Interruptible Remote Attestation of Low-end IoT Microcontrollers via Performance Counters

DAVIDE LI CALSI, Chair of Theoretical Information Technology, Technical University of Munich (TUM), Munich, Germany, Germany
VITTORIO ZACCARIA, DEIB, Politecnico di Milano, Milano, Italy, Italy

Remote attestation is a method used in distributed systems to detect integrity violations on a target device (prover) through a challenge–response protocol initiated by a verifier device. The prover calculates a hash of its memory, which is compared to a known good state hash by the verifier. We propose a novel technique, called Counters Help Against Roving Malware (CHARM), which uses hardware performance counters on the prover's side and machine learning on the verifier's side to make interruptible remote attestation feasible, even for constrained microcontrollers. We will demonstrate the effectiveness of various machine learning tools and data manipulation techniques on prediction accuracy in a variety of scenarios.

CCS Concepts: • **Security and privacy** → **Trusted computing**; **Distributed systems security;**

Additional Key Words and Phrases: Security, integrity, remote attestation, machine learning, interrupts, microcontrollers, performance counters

ACM Reference format:
Davide Li Calsi and Vittorio Zaccaria. 2023. Interruptible Remote Attestation of Low-end IoT Microcontrollers via Performance Counters. *ACM Trans. Embedd. Comput. Syst.* 22, 5, Article 87 (September 2023), 19 pages.
https://doi.org/10.1145/3611674

1 INTRODUCTION

In the context of distributed systems, *remote attestation* (RA) allows one to remotely detect integrity violations on a target device (or *prover*) through a challenge-and-response protocol initiated by a *verifier* device. The protocol involves the computation of a hash h_m of the prover's memory m, which is then compared against a known *good state* hash h by the verifier (Figure 1, conventional remote attestation). To guarantee the good faith of such a hash, the prover's device must disable interrupts when computing it. Disabling interrupts during remote attestation can help ensure the integrity of the attestation process. In fact, when interrupts are enabled, the attestation process can be interrupted and run concurrently with other tasks. This can potentially compromise the accuracy of the attestation by changing the state of the system being attested. For example, suppose that a prover is computing a hash of its memory as part of a remote attestation protocol. If interrupts are enabled, then it is possible for an interrupt to occur and modify the memory that is

Authors' addresses: D. Li Calsi, Chair of Theoretical Information Technology, Technical University of Munich (TUM), P. O. Box 1212, Munich 80333, Germany; email: davide.li-calsi@tum.de; V. Zaccaria, DEIB–Politecnico di Milano, Piazza Leonardo da Vinci 32, Milano 20133, Italy; email: vittorio.zaccaria@polimi.it.

1539-9087/2023/09-ART87 $15.00
https://doi.org/10.1145/3611674

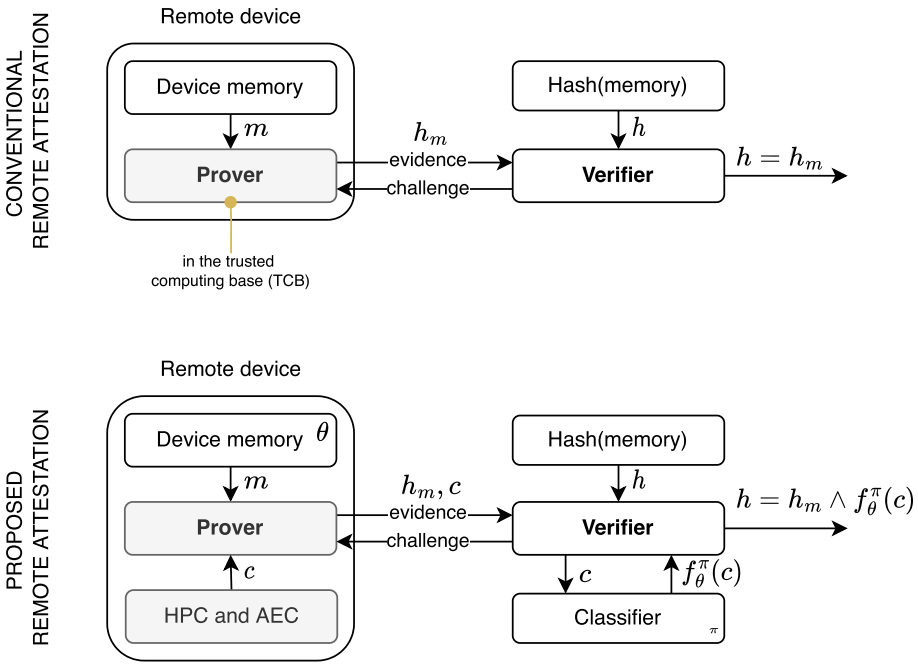

Fig. 1. Conventional vs. proposed remote attestation architecture.

being hashed while the hash is being computed. If this happens, then the resulting hash will not accurately reflect the state of the system at the time the attestation was initiated. By disabling interrupts, the prover can ensure that the attestation process is not interrupted and that the computed hash accurately reflects the state of the system at the time the attestation was initiated.

The contribution of this work is to show that interruptible RA is feasible, even for the most constrained microcontrollers, by exploiting hardware performance counters (HPC) on the prover's side and machine learning on the verifier's side (Figure 1, proposed remote attestation). We will corroborate this assertion in a wide range of scenarios and show that while support vector machines (SVM) are the most effective verifier's side machine learning tool when only HPC counters are considered, application event counters (APE) can be used to increase the prediction accuracy of less-sophisticated models and that appropriate data manipulation techniques, such as Box-Cox transformation and data augmentation (DA), can be equally effective. This result is consistent with findings in Reference [30], which cautions against relying solely on hardware performance counters for malware detection due to a lack of causation between low-level micro-architectural events and high-level software behavior.

The structure of this article is as follows: First, we will present an overview of remote attestation and the classifiers we utilized in Section 2. Then, we will introduce CHARM in Section 3 and describe its validation in Section 4. In Section 5, we will provide an overview of current works related to our article. Finally, we will conclude in Section 6 by summarizing our results and suggesting potential avenues for future research.

2 BACKGROUND

In this section, we will introduce the basic mechanisms of remote attestation and the machine learning techniques we will use in our approach.

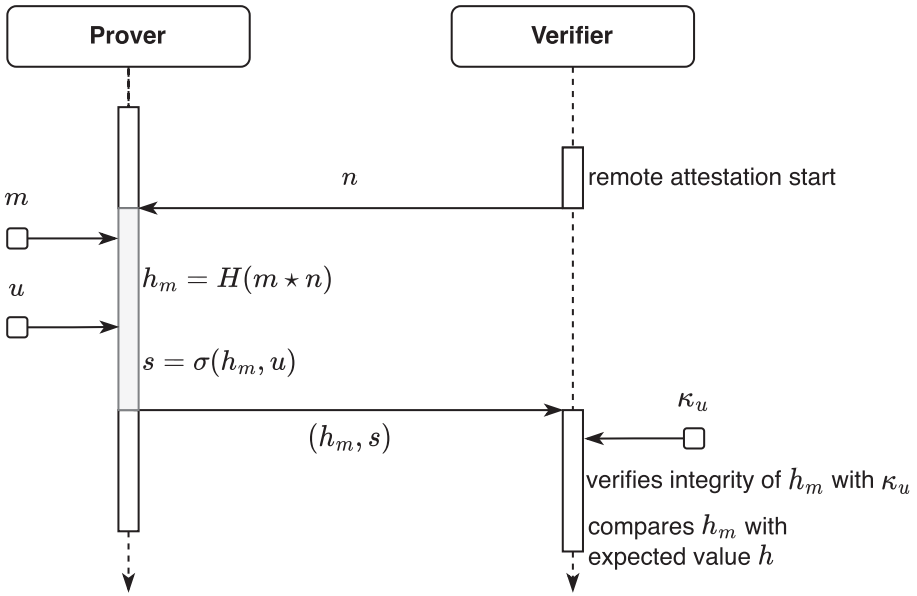

Fig. 2. Detailed overview of remote attestation. u and κ_u can be private/public keys (asymmetric case) or the same key (symmetric case).

