The energy overhead of the other baselines tends to be higher than ASVM, with Forest generally outperforming k-NN due to the linear time complexity.

Fig. 6(d) shows, on the other hand, that both SVM-based approaches take slightly longer to detect attackes compared to the other baselines. On average, ASVM (Regular SVM) takes roughly 12% (8%) additional time. Given the absolute numbers at hand, however, this additional time rarely corresponds to more than one application period. Provided the attack is eventually detected, this means the application spends limited time without being aware of that.

Given the accuracy and energy performance of the ASVM, the additional time to detect an attack is a fair price to pay, given that the other two baselines are slightly faster to detect an attack if they do detect one, but often miss the



(a) False negatives with varying k, $h = 1$. (b) False negatives with varying h, $k = 1$.

Fig. 7. Percentage of false negatives as a function of k and h.

detection altogether, as shown earlier in Fig. 6(b). The ASVM is slightly slower, but significantly more accurate, that is, "better (slightly) late than never".

Fig. 7 provides a different view on the results, plotting the percentage of false negatives depending on k and h. We concentrate solely on the false negatives because the false positives are limited, as discussed before, whereas energy overhead and time to detection are largely independent of k and h. Fig. 7(a) shows that both SVM-based designs consistently outperform the baselines regardless of k and h. The increasing trend is due to attacks with smaller k being easier to detect: the smaller the k, the more different is the manipulated signal compared to the original one. The corner case is with $k = 0$, which corresponds to completely zeroing the energy signal, for example, modeling an occlusion of a solar cell, which all designs recognize accurately. The more k approaches 1, that is, the closer is the manipulated signal to the original one, the wider is the gap between the SVM-based designs and the baselines.

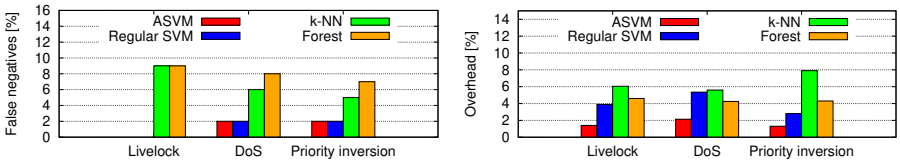
Similar considerations apply to Fig. 7(b), plotting the percentage of false negatives as a function of h. Again, the more h approaches 1, which means the more the manipulated signal is similar to the original one, the less accurate is the detection. The gap between the ASVM and either k-NN or Forest is largest precisely around $h \approx 1$, which is the most difficult setting.

5.3 Concrete Attacks

We verify the performance of the ASVM against the three specific attacks experimentally reported in the literature [49]. We use the same configuration and real hardware as in Sec. 3 and examine 22 instances of the attack leading to a livelock. For attacks leading to denial of service and priority inversion, we generate 100 instances each using the same setup and traces of Sec. 5.1 as input.

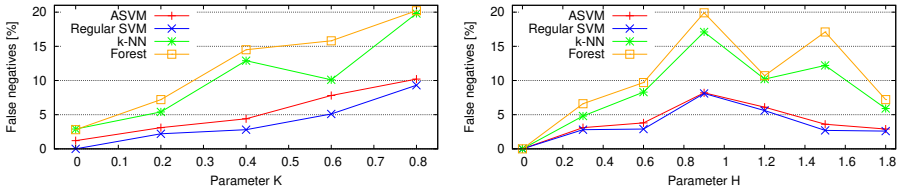
There is no guarantee that the attacks we generate are successful. By monitoring the signal $V_n(t)$ during the experiments, we measure that 16 instances (out of 22) are successful for the livelock attack, 88 instances (out of 100) are successful in the case of denial of service attack, and 38 instances (out of 100) are successful for the priority inversion attack. This models a realistic setting where one attempts to exploit a vulnerability, but does not always succeed.

Results. Fig. 8 shows the results. The trends of Sec. 5.2 are confirmed. False positives, not shown for brevity, top to 3% of the cases, and yet the ASVM, despite the approximate processing, returns no false positives at all. Crucially, Fig. 8(a)



(a) False negatives [%]. (b) Energy overhead [%].

Fig. 8. Performance for three concrete attacks [49].



(a) False negatives with varying k, $h = 1$. (b) False negatives with varying h, $k = 1$.

