REMOTE: Robust External Malware Detection Framework by Using Electromagnetic Signals

Nader Sehatbakhsh, Alireza Nazari, Monjur Alam, Frank Werner, Yuanda Zhu, *Student Member, IEEE*, Alenka Zajic, *Senior Member, IEEE*, and Milos Prvulovic, *Senior Member, IEEE*

Abstract—Cyber-physical systems (CPS) are controlling many critical and sensitive aspects of our physical world while being continuously exposed to potential cyber-attacks. These systems typically have limited performance, memory, and energy reserves, which limits their ability to run existing advanced malware protection, and that, in turn, makes securing them very challenging. To tackle these problems, this paper proposes, REMOTE, a new robust framework to detect malware by externally observing Electromagnetic (EM) signals emitted by an electronic computing device (e.g., a microprocessor) while running a known application, in real-time and with a low detection latency, and without any a priori knowledge of the malware. REMOTE does not require any resources or infrastructure on, or any modifications to, the monitored system itself, which makes REMOTE especially suitable for malware detection on resource-constrained devices such as embedded devices, CPSs, and Internet of Things (IoT) devices where hardware and energy resources may be limited. To demonstrate the usability of REMOTE in real-world scenarios, we port *two* real-world programs (an embedded medical device and an industrial PID controller), each with a meaningful attack (a code-reuse and a code-injection attack), to *four* different hardware platforms. We also port shellcode-based DDoS and Ransomware attacks to *five* different standard applications on an embedded system. To further demonstrate the applicability of REMOTE to commercial CPS, we use REMOTE to monitor a *Robotic Arm*. Our results on all these different hardware platforms show that, for all attacks on each of the platforms, REMOTE successfully detects each instance of an attack and has < 0.1 percent false positives. We also systematically evaluate the robustness of REMOTE to interrupts and other system activity, to signal variation among different physical instances of the same device design, to changes over time, and to plastic enclosures and nearby electronic devices. This evaluation includes hundreds of measurements and shows that REMOTE achieves excellent accuracy (< 0.1 percent false positive and > 99.9 percent true positive rates) under all these conditions. We also compare REMOTE to prior work *EDDIE* [1] and *SYNDROME* [2], and demonstrate that these prior work are unable to achieve high accuracy under these variations.

Index Terms—Cyber-physical-systems, IoTs, intrusion detection, side-channels, embedded system security, electromagnetic

1 Introduction

Embedded systems and its derivatives–Cyber-Physical Systems (CPS), Internet of Things (IoT) devices, and Programmable Logic Controllers (PLC)–are proliferating in numbers and importance. By 2025, these "smart" devices are expected to be a USD 6.2 trillion market globally (this is 8 percent of the entire world's 2016 GDP), and most of that is expected to be in healthcare (USD 2.5 trillion) and manufacturing (USD 2.3 trillion) [3], [4]. While the "smart" world can provide many benefits for industries and individuals, it, unfortunately, comes with new opportunities for cyber-attacks. For example, by 2020, it is estimated that more than 25 percent of known attacks in enterprises will involve the CPS while less than 10 percent of IT security spending will be on CPS, indicating that there is an emerging need for more attention on CPS and embedded systems security [5].

There is a wide range of embedded systems and CPS security targets: cameras, cars, industrial PLCs, critical infrastructures such as the power grid (power distribution, nuclear and other power-plants, etc.), hospitals and embedded medical devices, etc. Many of these targets have already been attacked (e.g., DDoS attacks [6] in DNS services occurred by Mirai malware-infected CPS, *Persiarai* [7], etc.).