2.1 Introduction to Remote Attestation

Remote attestation [6] is a challenge-and-response protocol that allows a remote party, the verifier, to attest the integrity of some device of interest, the prover. In our setting, the prover is a constrained device such as a simple microcontroller, but, in general, it can be any device whose security is at stake. However, the verifier has no constraints on its computational power and, in practice, it is often more computationally capable than the prover. Typically, the challenge of the attestation consists of the following steps (see Figure 2):

(1) To avoid time-of-check/time-of-use attacks, the verifier produces a nonce n that is sent to the prover along with the attestation request.
(2) Upon receiving the request, the prover measures its own memory m by computing a digest H of the nonce n and the memory content m, producing evidence h_m. This is typically done by the trusted computing base (TCB). Attested memory m can be program data and code (dynamic RA) or just program code (static RA).
(3) To prevent tampering, the prover authenticates the evidence by computing a signature s through an authentication function σ and a key (unique to the device) u. Typically, such authentication functions are symmetric and σ computes a *message authentication code*.
(4) The verifier checks the integrity of h_m through s and k_u (or u in the case of symmetric methods).
(5) The verifier checks if h_m is the expected value h. Assuming that the legitimate configurations of the prover's memory form a small set, the verifier can precompute the expected response.

Over time, attestation shifted from a pure software mechanism to one increasingly supported by hardware or executed on the trusted computing base. We will leave the relevant discussion of the state of the art in the appropriate section.

2.2 The Binary Classifiers Used in This Work

For this work, we aim to develop a simple yet efficient method of distinguishing between good states from bad ones, with the assumption that offline supervised learning can be utilized by the verifier. Our objective is to achieve better interpretability, alleviate the burden associated with model selection (hyperparameter setup), and reduce the likelihood of overfitting by minimizing complexity. Consequently, we have opted for three binary classifiers that employ supervised learning: logistic regression, decision trees, and support vector machines. It should be noted that selecting these classifiers may also lead to lower power requirements for the verifier; however, this is not our primary focus.

Logistic regression (LR) maps an input x to the probability that x belongs to one of the two classes

$$y(x) = \sigma(w^T \phi(x)) \in [0, 1],$$

where $\sigma(\cdot)$ is the sigmoid function, w is a parameter vector, and $\phi(\cdot)$ is a feature expansion function. The result of logistic regression can be interpreted as the posterior probability p of the sample x belonging to one of the two classes (the probability of the second class being $1-p$). Training is typically geared toward maximizing posterior class probability, and the classifier can effectively work with linearly separable data. However, a simple linear decision boundary might not be enough for the problem at hand. For this reason, we decided to include in our analysis both decision trees and support vector machines, which provide tools to address nonlinear decision boundaries as well. Decision trees perform binary classification by splitting the dataset into parametric regions according to the information gain criterion. They are popular and powerful classifiers and have previously been shown to perform well when dealing with HPC [23]. For some practitioners, they provide better interpretability and insight into the data with respect to logistic regression. A nonlinear *support vector machine* (SVM) is a kernel method that labels x with $+1/-1$ according to the predicted class; this is expressed as

$$y(x) = \text{sign}\left(b + \sum_{i \in T} y_i a_i k(x, x_i)\right),$$

where T is the set of training pairs (x_i, y_i) (x_i are called *support vectors*), $k(\cdot, \cdot)$ is a kernel function, a_n is a weight typically found with the well-known margin maximization criterion, and b is a bias. Support vector machines differ from logistic regression, because they are not based on any probabilistic/statistic maximization but only on maximizing class separation. Practically speaking, they strive to provide just the right answer instead of the right probability.

We will evaluate the performance of the classifiers based on *precision* and *recall*: Recall is the ratio

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

where TP (TN) is the number of true positives (negatives) while FP (FN) is the number of false positives (negatives). Recall reflects the ability of our model to detect malware when it is present; thus, it can be used as an estimate of the probability of detection of malware. Precision is defined as the ratio

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

that measures the consistency of experimental positives with reality. High precision means a low percentage of false positives. We will integrate these two metrics into the so-called F1 score, i.e.,

the harmonic mean of precision and recall

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}.$$

In our work, we will also try to improve the performance of the classifiers by using two data pre-processing techniques: *data augmentation* and the Box-Cox transform. Data augmentation is a standard technique in machine learning that can drastically boost a model's performance in the case of data scarcity and under-represented classes. It consists of applying some class-preserving transformations to the feature vectors, thus generating some "artificial" input records. In our context, this is useful in the training phase, because running actual malware relocations can degrade the system due to the underlying technology of Flash memories.[1] Data augmentation can thus allow us to use the system more efficiently, eventually improving the representability of the malware. The Box-Cox transform is a family of data transformations of type

$$y \mapsto \begin{cases} \log y & \lambda = 0 \\ \frac{y^\lambda - 1}{\lambda} & \lambda \neq 0 \end{cases},$$

where λ is the family parameter. This transform is used to normalize the feature vector, improving the robustness of classifiers against variability.

2.3 Performance Counters in Low-end Microcontrollers

Performance Counters are often included in vendor implementations, as they are useful for debugging, application profiling, and power consumption monitoring. Cortex-M microcontrollers typically have a Data Watchpoint and Tracing Unit, which includes six low-level counters that count micro-architectural events. Some (Cortex-M55) even have a Power Management Unit, allowing for detailed monitoring of architectural events. These components can count branch-related events (branch taken, mispredicted, retired), cache hits or misses, bus accesses, and other events. Atmel microcontrollers may have programmable counters, which can be programmed by developers to monitor events of interest. ESP32 MCUs have similar Performance Counters and provide Performance Monitor APIs to manipulate them. These registers track CPU-specific events, such as stalls and branch counts. Although each vendor may have different micro-architectures, some core components and events are present in almost all architectures. For instance, branch instructions are likely to be present in all systems and can be monitored. Similarly, memory access, bus access, and cache hit/miss can be monitored across multiple platforms. However, some events are strongly platform dependent and cannot be defined in other architectures. An example is the number of folded instructions, which is monitored in ARM microcontrollers and is related to how ARM's architecture "folds" instructions into a single one.