Fig. 9. Percentage of false negatives as a function of parameter k and h, when using non-overlapping parameter settings for training data and test data.

shows that the ASVM has the exact same accuracy as the REGULAR SVM. We conclude that the additional features processed by the REGULAR SVM are not sufficient to sway the classification, and hence the related processing entirely represents unnecessary overhead. The impact of the unnecessary overhead shows in Fig. 8(b), where the ASVM outperforms all other designs, with FOREST being the second most efficient technique as seen in Fig. 6(c). The time to detection for the ASVM, not shown for brevity, is comparable to K-NN and FOREST, and only the REGULAR SVM performs 17% better in the case of denial of service and prority inversion, while being roughly as fast for the livelock attack.

5.4 Unknown Attacks

We evaluate what accuracy we may obtain against unknown problem instances. We repeat the experiments of Sec. 5.2 with a different split between training data and test data. We use as training set the instances with $k \in \{0.2, 0.6\}$ and $h \in \{0.6, 1.2, 1.8\}$. All other parameter setting for either k or h are only used to generate test data. This has two effects: *i)* it reduces the size of the training data, and *ii)* the designs we test are confronted with unseen patterns of the energy signal, which might be legitimate or represent attacks.

Results. Fig. 9 shows the results. For the reasons explained earlier, we focus on the percentage of false negatives as a measure of accuracy. Compared to Fig. 7, the absolute values are generally larger, likely because of the reduction in the size of the training data, yet the trends remain largely the same. The observations we outline earlier, especially on the limited loss of accuracy of the ASVM compared to the REGULAR SVM, do remain.

Most importantly, the parameter settings that do not appear in the training set represent no particular outlier in Fig. 9 for either of the SVM-based designs.

This is not the case for the other baselines, as seen in Fig. 9(a) when $k = 0.4$ and in Fig. 9(b) when $h = 1.5$. This provides evidence of the general robustness of our design also when facing previously unseen problem instances.

6 Inspiring Defense

Designing defense techniques is a manifold problem, whose implications possibly percolate through both software and hardware layers. We discuss next key dimensions of the design space, hopefully inspiring follow-up work in the area.

Consider a scenario where *energy* is the major concern, hence the ratio between energy consumed and useful work is to be minimized. Defense techniques may be developed in this scenario by applying mixed-criticality concepts [16], that is, by splitting code functionality or application tasks in critical and non-critical ones. During an attack, the latter may be suspended, thus shifting the reduced energy budget towards critical functionality.

Say the amount of *collected data* is crucial, regardless of how energy is spent. One may employ concepts of context-oriented programming, which also exist for low-power embedded systems [2], to dynamically change the application behavior. For instance, the system may temporarily log data locally instead of using wireless transmissions, should the latter be affected by the energy attack.

One may design techniques that mitigate particular negative effects. These require an additional step to identify the type of attack. Examples are techniques that mitigate attacks preventing a device from making progress, that is, akin to the example attack of Sec. 3. Recognizing this kind of attack may be achieved by instrumenting the code with intermittence-aware programming constructs [45]. Adapting techniques that dynamically adjust the activation threshold [8] may provide a way to resolve the livelock.

On the other hand, we argue that generally-applicable defense techniques may take inspiration from energy management in mobile devices [32]. Mobile operating systems feature sophisticated techniques to handle situations of energy scarcity, for example, when the battery is about to run out. These include tuning a number of hardware knobs, which are normally not available on resource-constrained embedded platforms, as well as software techniques such as throttling the execution of a subset of system processes. The latter effectively represent a generalization of many of the attack-specific techniques we discuss. These techniques may be adapted to intermittently-computing devices.

The defense techniques outlined above are mainly implemented in software. A natural option at the hardware level is to rely on multiple energy sources. Existing hardware platforms [31] and deployed battery-less IoT systems [1, 18, 33] seldom rely on multiple energy sources, yet prototypes exist [42]. Combining energy-rich sources that may be attack vectors, with energy-poor sources that are exceedingly difficult to employ as attack vectors, may provide an effective combination to sustain long-term operation in regular conditions, while mitigating the effects of energy attacks when they occur.

7 Conclusion

We presented a technique to detect energy attacks that is accurate, timely, and imposes a limited energy overhead, enabling detection on resource-constrained IoT devices. Our ASVM design combines machine learning and approximate intermittent computing concepts. Our evaluation, entirely based on real-world energy traces, shows that the ASVM detects energy attacks with 92%+ accuracy, that is, up to 37% better than the baselines, and with up to one fifth of their overhead. We concluded with directions to inspire defense techniques.