Because these devices use various and customized hardware and software, they may not be upgraded or updated as often as general-purpose systems, and software updates are even less frequent for devices where extensive verification or regulatory approval is needed. This makes embedded systems, CPSs, and IoTs challenging to keep up-to-date with the ever-evolving landscape of possible vulnerabilities and threats [8]. Furthermore, existing techniques for malware detection, such as those based on scanning for malware signatures [9], sandboxing [10], hardware-support [11], [12], [13], machine learning [14], and dynamic analysis [15], impose significant computational and cost overheads, so they are difficult to adapt to devices that often have severe performance, resource, power, and cost constraints. Moreover, the

- *N. Sehatbakhsh, A. Nazari, M. Alam, and M. Prvulovic are with the College of Computing, School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332. E-mail: {nader.sb, anazari, malam31}@gatech.edu, milos@cc.gatech.edu.*
- *F. Werner, Y. Zhou, and A. Zajic are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332. E-mail: {fwerner6, yzhu94}@gatech.edu, alenka.zajic@ece.gatech.edu.*

Manuscript received 4 Oct. 2018; revised 20 Sept. 2019; accepted 30 Sept. 2019. Date of publication 7 Oct. 2019; date of current version 7 Feb. 2020.
(Corresponding author: Nader Sehatbakhsh.)
Recommended for acceptance by T. Guneysu.
Digital Object Identifier no. 10.1109/TC.2019.2945767

low-complexity nature of *most* of these devices makes it easier to obtain full access to the device and then disable or even coopt its monitoring functionality.

To circumvent the difficult problem of implementing effective malware detection on the resource-constrained device (e.g., a CPS) itself, a recently proposed approach called EDDIE [1] (and its adoption in SYNDROME [2]) leverages electromagnetic (EM) emanations from the device to externally monitor it without requiring any support from, resources on, or changes to, the monitored device itself. While those have demonstrated the feasibility of efficient detecting malware on a CPS [1], [2], they did not consider a number issues that are key for practical adoption. We will show that without considering these issues, EDDIE and SYNDROME will fail to achieve high accuracy under different sources of variabilities and a new system should be designed to be robust against these issues.

In this paper, we propose a new framework, REMOTE (Robust External Malware Detection Framework by Using Electromagnetic Signals), that is designed to address practical issues for monitoring of resource-constrained devices (e.g., embedded devices, IoTs, CPS, etc.), which include:

- *Source Code and Infrastructure Availability*: application's source code, measurement, and/or instrumentation infrastructure might be *unavailable*,
- *Hardware and Software Variability*: the hardware might be based on different processor architectures and/or the device may use different operating systems or even not have one (bare-metal),
- *Physical Limitations*: enclosures may prevent direct access to the device, e.g., to place a power or EM probe very close to its processor and/or even system board,
- *Environmental Noise*: when using analog signals for monitoring, the environment may change the emanated signal and/or add interference to the received signal.

To develop REMOTE, we first understand how these issues may affect the EM signal, by carefully designing a signal measurement campaign on different systems, at different distances, under different environmental conditions. We then analyze these signals to understand in what ways they change and what they have in common. Using insights from this analysis, we design REMOTE to be robust to signal-related issues and to only rely on signal analysis (not the source code, instrumentation, etc.) for its training.

To emphasize practical applicability, our evaluation uses several real-world cyber-physical-systems: a medical embedded device called *Syringe-Pump*, a controller for the temperature of a soldering iron, an IoT device while executing a set applications from a standard embedded benchmark suite, and finally, a commercial industrial robotic arm. We implement these applications on a variety of well-known CPS/embedded devices including an Arduino UNO board, Altera's FPGA Nios II soft-core, and two Linux mini-computers: OlimexA13 and TS-7250. For monitoring these applications, we also implement several meaningful attacks. For Syringe-Pump, we implement a *Code Reuse Attack* by exploiting a buffer-overflow vulnerability that already existed in its open-source code. For the PID controller, we implement a *Stuxnet*-like *Advanced-Persistent Attack* where we assume the adversary has the ability to inject malicious instructions into the source code itself and modify the firmware of the system. Furthermore, we implement a *Shellcode Injection Attack* (i.e., gain shell access by overflowing a vulnerable buffer) on several applications from a standard embedded benchmark suite, MiBench [16], which represents usual activities of embedded/Internet-of-Things systems in the market. After hijacking the control-flow and invoking the shell, either a DDoS payload of the IoT botnet *Mirai* or encryption activity of a *Ransomware* payload is launched by the shell. We picked these two malwares since DDoS and Ransomware have become widespread threat and popular malware in recent years. We then show how REMOTE can find all the instances of these attacks with excellent accuracy. Lastly, for the Robotic-arm system, we use a commercial device (LewanSoul LeArm 6DOF [17]) and implement a *Firmware-modification/Zero-day* attack where we assumed that the libraries are compromised.