3 CHARM: COUNTERS HELP AGAINST ROVING MALWARE

The majority of RA protocols require interrupts to be disabled to ensure the integrity of the attestation. In fact, if a malware is able to interrupt RA before it is detected, then it can successfully move to some previously examined memory area and cover its traces by writing benign code at its previous location. Some approaches try interruptible RA by exploiting special hardware mechanisms (EA-MPU, [7]) or by randomizing the attestation [10]. We believe that a different path is possible; in fact, even allowing interrupts, we can distinguish between malware attacks and benign activity by verifying the trace of events occurring during attestation itself and available in the platform

[1] These memories are designed to withstand only a limited number of write cycles per memory cells. For example, the flash memory used by our Cortex-M33 has an endurance of 10K cycles [25, p. 226].

counters. Previous work on counter-based malware detection techniques [5, 27] showed the feasibility of using performance counters to separate benign activity from malicious attacks. CHARM applies this idea to detect malicious relocations performed by roving malware. Our protocol introduces a *second-level attestation* that verifies the integrity of the attestation response itself. This higher-level attestation uses the values of the available counters as its integrity evidence by grouping them into a counter vector c. Choosing a proper set of events allows this object to capture an informative history of what happened at attestation time. This information is then compared with previous knowledge about the benign activity of the system. This high-level goal translates in practice into a verifier-side binary classifier that was trained to draw a decision boundary between malicious and benign counter vectors. The classifier's result is the second-level attestation response, and its value is checked to determine whether trust in RA is broken. This idea requires CHARM to have a bipartite structure, with two phases,

- *Offline phase* - run several attestation routines with interrupts enabled. The prover-side application interrupts all the attestations, while malware only interrupts a subset. Furthermore, one must reset the available counters at the beginning of each attestation and collect their values at the end. This step must happen in the Trusted Computing Base[2] to avoid malicious resets tampering with the counters' vector. The corresponding vectors form a dataset D that one must use to train a classifier.
- *Online phase* (shown in the lower part of Figure 1) - perform static attestation on the Program Memory, but activate counters to capture a trace of the events that occurred. The collected counter vector c is attached to the attestation response h_m and forwarded to the verifier. The latter feeds c to the pre-trained classifier to compute the second-level attestation response $f_\theta^\pi(c)$. The final attestation outcome h is a combination of $f_\theta^\pi(c)$ and h_m's verification. If and only if both verifications are passed, then we can trust that the prover is in a good state. We represent this concept by using the synthetic notation

$$h = h_m \wedge f_\theta^\pi(c)$$

  to express the logical bond between h and the two partial responses.

3.1 Architecture of the System

The proposed architecture for remote attestation is characterized by two additional modules compared to the conventional architecture (Figure 1). On the remote device, we assume that the trusted computing base is equipped with the capability to measure hardware performance counters (HPC) and/or has been instrumented to measure application event counters (APE). On the verifier side, integrity verification is conditioned on the result of a *classifier* that, based on the counter value c, the characterization θ of the application, and some design-time configuration parameter π, can detect whether suspicious activity occurred while *measuring* the device memory itself (f_θ^π predicate). Many established architectures include hardware units that provide HPC, because they are relevant for other applications, such as debugging or profiling. APE do not require dedicated hardware for their measurement and hence are more flexible and implementable on any platform that has adequate protections. These mechanisms are already present in some micro-controllers from the Cortex-M family (i.e., TrustZone), and we are positive that advances in research will ease their deployment on low-end devices. As for the classifier, machine learning offers several possibilities when dealing with classification tasks. Furthermore, having a copy of the application running on the microcontroller is sufficient to collect enough information to train a statistical model. The

[2]On TrustZone-equipped systems, memory areas can be added to the Secure world. With the same mechanism, one can protect and securely manage peripherals.

method proposed is based on machine learning and therefore has a probabilistic nature; in other words, there exists a non-zero chance of false negatives. One might inquire whether probabilistic models, despite having a high success rate, could replace deterministic models for security checks. On the one hand, like other methods [10], if multiple measurements are taken, then the probability of overall false negatives can decrease exponentially. These measurements can be obtained through independent consecutive attestations instances or through batch mode where the verifier sends m challenges simultaneously and receives m measurements. On the other hand, it is important to note that this is always a tradeoff with expected overhead. If non-negligible probability of false negatives is not acceptable, then higher overhead methods (potentially resulting in service interruption) must be employed.

3.2 Threat Model Assumptions

The assumption of our threat model is that malware (whose presence could be identified with a simple attestation) could exploit interrupts to escape attestation if interrupts are enabled. An example could be a malware that copies itself (through low-level loads and stores) into an already-attested region of memory. Malware that can roam freely through memory is known as *roving malware*. The aforementioned type of roving malware is a *self-relocating* one, i.e., malware that writes its content to attested memory and then self-erases. Instead, *transient* malware only self-erases from memory, hoping to later re-infect the system. In both cases, the malicious loads and stores executed by the malware nullify RA's detection ability. Low-level loads and stores are continuously traced by HPC and could be used to detect such a threat. The difficulty lies in determining whether such actions come from legitimate applications or system components. Another degree of freedom to be considered is the size of the malware, which, given the target IoT targets, is between 20 and 50 KiB[3] [18]. These two orthogonal dimensions are relevant, because varying them affects the roving malware's impact on the counter vector. It is clear that larger malware samples require more operations to relocate. Similarly, self-relocating malware performs roughly twice more operations than transient malware (assuming equal size). Since both parameters are unknown at attestation time, CHARM must be able to detect roving malware of variable type and size. This constraint impacts the dataset construction in the offline phase. One can in principle relocate the same malware sample, but that will lead to a specialized model that only recognizes a very specific type of attack. For robust roving malware detection, one must relocate malware samples with variable sizes, both in self-relocating and transient fashion. Our threat model does not consider hardware attacks on the device, and we assume a trustworthy implementation of the available hardware protection mechanisms. We also assume that no cryptographic or side-channel attack on the employed cryptographic primitives is possible. These forms of attack can be successfully prevented via orthogonal countermeasures that do not affect CHARM.

3.3 The Prover/Verifier Design Space

Given a vector of hardware and event counter values $c \in \text{HPC} \times \text{APE}$, the classifier f is modeled as a Boolean predicate that validates the good faith of the attestation procedure given the value of the counters. This must be parameterized by two additional parameters θ and π that model, respectively, the application to be attested and the classifier to use

$$f_\theta^\pi : \text{HPC} \times \text{APE} \rightarrow \mathbb{B}$$

[3]This is an empirical bound. We have inspected some public malware repositories and established these thresholds by checking the found samples' sizes.

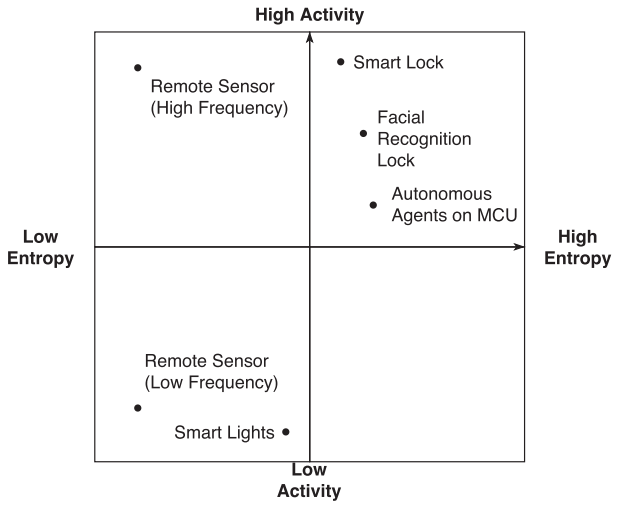

Fig. 3. Coarse-grained Entropy and Activity characterization for several Iot scenarios.

3.3.1 Application Characterization. $\Theta \ni \theta$ describes the nature of the application itself: Some stimuli on the hardware counters could be completely normal for certain applications and abnormal for others. This parameter is typically fixed once the application is known. Θ is the Cartesian product of two spaces[4]:

- Entropy level: the degree of unpredictability of the counters, measured as the Shannon's entropy, in four quantitative levels (very low, low, medium, high).
- Activity level: how much the performance counters are stressed by the applications, measured as the expected value of the counters (very low, low, medium, high).

Both parameters can facilitate or hinder the classification task, and each might be associated with a specific IoT attestation scenario (see Figure 3). Unpredictable behavior makes it harder to define a form of *normal behavior*, and hence higher Entropy levels lead to a drop in the classifier's detection rate. As for Activity level, applications that on average produce benign counter vectors close to malicious ones are much harder to separate from roving malware. As this average difference increases, computing a decision boundary becomes simpler and simpler.