References

1. Afanasov, M., et al.: Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In: Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SENSYS) (2020)
2. Afanasov, M., Mottola, L., Ghezzi, C.: Context-oriented programming for adaptive wireless sensor network software. In: IEEE International Conference on Distributed Computing in Sensor Systems (2014)
3. Ahmed, S., et al.: The betrayal of constant power × time: Finding the missing joules of transiently-powered computers. In: International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES) (2019)
4. Ahmed, S., et al.: Efficient intermittent computing with differential checkpointing. In: Proceedings of the ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES) (2019)
5. Ahmed, S., et al.: The Internet of Batteryless Things. Commun. ACM (2024)
6. Alajlan, N.N., Ibrahim, D.M.: Tinyml: Enabling of inference deep learning models on ultra-low-power iot edge devices for ai applications. Micromachines (2022)
7. Asad, H.A., et al.: On securing persistent state in intermittent computing. In: Int. Workshop on Energy Harvesting and Energy-Neutral Sensing Systems (2020)
8. Balsamo, D., et al.: Hibernus++: A self-calibrating and adaptive system for transiently-powered embedded devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2016)
9. Bambusi, F., et al.: The case for approximate intermittent computing. In: International Conference on Information Processing in Sensor Networks (IPSN) (2022)
10. Banbury, C.R., et al.: Benchmarking TinyML systems: Challenges and direction. arXiv preprint 2003.04821 (2020)
11. Bhatti, N.A., Mottola, L.: Harvos: Efficient code instrumentation for transiently-powered embedded sensing. In: International Conference on Information Processing in Sensor Networks (IPSN) (2017)
12. Bhatti, N.A., et al.: Energy harvesting and wireless transfer in sensor network applications: Concepts and experiences. ACM Trans. on Sensor Networks (2016)
13. Borghoff, J., et al.: PRINCE–a low-latency block cipher for pervasive computing applications. In: International Conference on the Theory and Application of Cryptology and Information Security (2012)
14. Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: Proceedings of the Workshop on Computational Learning Theory (1992)
15. Branco, A., et al.: Intermittent asynchronous peripheral operations. In: International Conference on Embedded Networked Sensor Systems (SENSYS) (2019)

16. Burns, A., Davis, R.I.: A survey of research into mixed criticality systems. ACM Computing Surveys (CSUR) **50**(6) (2017)
17. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) **41**(3) (2009)
18. Chen, Q., et al.: Harvest energy from the water: A self-sustained wireless water quality sensing system. ACM Trans, on Embedded Computing Systems (2017)
19. Chen, S., et al.: Power attack and detection technology in data centers: A survey. In: International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI) (2020)
20. Colin, A., et al.: A reconfigurable energy storage architecture for energy-harvesting devices. In: International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2018)
21. da Costa, K., et al.: Internet of things: A survey on machine learning-based intrusion detection approaches. Computer Networks (2019)
22. David, R., et al.: Tensorflow lite micro: Embedded machine learning for TinyML systems. Proceedings of Machine Learning and Systems **3** (2021)
23. Denby, B., et al.: Kodan: Addressing the computational bottleneck in space. In: International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2023)
24. Dietterich, T.G.: Ensemble methods in machine learning. In: International Workshop on Multiple Classifier Systems. Springer (2000)
25. Furlong, M., et al.: Realistic simulation for tiny batteryless sensors. In: Int. Workshop on Energy Harvesting and Energy-Neutral Sensing Systems (ENSSYS) (2016)
26. Gay, D., et al.: The nesC language: A holistic approach to networked embedded systems. Acm Sigplan Notices (2003)
27. Geissdoerfer, K., Zimmerling, M.: Learning to communicate effectively between battery-free devices. In: USENIX Symposium on Networked Systems Design and Implementation (NSDI) (2022)
28. Ghodsi, Z., et al.: Optimal checkpointing for secure intermittently-powered iot devices. In: International Conference on Computer-Aided Design (ICCAD) (2017)
29. Greenberg, A., Hamilton, J., Maltz, D.A., Patel, P.: The cost of a cloud: research problems in data center networks (2008)
30. Grisafi, M., Ammar, M., Yildirim, K.S., Crispo, B.: MPI: memory protection for intermittent computing. IEEE Trans. on Information Forensics and Security (2022)
31. Hester, J., Sorber, J.: Flicker: Rapid prototyping for the batteryless Internet of Things. In: Int. Conference on Embedded Network Sensor Systems (2017)
32. Hoque, M.A., et al.: Modeling, profiling, and debugging the energy consumption of mobile devices. ACM Computing Surveys (CSUR) (2015)
33. Ikeda, N., et al.: Soil-monitoring sensor powered by temperature difference between air and shallow underground soil. International Conference on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT) (2020)
34. Jackson, N., et al.: Capacity over capacitance for reliable energy harvesting sensors. In: Int. Conference on Information Processing in Sensor Networks (IPSN) (2019)
35. Kanwar, J., et al.: JamSense: Interference and jamming classification for low-power wireless networks. In: Wireless and Mobile Networking Conference (WMNC) (2021)
36. Krentz, K.F., et al.: Countering three denial-of-sleep attacks on ContikiMAC. In: Int. Conference on Embedded Wireless Systems and Networks (EWSN) (2017)
37. Krishnan, A., Schaumont, P.: Exploiting security vulnerabilities in intermittent computing. In: SPACE International Conference (2018)
38. Krishnan, A.S., Schaumont, P.: Benchmarking and configuring security levels in intermittent computing. ACM Trans. on Embedded Computing Systems (2022)