Specifically, the contributions of this paper are:

- *Black-box training model and a new distance metric*: unlike existing approaches, our method can be trained and monitors the execution without requiring any access to the source code. Moreover, based on the insights from [1], [18], we propose a new distance metric that is robust against several kinds of variability. (Section 3)
- *Robustness*: unlike state-of-the-art, our method is designed to be robust against a variety of hardware (e.g., ISA), software (e.g., OS), and environmental (e.g., temperature) variability. (Section 5)
- *Real attacks and security analysis:* we present a variety of real attacks including a code-reuse attack, an APT, a zero-day, and a shellcode injection attack. (Section 4)

We envision that REMOTE can be used in scenarios where the security of the device is critical, e.g., devices that control critical infrastructures, military systems, hospital equipment, etc. In these scenarios, the cost for deploying REMOTE is very low compared to the cost of the monitored system, and the complexity of deploying REMOTE is relatively low because it requires no changes to the monitored device and thus creates no regulatory, safety, or disruption concerns for the system since it is implemented on a *separate* system. Moreover, to further simplify the implementation and reduce the cost, especially in industrial scenarios, REMOTE can be implemented *as a part of* the existing (industrial) control infrastructure (e.g., Supervisory Control and Data Acquisition (SCADA) systems, Industrial Control Systems, etc.) that controls the embedded systems in the network. For example, REMOTE is very suitable for scenarios like an industrial robotic arm or a CPS in a power-plant where it can be implemented as a *dedicated* system to monitor the critical system. Alternatively, REMOTE can be used as a *mobile* setup (e.g., in a factory line) where it occasionally is utilized to monitor one or set of devices with similar tasks.

The rest of this paper presents some relevant background and the threat model (Section 2), describes the overall REMOTE framework (Section 3), presents malware prototypes and an evaluation of REMOTE (Section 4), demonstrates the importance of the robustness-oriented aspects of REMOTE

(Section 5), discusses related work (Section 6), and finally presents conclusions (Section 7).

2 Background

External Intrusion Detections (EMD). To externally monitor a system, a variety of side-channel signals could be used. Traditionally, side-channel signals were only used for extracting sensitive information through different techniques such as "template" attack [19] and/or differential power analysis (DPA) [20]. However, recently, especially due to advancements in machine learning and DSP algorithms, there has been a growing interest in using different side-channel signals for profiling and monitoring systems. The overall idea is that, because there is a correlation between the side-channel signals and the application that is being executed, these signals can be used to build a *reference* model for the normal behavior of the system, and then during monitoring, the receiving signal can be compared with the model, and a possible malicious activity will be detected if the signal and model are significantly different. It is also important to mention that similar approach (i.e., leveraging a reference/golden model to detect anomalies by using side-channel signals) can be used to detect another important class of attacks: hardware Trojans [21], [22], [23], [24], [25].

Among different types of physical side-channel signals, power and electromagnetic are more popular due to their availability and simplicity of measurements. While both can be used as a source for an external malware detector (and/or hardware Trojans), fundamentally, the advantage of using EM signals over power consumption is that EM signals usually have much more bandwidth than power traces mainly because, unlike power side-channel, EM emanated signals do not go through the existing, and often unwanted, (low-pass) filtering circuitry on the board thus the received EM side-channel contains both high-frequency and low-frequency information. Moreover, EM signals can be measured from some distance and can be received from multiple and/or localized sources so it provides unique information about a specific part of the system.

Agrawal et al. [26] observed that the amount of circuit activity inside a processor in each clock cycle depends on the program the processor is currently executing, so the EM signals produced by the clock are "unintentionally" *amplitude modulated* (AM) by program activity.[1] This results in EM emanations at the frequency of the processor clock, where clock signal acts as a carrier signal, and program activity acts as the modulator. This modulated clock-frequency signal is usually powerful, and at frequencies that avoid most human-made sources of noise (e.g., hundreds of MHz) so that it can be received at some distance with a reasonably high signal-to-noise ratio.