3.3.2 Classifier Characterization. The parameter θ does not model the entire design space of our architecture. In fact, there is a potential family $\Pi \ni \pi$ of classifiers according to several degrees of design freedom. One degree of freedom is which counters to use. We distinguish between HPC, measured directly by the hardware and automatically trusted, and APE, which must be logged via software and potentially need an extension of the trusted computing base. HPC typically monitor low-level micro-architectural events, while APE measure high-level events related to the application running on the prover. The other degrees of freedom are the type of classifier and the data transformations applied to both training data and inference data. Given the low complexity requirements of target systems, we will consider rather unsophisticated classifiers such as *logistic regression* (LR), *decision trees*, and *support vector machines* (SVM) and the optional intervention of data augmentation (for training) and variance reducing transformations for inference data (Box-

[4]While one can define a significant amount of parameters characterizing the application, we restrict our analysis to the candidates that are more likely to influence CHARM's success.

Cox transform). Testing several classifiers is advisable due to the well-known *no-free-lunch* theorems, which indirectly prove that there is not a unique model that outperforms all the others in every problem. Additionally, although some models yield lower detection rates they could still be preferable in favor of higher and easier interpretability.

3.4 The Counters' Design Space

Our approach requires adequate hardware components to function, namely a set of HPC to monitor micro-architectural events and/or hardware protection to set-up and monitor some APE that track high-level events. Although not every vendor currently provides them, HPC are present in the majority of modern devices. Furthermore, current developments in hardware protections may facilitate the deployment of such components even on cheap microcontrollers, as you can already witness in modern devices (e.g., some Cortex-M microcontrollers). CHARM's effectiveness strongly depends on the set of counters available to the classifier. Not only do they determine the latter's detection capability, but they also affect the prover-side power consumption and overhead. HPCs are the easiest option, as they are updated by definition via hardware and are available, in some form, on almost any microcontroller.[5] However, microarchitectural events might not provide all the meaningful information to determine a decision boundary between benign and malicious activity, which is only attainable by measuring application events as well. In fact, experimental evidence [28] suggests that low-level events alone might not be sufficient to discriminate between malicious and benign activity. While malware and benign software often differ at a microarchitectural level, there could be high-level differences that only high-level events capture. Such events can be measured through appropriate APEs and depend on the application itself.[6] However, introducing APEs is significantly more challenging. On the one hand, even for a fixed θ, one should explore a wide application-dependent design space in the search for an effective set of APE; on the other hand, additional obstacles can arise, because, being software-managed, attackers could easily manipulate APEs. For these reasons, one must find ways to extend the trusted computing base to measure these events, a capability that most microcontrollers lack at the moment. Furthermore, the software instructions for their update represent an additional cost, which we estimate coarsely in Section 4. Therefore, APE is not the silver bullet but rather a pricey optional that could be beneficial for some applications and unfeasible for others. Fortunately, some techniques, such as the aforementioned Box-Cox and DA, can be used to extract more information from simple HPCs, moving the computational burden to the verifier, representing a valid alternative to APE when high-level monitoring is beyond the system's capacity. We acknowledge that there are other overheads associated with the use of HPCs and APE for remote attestation. These include time overheads due to additional logging that might impact energy consumption as well. However, these apply only when remote attestation is executed and might be excluded when this is not active. The average overhead will thus depend on the frequency of attestation requests that reach the prover.

4 EXPERIMENTAL VALIDATION

The first goal of experimental validation is to determine, over a sufficiently broad set of applications, the general precision in detecting malware when using a subset of potential counters or all counters. We are then interested in determining whether some machine learning techniques can be used to improve the performance of simpler methods (data augmentation and Box-Cox transforms) and finally to analyze the relative importance of counters in determining a successful recall.

[5]In general, some form of dynamic instruction count, instruction cycles, and interrupt count is always available.
[6]For example, an application that sends packet over a bluetooth connection might measure the average signal quality and the number of high-level packets sent.

Concerning the low power requirements of the prover, we also estimate the prover overhead that CHARM introduces. Let us first describe the platform on which we will experiment.

4.1 Platform Description and Implementation

We ran our experiments on a Cortex-M33 with 512 KiB Flash and 256 KiB SRAM. The microcontroller provided a convenient Data Watchpoint and Trace (DWT) Unit with six HPC whose original purpose is debugging. The device also offers ARM's TrustZone protection, a feature we exploited to secure APE's values. We used the FreeRTOS microkernel to implement and manage some useful tasks. We implemented an *Attestation Routine*, a *Malicious Task* (see Section 4.1.3) to perform malicious relocations in Flash memory, and a set of *Application Tasks* (see Section 4.1.1) to stress the available counters. The Attestation Routine computes a SHA256 HMAC over the entire Flash and a random nonce. Other tasks run at a higher priority and occasionally interrupt the Attestation Routine. This leads to an interleaved execution that simulates what should happen in a real system at attestation time.

4.1.1 Synthesized Workloads. To emulate the behavior of realistic applications, we resorted to synthetic workloads. Although they are intrinsically different from real-life applications, they stress the available counters in a similar manner. These workloads were obtained by sampling the θ parameter space. We used a total of 16 synthetic workloads, each corresponding to a θ = (Entropy, Activity) combination taken from the set Θ. After fixing θ, an application task generates a behavior corresponding to the chosen Entropy and Activity. We implemented an application task per HPC for each of the DWT counters:

- DWT_LSUCNT: counts the extra clock cycles taken to complete a Load/Store instruction.
- DWT_EXCCNT: measures the exception overhead
- DWT_FLDCNT: counts the number of folded instructions. The latter are special ARM instructions that are executed in 0 clock cycles, being merged with the previous one.
- DWT_CPICNT: counts the extra clock cycles (besides the first) to execute some instruction.
- DWT_SLEEPCNT: counts clock cycles spent sleeping.
- DWT_CYCCNT: counts the number of cycles since its activation.

Both the application tasks and the malicious task interrupted several attestations, thus generating the corresponding counter vectors that we grouped into a dataset. The process was repeated once for each value of θ, for a total of 16 datasets.[7] Then,every classifier was trained and tested on all of them.

4.1.2 Application Event Counters. We defined one APE per application task, each counting the number of iterations of the corresponding task.[8] To secure their values, we placed them on TrustZone's Secure side. This prevents attackers from altering their values. Counters are made available to applications that increment them via **Non-Secure Callable (NSC)** functions [22] on the secure side. NSC memory provides a trusted entry point to Secure code from Non-Secure resources while preventing unauthorized access to the Secure side.

4.1.3 Tested Malwares. For realistic results, we designed the Malicious Task to emulate the relocations of varying malware samples. In fact, we cannot assume *a priori* the characteristics of the roving malware. We restrict these considerations to the two dimensions that reduce or increase the classification task's complexity: roving malware type (self-relocating vs. transient)

[7] Due to the aforementioned limits of Flash memories, the datasets were imbalanced with a benign:malicious ratio of approximately 9:1. Datasets can be found here https://github.com/DavideLiCalsi/CHARM

[8] As shown in FreeRTOS' documentation [1], we implemented each task as a non-terminating loop.