39. Langley, P., Iba, W., Thompson, K., et al.: An analysis of bayesian classifiers. In: International AAAI Confernce on Artificial Intelligence (1992)
40. Li, C., et al.: Power attack defense: Securing battery-backed data centers. ACM SIGARCH Computer Architecture News (2016)
41. Liu, F.T., et al.: Isolation forest. In: Int. Conference on Data Mining (2008)
42. Liu, H., et al.: Hybrid energy harvesting technology: From materials, structural design, system integration to applications. Renewable and sustainable energy (2021)
43. Liu, Q., et al.: Safe and secure wireless power transfer networks: Challenges and opportunities in RF-based systems. IEEE Communications Magazine (2016)
44. Maeng, K., et al.: Alpaca: Intermittent execution without checkpoints. Proceedings of the ACM Programming Languages (2017)
45. Maioli, A., Mottola, L.: Intermittence anomalies not considered harmful. In: Int. Workshop on Energy Harvesting and Energy-neutral Sensing Systems (2020)
46. Maioli, A., Mottola, L.: Alfred: Virtual memory for intermittent computing. In: Int. Conference on Embedded Networked Sensor Systems (SENSYS) (2021)
47. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of machine learning. MIT press (2018)
48. Mottola, L., et al.: Enabling scope-based interactions in sensor network macroprogramming. In: Int. Conf. on Mobile Ad-hoc Sensor Systems (MASS) (2007)
49. Mottola, L., et al.: Energy attacks in the Battery-less Internet of Things: Directions for the future. In: European Workshop on Systems Security (2024)
50. Naderi, M.Y., et al.: RF-MAC: a medium access control protocol for re-chargeable sensor networks powered by wireless energy harvesting. IEEE Transactions on Wireless Communications (2014)
51. Nguyen, V.L., Lin, P.C., Hwang, R.H.: Energy depletion attacks in low power wireless networks. IEEE Access **7** (2019)
52. Portilla, J., et al.: Adaptable security in wireless sensor networks by using reconfigurable ecc hardware coprocessors. International Journal of Distributed Sensor Networks (2010)
53. Pradhan, A.: Support vector machine-a survey. International Journal of Emerging Technology and Advanced Engineering **2**(8) (2012)
54. Ransford, B., et al.: Mementos: System support for long-running computation on rfid-scale devices. ACM SIGARCH Computer Architecture News (2011)
55. Sample, A., et al.: Design of an RFID-based battery-free programmable sensing platform. IEEE transactions on instrumentation and measurement (2008)
56. Tahsien, S.M., et al.: Machine learning based solutions for security of Internet of Things (IoT): A survey. Journal of Network and Computer Applications (2020)
57. Thakor, V.A., et al.: Lightweight cryptography algorithms for resource-constrained IoT devices: A review, comparison and research opportunities. IEEE Access (2021)
58. Udupa, S.K., et al.: Deobfuscation: Reverse engineering obfuscated code. In: Working Conference on Reverse Engineering (WCRE) (2005)
59. Umesh, S., Mittal, S.: A survey of techniques for intermittent computing. Journal of Systems Architecture **112** (2021)
60. Valea, E., et al.: SI ECCS: SECure context saving for IoT devices. In: Int. Conference on Design & Technology of Integrated Systems In Nanoscale Era (2018)
61. Van Der Woude, J., Hicks, M.: Intermittent computation without hardware support or programmer intervention. In: Int. Conference on Operating Systems Design and Implementation (OSDI) (2016)
62. Yildirim, K.S., et al.: InK: Reactive kernel for tiny batteryless sensors. In: Int. Conference on Embedded Networked Sensor Systems (SENSYS) (2018)