Recently, Callan et al. (ZOP [27]) used this observation to achieve fine-grained profiling of programs as they execute, avoiding the distortion of profiling results that is introduced by traditional (instrumentation-based or interrupt-intensive) approaches but with signal matching that is very computationally intensive.

More recently, Sehatbakhsh et al. (*Spectral Profiling* [18]) made an additional observation that much of the activities in an application, especially in less complex processors (e.g., CPS), are repetitive and happened in loops, and that for such activity the spectrum will have a peak at frequency $f_l = \frac{1}{T}$, where T is the average duration of one iteration of the loop. Since the program activity is not an ideal sinusoid, harmonics of this signal (at frequencies $2f_l, 3f_l, \ldots$) will also produce peaks. When this is combined with the modulated-clock observation, in the EM spectrum, a strong peak will be present at frequency f_{clk} (the clock), and the modulation of the clock by periodic program activity will produce additional peaks at frequencies $f_{clk} \pm f_l, f_{clk} \pm 2f_l, \ldots$. The authors used this observation to achieve efficient (usable in real-time) loop-level profiling of program execution without instrumentation-induced distortion of the results.

Even more recently, Nazari et al. (EDDIE [1]) leveraged the loop-created spectral peaks around the clock frequency to detect malware and other deviations from expected program execution. Specifically, they used lightweight instrumentation during training to identify which part of the signal corresponds to which loop in the program. The authors then determined peaks that can be used to identify each loop, and analyzed the source code to identify the valid loop-to-loop transitions that are possible in the program. During monitoring, a statistical test was used to check whether the behavior of each monitoring-time peak matches the peak's behavior from training on a candidate region, and an anomaly was reported if none of the regions that are allowed to execute at that point match the currently observed spectrum. Similarly, Han et al. (ZEUS [28]) showed that similar approach could be used in PLCs.

Our approach is also based on the observations from ZOP and Spectral Profiling, but is the first approach designed to be *robust* to variations in hardware, software, and environment, and also to work without source code and instrumentation even during training. Remote is also the first EMD approach to be systematically tested under real-world CPS and malware that targets them, on multiple kinds of devices, on many instances of the same device with cross-training (train on one instance then monitor others), over time, and under varying measurement conditions. Furthermore, unlike ZOP, Spectral Profiling, EDDIE, and ZEUS, that only studied *synthetic* attacks, in this work we will show that Remote is able to detect end-to-end attacks with high accuracy and low latency.

Compared to the existing *hardware* (internal) malware detectors [11], [13], [29], [30], [31], [32], the main advantage of EMD is that it does not require instrumentation and thus does not create any performance overhead (due to requesting interrupts) on the monitored system. Moreover, hardware malware detectors (HMD) are typically less resilient to evasion [33], [34] since they only leverage low-level features in the system which can be modified by an adversary who has full control on the device. EMDs, however, rely on physical characteristics (e.g., temperature, power, electromagnetic, etc.) of the device which are significantly more difficult to control and modify. The main advantage of HMDs over EMDs is that they do not require physical

1. Agrawal et al. [26] also discussed that the signal can be *angle-modulated*, however, they showed that detecting AM-modulated signals are quite simpler and more effective

proximity to the device for the signal measurements, and also, the measured signal is significantly less noisy since it has much less interference. Depending on measurement/instrumentation setup in EMDs and HMDs, both methods can achieve similar malware detection granularity (in terms of malware execution time), however, other limitations (e.g., overhead, cost, noise, etc.) can decide which method is more suitable for a specific system.