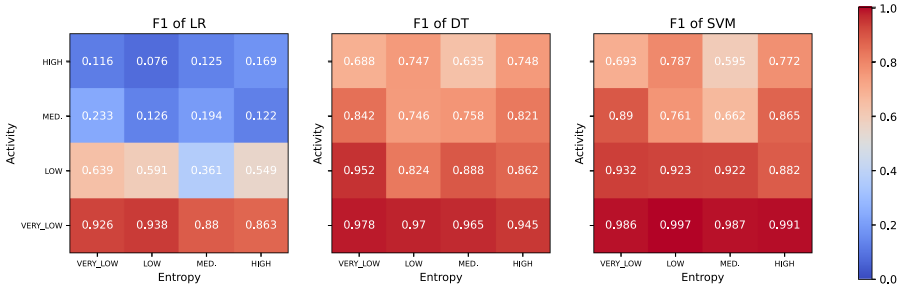

Fig. 4. F1 metric for each considered classifier, no data augmentation, HPC counters.

and malware size. With a fixed θ (that is, application behavior), the malicious task picks a random $(\text{Type}, \text{Size}) \in T \times S$ pair and relocates a sample with these characteristics. T is trivially the set $T = \{\text{self-relocating}, \text{transient}\}$. S is a set of reasonable malware sizes and depends on the target prover. In our experiments, we set it to the interval 2–16 KiB.[9]

4.2 Classifier's Performance Using HPC or HPC + APE Counters

Are application event counters (APE) necessary? To answer this question, we evaluated the classifiers using only counters HPC in the full parameter space of $\theta = (\text{Entropy}, \text{Activity})$. The F1 metric obtained shows that, for low levels of Activity, all classifiers provide discrete performance in both recall and precision (Figure 4). Higher levels of Activity tend to be significantly more difficult to address by logistic regression than the other classifiers. Increasing the Entropy level still makes classification more challenging, yet the increase is not as steep. SVM turns out to be the best classifier. It surpasses both competitors in almost every Entropy-Activity combination and provides F1 and recall scores above 0.7 in all but three combinations. The latter pose a complex challenge that HPC alone cannot overcome. This further motivates the introduction of APE as an optional enhancement for highly adversarial settings.

Adding application counters (APE) to the mix tends to improve SVM and logistic linear regression more than decision trees (Figure 5). The latter undergoes negligible variations in both the F1 and the recall scores. The other models show notable improvements. In high Activity and Entropy levels, logistic regression goes from being inapplicable to showing higher metrics. SVM strongly improves its detection ability. In general, increasing the Activity level is still the main source of difficulties, with Entropy playing a minor role.

4.3 APE-FREE Enhancements: Data Augmentation and/or Box-Cox Transforms

Instead of adding application counters APE to the mix, it is possible to improve the performance of classifiers by using data augmentation. We transform the counter-vectors by adding a small multivariate Gaussian noise $\epsilon \sim N(0, \sigma^2 I)$, where σ^2 was set to a small value ($\sqrt{5}$ in our setting). The idea is that small variations in the measured counters do not change the nature of the measured activity. For example, if one application performs only one more read than one that is known to be malicious, then that application is likely malicious as well. We tested this concept on LR and SVM, because they are the models that lost the most detection ability when deprived of APE. DT were excluded as we did not measure any loss related to APE that we had to compensate for. DA generated a noticeable gain in recall for LR (Figure 6, left), but this has a significant impact on

[9] Although malware for ARM IoT devices often has a size between 20 and 50 KiB, we considered smaller samples for greater security margins. It also mitigates *compress-and-relocate* attacks that could temporarily reduce the size of the malware.

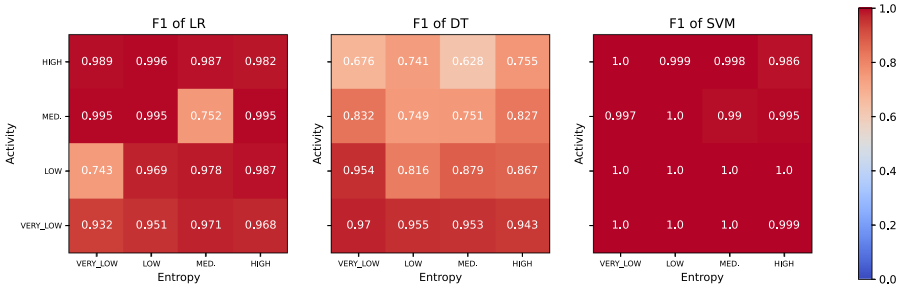

Fig. 5. F1 metric for each considered classifier, no data augmentation, HPC + APE counters.

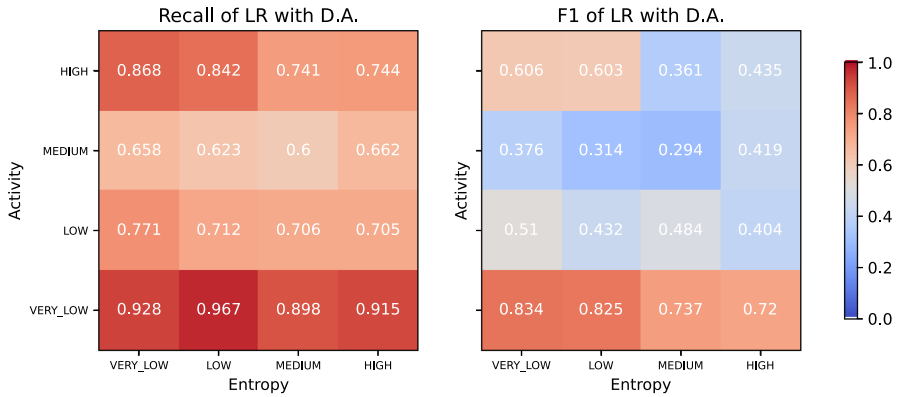

Fig. 6. F1 metric and recall when applying data augmentation to LR.

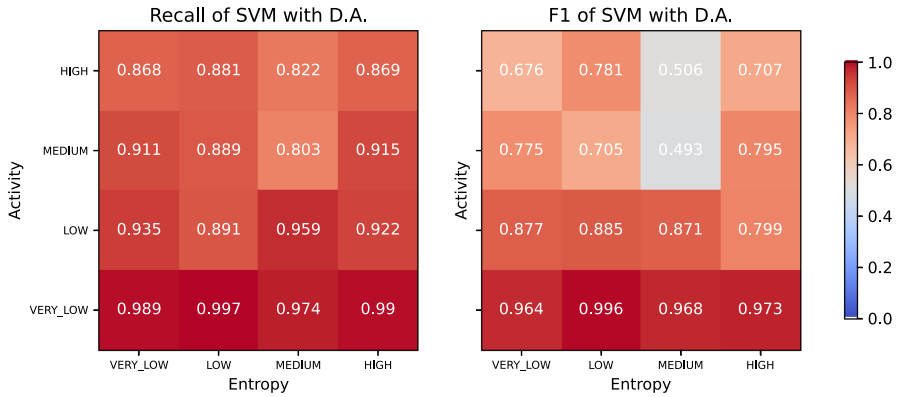

Fig. 7. F1 metric and recall when applying data augmentation to SVM.

precision, eventually improving only marginally LR's F1 score (Figure 6, right compared to Figure 4, left). As Figure 7 shows, SVM experiences a similar increase in recall with a nearly constant F1 score. This phenomenon is likely due to the fact that we only augmented malicious samples to compensate for the imbalance in our datasets.

The Box-Cox transform can be used to reduce the variance in the data and improve the performance of the model. We identified by cross-validation that a value of $0.3 \leq \lambda \leq 0.7$ is generally useful to greatly improve the performance of LR (Figure 8) and SVM (Figure 9).

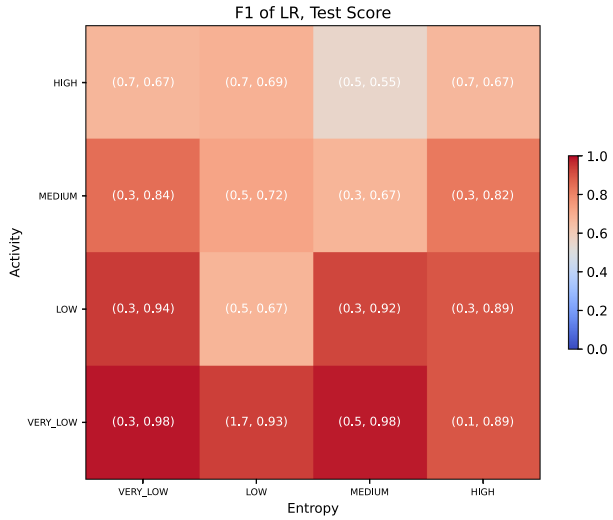

Fig. 8. F1 metric when applying the Box-Cox transform to LR. Each box is annotated with a pair (λ, m), where λ is the best value of the Box-Cox parameter while m is the actual value of the F1 metric.

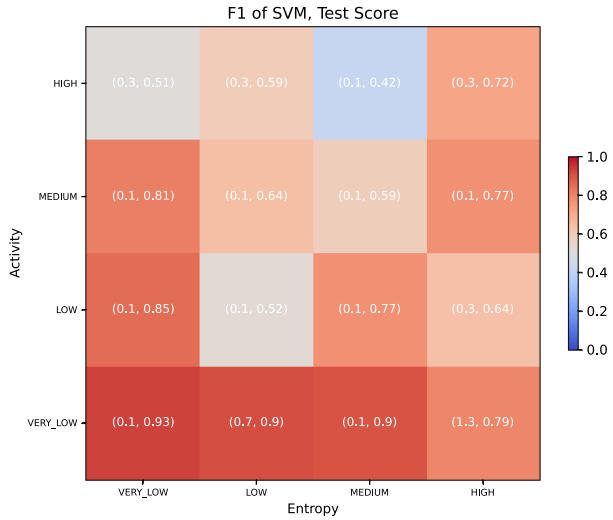

Fig. 9. F1 metric when applying the Box-Cox transform to SVM. Each box is annotated with a pair (λ, m) where λ is the best value of the Box-Cox parameter while m is the actual value of the F1 metric.

4.4 On the Importance of HPC Counters

Which HPC counters are really necessary? To answer this question, we use the forward selection technique to identify, for each $\theta \in \Theta$ the minimum subset of counters that could introduce an improvement in recall for the SVM classifier (Figure 10). We chose to focus only on SVM, because, given the detection ability shown, it is the most fit classifier for the real-life use of CHARM. This technique operates iteratively; at each iteration, one works with a model M_i that classifies counters $\{c_1, \ldots, c_i\}$ and obtains a model M_{i+1} classifying $\{c_1, \ldots, c_i, c_{i+1}\}$. The added counter c_{i+1} is taken from the remaining counters (i.e., those that are not in $\{c_1, \ldots, c_i\}$ yet) by selecting

	DWT_LSUCNT	DWT_CPICNT	DWT_EXCCNT	DWT_CYCCNT	DWT_SLEEPCNT	DWT_FOLDCNT	LSU_stim	CPI_stim	FLD_stim	EXC_stim	TIME
(VERY_LOW,VERY_LOW)			X			X				X	
(VERY_LOW,LOW)			X			X				X	
(VERY_LOW,MEDIUM)			X			X				X	
(VERY_LOW,HIGH)			X			X				X	
(LOW,VERY_LOW)		X	X	X			X	X	X	X	
(LOW,LOW)	X			X			X	X	X	X	
(LOW,MEDIUM)	X	X	X	X		X	X				
(LOW,HIGH)	X		X	X				X	X		
(MEDIUM,VERY_LOW)	X	X	X				X				
(MEDIUM,LOW)	X	X	X	X		X	X		X		
(MEDIUM,MEDIUM)	X										
(MEDIUM,HIGH)	X		X			X	X				
(HIGH,VERY_LOW)	X	X	X	X			X			X	
(HIGH,LOW)	X	X	X								
(HIGH,MEDIUM)	X	X	X			X	X	X	X		X
(HIGH,HIGH)	X	X	X								

Fig. 10. Prover-side overhead as a function of the: attested memory, using only HPC

the one whose addition maximizes the increase in recall. The procedure starts from an empty model M_0 and ends when the increase in recall is below $\tau = 0.05$. Reading the table columnwise highlights the counters that provide more information regardless of the selected θ. The most frequent counter is DWT_EXCCNT, followed by DWT_LSUCNT, which seems to indicate that the actual number of executed instructions and the number of loads and stores are more important than the cycle-based counters.

4.5 Prover-side Overhead

The prover-side overhead depends on the available hardware and the monitored events. We define it as the ratio

$$\frac{T_{CHARM} - T_{BASE}}{T_{BASE}},$$

where T_{BASE} is the time to attest memory without CHARM, and T_{CHARM} is the time taken by CHARM-enhanced attestation.[10] Using only HPC results in low overhead as they require only an initial reset and a final read, both of which are accomplished with a few instructions. Our platform's overhead, as shown in Figure 11, decreases with increasing attested memory size and is always less than 0.4%.

Hardware limitations can cause real-life overhead to increase. Some architectures have counters with few bits, which can lead to overflow that must be managed with software. However, throughout our experiments, we have never seen counter values that require more than 32 bits. Adding APE jeopardizes performance, as they require software management on the prover side. A precise estimation of this setting's overhead is a complex task on its own, as it is a function of the monitored events and the interaction with the external environment, but we leave it to future work. However, a rougher estimate can be found by checking the average rates at which the application-level events occur. We have found several examples of real applications by sampling quantities at frequencies between 0.3 and 50 Hz [17, 20, 26]. At these frequencies, the associated APE overhead is likely to still be modest. Hence we argue that there is a non-negligible class of application events that could be monitored with APE. High-frequency application events enhance the classifier's detection ability but jeopardize the prover-side overhead. This causes a *detection-overhead* tradeoff that should be carefully managed when selecting the set of events to record.

5 RELATED WORK

In this section, we strive to provide a taxonomy along several dimensions of the current state of literature around remote attestation. The first dimension we consider is the implementation support from the platform. Remote software attestation [24] is implemented completely in software and is

[10]For a fair comparison, this term does not include the time taken by applicative and malicious tasks to operate.

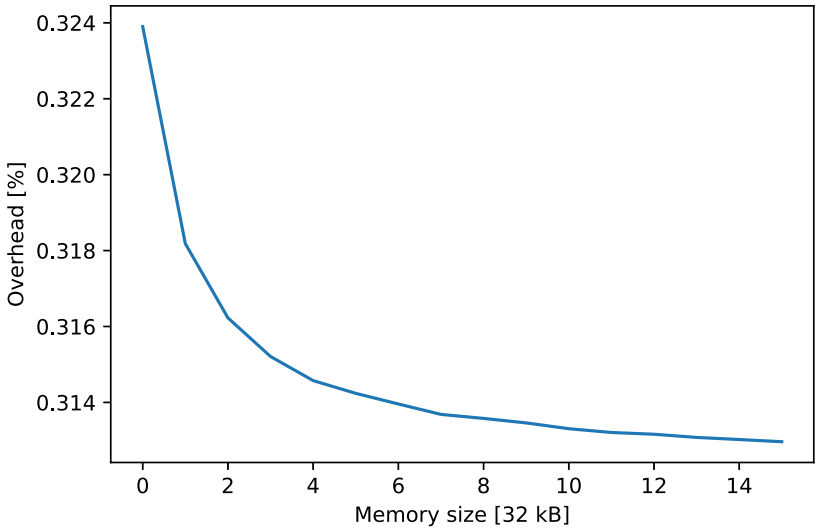

Fig. 11. Relevant HPC counters for each $\theta \in \Theta$ (rows).

the most flexible option but unfortunately is also the weakest one, because one must ensure that the prover's code is not tampered with. To avoid this, careful consideration of timings could be done, but some works [11] have shown that *return-oriented programming* and compression attacks can still compromise the system. To avoid attackers tampering with remote attestation code, hardware RA assumes that the trusted computing base is extended through a secure enclave mechanism that contains the prover's code (our approach follows this philosophy). Other approaches introduce smaller hardware requirements but rely on some mechanism of formal verification [13, 15]. Remote attestation can verify simple code memory (static RA, such as our approach), code and data memory (dynamic RA, [16]), and control flow integrity [14] and can work on one or more provers at the same time in a distributed way [3, 8].