Threat Model. REMOTE is a novel runtime *anomaly* detection framework. For detecting the malicious activities, REMOTE has *no* a priori knowledge about the nature of the attack or its EM signature(s) and only relies on the signatures for the monitored application itself. We assume that REMOTE always has the correct reference models for malware-free signatures of the monitored applications and these models are stored in REMOTE and can not be compromised. Note that collecting the reference model needs to be done only once. We assume that this can be provided by the device manufacturer and/or the software developer. However, care should be taken if the model needs to be *updated* once the monitor is already deployed. The user needs to make sure that the device is not compromised (e.g., by establishing trust through other methods such as *attestation*) before using it to update the reference model. The main advantage of REMOTE over malware detectors that are implemented as part of the monitored system is that the REMOTE monitor is an entirely separate system, so it cannot be subverted by the same attack that succeeds in completely taking over the monitored system. Providing this *air-gap* eliminates the possibility of the monitor infection by the same attack vectors. As discussed in Section 1, we envision that REMOTE be implemented as a part of the central control system that controls the entire system and hence it would be protected by the several layers of security (e.g., firewalls, intrusion detection, etc.) typically in place in industrial control systems.

Further, our assumption in this paper is that the adversary has prior knowledge of the CPS and the program(s) being executing on the system (including existing vulnerabilities) and can manipulate the system by sending arbitrary inputs, or even has access to reprogram the system or modify the firmware.

3 REMOTE

3.1 Spectral Samples (SS)

At a high level, REMOTE has two phases: training and monitoring. In both phases, the EM signal is first transformed into a sequence of spectral samples (SS) by using short-time Fourier transform (STFT), which divides the signal into equal-sized segments (*windows*), where consecutive segments overlap to some degree. STFT then applies the Fast Fourier Transform (FFT) to each window to obtain its spectrum. In our measurements, we use a $1\,\text{ms}$ window size[2] with 75 percent overlap between consecutive windows, which provides a balance between the computation complexity and frequency/time resolution. The rest of the training and monitoring operates on this sequence of spectra, where each spectrum (i.e., the spectrum of one window) is referred to as a Spectral Sample.

3.2 Distance Metric for Comparing SSs

In both training and monitoring, REMOTE will need to compare SSs to each other, and for that, it requires a distance metric–a way to measure the "distance" between SSs in a way that corresponds how likely/unlikely they are to have been produced by execution of the same code. This distance metric should be sensitive to the aspects of the signal that change when executing different code, but insensitive to aspects of the signal that change between physical instances of the same device or over time on the same device instance. To achieve this, we create a new distance metric, *Clock-Adjusted Energy and Peaks (CAPE)*.

Based on the insights from prior work [1], [18], [28], the frequencies of the peaks in the signal around the clock frequency are an excellent foundation for constructing a distance metric that is sensitive to which region of code is executing. Unfortunately, our experiments have shown that the clock frequency can vary over time and among device instances, and a change in clock frequency also changes the frequencies of loop-related peaks around it. One difference is that, because the peaks' frequencies are all relative to the carrier frequency, any shift in the clock frequency also shifts the frequencies of the loop-related peaks by the same amount. The second change is caused by the relationship between clock frequency and program performance. Specifically, as the clock frequency increases, the program executes faster, leading to a lower per-iteration time T, higher frequency of the loop ($f_l = 1/T$ in Section 2), and thus moving the loop-related peaks away from the clock's frequency. Similarly, lower clock frequency moves the loop-related peaks closer to the clock frequency.

Thus the first step in computing our CAPE distance metric is to, for each frequency f that is of interest in an SS, compute the corresponding normalized frequency as $f_{norm} = \frac{f - f_{clk}}{f_{clk}}$, where F_{clk} is the clock frequency for that SS. This normalized frequency is expressed as an offset from the clock frequency so that a shift in clock frequency does not change f_{norm} with it and is normalized to the clock, so it accounts for the clock frequency's first-order effect on execution time.