Most RA methods require that interrupts be disabled during the attestation routine. This limitation is quite painful but necessary to guarantee the integrity of the attestation. In fact, consider the case of some malware that is located in memory. If malware is able to interrupt RA before it is detected, then it can successfully move to some previously examined memory area and cover its traces by writing benign code to its previous location. By doing so, the attestation's output is identical to what one would obtain on some "clean" memory even though malware is present. This type of malware is known as *roving malware*. Interrupts are a key feature that allows for a responsive system, and hence disabling them for too long is not a good idea. Experimental evidence shows that RA takes hundreds of milliseconds for a few MiBytes of memory. Our own tests using SHA256 HMAC resulted in about 300 ms to attest 512 KiB (although lighter hash functions [4] might reduce the computation time). Although some applications can tolerate that, it is unfeasible for MCUs employed in time-critical scenarios. In these settings, we would like to keep interrupts always enabled without sacrificing security. TyTan [7] addresses the issue by attesting one process at a time. All processes, except the one under attestation, are allowed to interrupt the RA. Furthermore, thanks to an EA-MPU, TyTan ensures that the memory of the process that is being attested cannot be altered.

SMARM [10] tackles roving malware by attesting memory in random order. Assuming that the memory is divided into n blocks, SMARM determines a pseudo-random permutation of the

memory blocks using the Fisher-Yates algorithm and attests them in that order. The effectiveness of this method is based on the unpredictability of the permutation. In fact, the best attacker's strategy consists in constantly moving to a random memory block as soon as the attestation routine completes a single block's attestation. Due to the stochastic nature of both the RA and the malware's actions, malware detection is no longer deterministic. The probability of detecting malware depends on the malware's knowledge of the attestation method and its size. Even in the most promising case, we only have a detection probability slightly higher than 63%. For malware fitting in s blocks, the detection probability increases to $1 - e^{-s}$, which is acceptable for s large enough. The authors of Reference [10] also suggested repeating RA multiple times to increase the probability of detection to the desired level. In addition to that, a single-block attestation still needs to be atomic, i.e., one can only interrupt between the attestation of a block and the next.

Memory locks [9] are also a valid alternative. In this context, locking a memory area means temporarily making it read-only. By properly locking and unlocking the memory, one can effectively hinder the roving malware and allow interrupts. Memory locking can be done in several ways, depending on what one locks and when. The most trivial locking strategy requires one to lock the entire memory, but it is convenient to resort to more clever techniques. For example, one can lock only the memory addresses that has already attested, which ensures that roving malware cannot contaminate them. The main drawback in this idea is that memory locks require an underlying microkernel to be implemented. Although this is not a major issue for the average device, very constrained MCUs lack the requirements to run it. Indeed, the source itself states that memory locking is not applicable in these contexts. Finally, it is worth mentioning that some solutions that circumvent the problem by minimizing the attestation time. Rather than allowing interrupts, they simply minimize the timespan in which they are disabled. HATT [2] implements this idea by combining the idea of shuffled measurements and subsampling of some bits to attest. The verifier sends two seeds: One determines a pseudo-random permutation on the memory blocks (i.e., the order in which one attests them), and the other is used to subsample a few bits in every memory word. Only these bits are actually attested, thus considerably reducing the attestation time.

Proof of Execution (PoX) is an extension of RA that generates unforgeable proofs of software tasks' proper execution and their interactions with analog peripherals using General Purpose Input/Output interfaces. Recent research has focused on enhancing RA architectures to generate PoX, as RA can only confirm the installation of appropriate software binary but cannot provide evidence of correct execution. Perhaps similarly to our approach is a technique for interruptible PoX named ASAP [12]. Differently from our approach, this technique relies on two assumptions: Immutable **interrupt vector table (IVT)** and Immutable ISRs. While the latter ensures through software that the attestation reflects the implemented behavior, hardware must ensure the immutability of IVT. **Control flow attestation (CFA)** [19] is another extension of RA that verifies the sequence in which instructions were executed in the attested binary, providing information to detect control-flow attacks during execution. Unlike RA, CFA does not actively protect against control-flow attacks but provides verification. TEE-based CFA methods use automated binary instrumentation to build an authenticated control flow log containing all control flow transfers for verification. However, interrupts can cause unreliable attestation reports but approaches such as ISC-FLAT enhance TEE-based CFA to generate reliable reports while enabling interrupts. This is done by securely initializing attestation and creating a dispatcher that configures protections before the ISR can execute. However, such techniques might generate large logs, do not detect non-control attacks, and require significant changes to the non-secure OS.

Research on hardware assisted malware detection has gained momentum recently. The article [21] presents an exploration of hardware-assisted malware detection using machine

learning and demonstrates the effectiveness of this approach with the use of hardware performance counters, embedded trace buffer, and on-chip network traffic analysis. In contrast to this positive outlook, [30] cautions against relying solely on hardware performance counters for malware detection due to a lack of causation between low-level micro-architectural events and high-level software behavior. Our approach agrees with this caution by also considering application level events in addition to hardware performance counters. Additionally, Reference [29] shows that decision trees and random forests perform well in detecting malware using HPCs, but these data are tested on general-purpose x86 architectures, whereas our approach is focused on microcontrollers. Overall, while there are limitations to relying solely on HPCs for malware detection as highlighted by Reference [30], our approach utilizing explainable machine learning and a combination of event sources including HPCs aims to address these limitations.

6 CONCLUSIONS

In summary, our work demonstrated that using hardware performance counters on the prover's side and machine learning on the verifier's side is a feasible approach for interruptible remote attestation, even for constrained microcontrollers. Based on the results of our evaluation, we found that support vector machines were the best classifier when using only hardware counters. While all classifiers showed discrete performance at low levels of Activity, higher levels of Activity were more difficult to address with logistic regression. Increasing the Entropy level also made classification more challenging, but not to the same extent as increasing Activity. Adding application counters to the mix tended to improve the performance of support vector machines and logistic linear regression more than decision trees, but data augmentation also showed potential for improving the performance of logistic regression. Using hardware performance counters on the prover's side to enable interruptible remote attestation results in low overhead, typically below 0.4%. However, adding application event counters can significantly increase the overhead, as they require software management on the prover side. There is thus a tradeoff between the detection ability of the classifiers and the prover-side overhead when using application event counters, and this should be carefully considered when selecting the events to monitor. In future work, we plan to evaluate neural networks against SVMs to validate our findings in terms of model performance, even if this means even less interpretability than the one available with support vector methods.

REFERENCES

[1] FreeRTOS' online documentation. Retrieved from https://www.freertos.org/a00125.html

[2] Muhammad Naveed Aman, Mohamed Haroon Basheer, Siddhant Dash, Jun Wen Wong, Jia Xu, Hoon Wei Lim, and Biplab Sikdar. 2020. HATT: Hybrid remote attestation for the internet of things with high availability. *IEEE IoT J.* 7, 8 (2020), 7220–7233. https://doi.org/10.1109/JIOT.2020.2983655

[3] Moreno Ambrosin, Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, and Silvio Ranise. 2018. PADS: Practical attestation for highly dynamic swarm topologies. arXiv:1806.05766. Retrieved from http://arxiv.org/abs/1806.05766

[4] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2: Simpler, smaller, fast as MD5. In *Applied Cryptography and Network Security*. Springer, Berlin, 119–135.