To make CAPE robust to weak signals and/or signals that have no well-defined peaks, we first consider the overall signal power (sum of magnitudes in the spectrum) of the signal outside the vicinity of the clock. The power of a poorly-defined peak is spread across a range of frequencies–visually it is a wide and not-very-tall "hump" rather than a narrow and tall "peak". When comparing two SSs that are different but each contain only "humps" and no sharp peaks, if we only consider whether the SSs have power concentrations at the same (clock-adjusted) frequencies, the overlap among their "humps" causes these SSs to match much better than they otherwise should, and this can prevent detection of malware-induced changes in signals. Moreover, under poor signal-to-noise conditions (e.g., when the signal is received at a distance) sharp peaks are likely to still stand out of the noise, so due to random variation in noise, some "humps" end up below the noise level and some do not. For two SSs that should be the same (except for the noise), this

2. the window size should be determined based on sampling rate, clock frequency, and the required time resolution.

causes poor matches, and this can lead to false positives. Thus to make our CAPE distance metric more robust against weak/noisy signals, we use a new insight, called *"non-clock-energy"* test, that non-clock power varies very little among SSs that do belong to the same region, and that increases/decreases in a loop's overall per-iteration time concentrate less/more power toward the clock frequency in an SS. Therefore, SSs whose non-clock power differs by more than 0.5 dB are considered dissimilar by CAPE, and no further comparison between them is needed.

If the two SSs pass the "non-clock-energy" test, REMOTE compares them according to the (clock-adjusted) frequencies of their most prominent peaks. Specifically, we take N highest-magnitude frequency bins from the spectral sample that are each (i) not part of the *NoiseList*, and (ii) not within D spectral bins of a higher-amplitude spectral bin. The number N is determined differently for training and monitoring, as will be described shortly. The *NoiseList* contains frequencies of signals that are present regardless of which specific region of the application is executing. For finding the *NoiseList*, we record the EM signal several times and average them while no program is being executed (the system is idle). We then choose 10 random SSs in the recorded signal, and then for each SS, sort it and find all the spikes that are at least 5 dB above the noise floor and put them in the *NoiseList*. We empirically find that choosing 10 points is sufficient to find all the strong peaks since it can accurately capture the transient behavior of the environmental noise. It is also important to point that, using this method, our detection algorithm is robust to interference from nearby devices (that are not identical to the monitored device), as their clock and other frequently-occurring peaks will end up on the *NoiseList*. The reason for ignoring D spectral bins that are too close to even-higher-magnitude ones is that a very prominent peak in the spectrum typically has "slopes" whose magnitude can exceed the magnitude of other peaks, and we found that REMOTE is more robust when its decisions are based on separate peaks rather than just a few (possibly even one) very strong peaks and a number of frequency bins that belong to their "slopes".

Finally, REMOTE combines the information about the frequencies of the peaks in the two SSs into a single value that represents the distance among the SSs. For each peak in one SS, REMOTE finds the closest-frequency peak the other. If the frequency difference is large enough, the peak votes for a mismatch, and the ratio of the mismatch votes to the number of all (mismatch and match) votes is used as the distance metric between the two SSs.

3.3 Black-Box Training

To train REMOTE, signals are collected as the unmodified monitored device emanates them. However, care should be taken to achieve good coverage of the software behaviors, e.g., by using the same methods that are used to test program correctness. The problem of achieving good coverage tends to be easier for many applications in the CPS domain, especially those where correct operation is critical, because correctness concerns and the need for easy verification of correct operation motivates developers to produce code that has relatively few code regions, and with very stable patterns for how the execution transitions between them. In such cases, normal use of the device is likely to provide good coverage of the application's code regions after a while.

After signals are obtained and converted into SSs, a key part of training is to associate SSs with the code regions they correspond to. To achieve this without using instrumentation or other on-the-monitored-device infrastructure, REMOTE relies on a general observation that a given region of code tends to produce EM signals whose SSs are similar to each other, while the SSs from different regions tend to differ from each other to various degrees. This observation allows us to group SSs according to similarity, and for that we use Hierarchical Density-Based Spatial Clustering of Applications with Noise *(HDBSCAN)*, a technique that performs clustering without any a priory knowledge about which cluster (region) each sample (SS) corresponds to, and with no a priori knowledge about the number of clusters (regions). Like other clustering algorithms, HDBCASN needs a distance metric, and in REMOTE that distance metric is the new CAPE metric defined in Section 3.2, using $N = 10$ peaks. Using this variation-robust metric allows training signals to be collected over time (e.g., over many hours), and/or on multiple device instances.