[5] Mohammad Bagher Bahador, Mahdi Abadi, and Asghar Tajoddin. 2014. HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In *Proceedings of the 4th International Conference on Computer and Knowledge Engineering (ICCKE'14)*. 703–708. https://doi.org/10.1109/ICCKE.2014.6993402

[6] Alexander Sprogø Banks, Marek Kisiel, and Philip Korsholm. 2021. Remote attestation: A literature review. arXiv:2105.02466. Retrieved from https://arxiv.org/abs/2105.02466

[7] Ferdinand Brasser, Brahim Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. 2015. TyTAN: Tiny trust anchor for tiny devices. In *Proceedings of the 52nd Annual Design Automation Conference*. 1–6. https://doi.org/10.1145/2744769.2744922

[8] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. Lightweight swarm attestation: A tale of two LISA-s. In *Proceedings of the ACM on Asia Conference on Computer and Communications Security (ASIA CCS'17)*. Association for Computing Machinery, New York, NY, 86–100. https://doi.org/10.1145/3052973.3053010

[9] Xavier Carpent, Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2018. Temporal consistency of integrity-ensuring computations and applications to embedded systems security. In *Proceedings of the Asia Conference on Computer and Communications Security (ASIACCS'18)*. Association for Computing Machinery, New York, NY, 313–327. https://doi.org/10.1145/3196494.3196526

[10] Xavier Carpent, Norrathep Rattanavipanon, and Gene Tsudik. 2018. Remote attestation of IoT devices via SMARM: Shuffled measurements against roving malware. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST'18)*. 9–16. https://doi.org/10.1109/HST.2018.8383885

[11] Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente. 2009. On the difficulty of software-based attestation of embedded devices. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*. Association for Computing Machinery, New York, NY, 400–409. https://doi.org/10.1145/1653662.1653711

[12] Adam Caulfield, Norrathep Rattanavipanon, and Ivan De Oliveira Nunes. 2022. ASAP: Reconciling asynchronous real-time operations and proofs of execution in simple embedded systems. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC'22)*. Association for Computing Machinery, New York, NY, 721–726. https://doi.org/10.1145/3489517.3530550

[13] Ivan De Oliveira Nunes, Karim Eldefrawy, Norrathep Rattanavipanon, Michael Steiner, and Gene Tsudik. 2019. VRASED: A verified hardware/software co-design for remote attestation. In *Proceedings of the 28th USENIX Security Symposium*.

[14] Ghada Dessouky, Tigist Abera, Ahmad Ibrahim, and Ahmad-Reza Sadeghi. 2018. LiteHAX: Lightweight hardware-assisted attestation of program execution. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'18)*. IEEE Press, 1–8. https://doi.org/10.1145/3240765.3240821

[15] Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. HYDRA: HYbrid design for remote attestation (using a formally verified microkernel). arXiv:1703.02688. Retrieved from http://arxiv.org/abs/1703.02688

[16] Boyu Kuang, Anmin Fu, Lu Zhou, Willy Susilo, and Yuqing Zhang. 2020. DO-RA: Data-oriented runtime attestation for IoT devices. *Comput. Secur.* 97 (2020), 101945. https://www.sciencedirect.com/journal/computers-and-security/vol/97/suppl/C

[17] Roberto López-Blanco, Miguel A. Velasco, Antonio Méndez-Guerrero, Juan Pablo Romero, María Dolores del Castillo, J. Ignacio Serrano, Eduardo Rocon, and Julián Benito-León. 2019. Smartwatch for the analysis of rest tremor in patients with Parkinson's disease. *J. Neurol. Sci.* 401 (2019), 37–42. https://doi.org/10.1016/j.jns.2019.04.011

[18] MalwareBazaar. [n. d.]. Malware samples for ARM 32-bit microcontrollers. https://bazaar.abuse.ch/sample/5f59b930fabf90bf48106e4cd778c21f88a51cbe12d60062bee54f51998b16ad/, https://bazaar.abuse.ch/sample/80415d67fa20f3b053a155da702e8b934e83dbaaf6119e06fa5f6cd1e66b0b20/, https://bazaar.abuse.ch/sample/dd1bb83aab0ab77cbccd54bd72bfc64baaa4aea86bae4b7adbd684743dbdbee5/

[19] Antonio Joia Neto and Ivan de Oliveira Nunes. 2023. ISC-FLAT: On the conflict between control flow attestation and real-time operations. arXiv:2303.03561 [cs.CR]. Retrieved from https://arxiv.org/abs/2303.03561

[20] Pacific Marine Enviromental Laboratory OCS: Ocean Climate Station. [n. d.]. Sampling Rates in a Weather Monitor. Retrieved from https://www.pmel.noaa.gov/ocs/sampling-rates

[21] Zhixin Pan, Jennifer Sheldon, Chamika Sudusinghe, Subodha Charles, and Prabhat Mishra. 2021. Hardware-assisted malware detection using machine learning. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'21)*. 1775–1780. https://doi.org/10.23919/DATE51398.2021.9474050

[22] Sandro Pinto and Nuno Santos. 2019. Demystifying ARM TrustZone: A comprehensive survey. *Comput. Surv.* 51 (1 2019), 1–36. https://doi.org/10.1145/3291047

[23] Martin Rosso, Joost Renes, Nikita Veshchikov, Eduardo Alvarenga, and Jerry den Hartog. 2021. Actionable malware classification in embedded environments using hardware performance counters. In *Proceedings of the 11th International Conference on Security, Privacy and Applied Cryptographic Engineering (SPACE'21)*. Conference date: 10-12-2021 Through 13-12-2021. https://cse.iitkgp.ac.in/conf/SPACE2021/

[24] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2004. SWATT: SoftWare-based ATTestation for embedded devices. *IEEE Secur. Priv. Mag.* (2004), 272–282. https://doi.org/10.1109/SECPRI.2004.1301329

[25] ST Microelectronics 2020. *Datasheet-STM32L552xx*. Retrieved from https://www.st.com/resource/en/datasheet/stm32l552cc.pdf

[26] Xueyi Wang, Joshua Ellul, and George Azzopardi. 2020. Elderly fall detection systems: A literature survey. *Front. Robot. AI* 7 (23 June 2020). https://doi.org/10.3389/frobt.2020.00071

[27] Xueyang Wang, Charalambos Konstantinou, Michail Maniatakos, and Ramesh Karri. 2015. ConFirm: Detecting firmware modifications in embedded systems using Hardware Performance Counters. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'15)*. 544–551. https://doi.org/10.1109/ICCAD.2015.7372617

[28] Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, and Ajay Joshi. 2018. Hardware performance counters can detect malware: Myth or fact? In *Proceedings of the ACM ASIA Conference on Computer and Communications Security (ASIACCS'18)*. Association for Computing Machinery, New York, NY, 457–468. https://doi.org/10.1145/3196494.3196515

[29] Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, and Ajay Joshi. 2018. Hardware performance counters can detect malware: Myth or fact?. In *Proceedings of the Asia Conference on Computer and Communications Security (ASIACCS'18)*. Association for Computing Machinery, New York, NY, 457–468. https://doi.org/10.1145/3196494.3196515

[30] Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, and Ajay Joshi. 2021. A cautionary tale about detecting malware using hardware performance counters and machine learning. *IEEE Des. Test* 38, 3 (2021), 39–50. https://doi.org/10.1109/MDAT.2021.3063338

Received 16 March 2023; revised 13 June 2023; accepted 20 July 2023