Because HDBSCAN clustering is based solely on similarity among SSs, its result may not precisely correspond to actual regions of the code, e.g., one region may produce more than one cluster if there are several distinct ways in which the region can execute, or two regions may end up in the same cluster if their execution produces very similar signals. Neither of these possibilities is a problem for REMOTE, and in fact, they result in improved sensitivity and performance. If separate clusters for distinct behaviors were forced into a single cluster, the resulting unified cluster would allow a wide variety of SSs to match - all the valid SS options and *also* everything that lays in-between in the distance-space used for clustering. By creating a separate cluster for each distinct possibility, REMOTE will detect anomalies that produce SSs that are not valid but lay in-between the valid ones. Conversely, when multiple regions are clustered together, they have very similar (practically indistinguishable) signals and it is more efficient and robust to treat them as one cluster. During monitoring, a Finite-State Machine (FSM) is used to keep track of the current region of the code. For each test, REMOTE compares the new SS to either the current region or any valid "next region" that has been seen during training.

3.4 Monitoring

During monitoring, REMOTE receives the signal and converts it to SSs in the same way it was done in training (same window size and overlap). After that, REMOTE can be viewed as a classifier that places each spectral sample into either one of the known categories (clusters identified during training) or into the "unknown" category that represents anomalous behavior, according to our CAPE distance metric (Fig. 1 shows the flow-chart of REMOTE's monitoring algorithm). Specifically, a candidate region is rejected if its distance metric is above 50 percent (fewer than half of the peaks match). If all candidates are rejected, the observed SS is categorized as "unknown," otherwise it is categorized into the candidate category with the lowest CAPE distance metric. The number of peaks used for each cluster is no longer fixed at 10–instead, it is identified for each cluster during training. We start with ten peaks, but then remove those that occur in

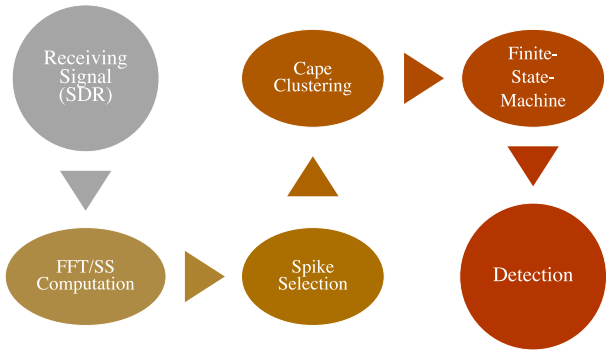


Fig. 1. REMOTE's monitoring flow-chart.

fewer than 10 percent of the SS in the cluster. If this results in removing all peaks, we still retain the two most frequently occurring (among SSs from that cluster) peaks. This helps matching accuracy when the SSs in a cluster have few prominent peaks and a number of very weak peaks–in such cases it is more robust to use only the overall non-clock energy and the prominent peaks for matching than to use the peaks that may "disappear into the noise" due to changes in distance, antenna position, etc.

However, if the overall decision to report malware is based on only one SS, brief occurrences of strong transient noise can result in false positives. To avoid that, REMOTE only reports an anomaly if N consecutive SSs are classified as "unknown." The value of N should be chosen depending on the EM noise characteristics of the environment, but we found that N between 3 and 5 tends to work well in all our experiments. We use N=5 because it biases REMOTE toward avoiding false positives, while still maintaining an excellent detection latency (N=5 corresponds to only 1.25 ms detection latency in our setup). As mentioned in Section 3.3, an FSM is used to count N, report an anomaly, and to keep track of current valid region of code to ensure that the program follow a correct ordering of regions.

Finally, we found that in the presence of an OS, interrupts and other system activity that occurs during an SS can make that SS dissimilar to those from training. For example, an interrupt that lasts <1 ms can affect four consecutive SSs (recall that we use 1 ms windows with 75 percent overlap), so a naive solution would be to add 4 to N (number of consecutive "unknown" SSs that are needed to trigger anomaly reporting). Using $N = 9$ indeed eliminates interrupt-induced false positives, but also prevents detection of attacks that are brief. Unfortunately, real-world malware (e.g., the attack on *Syringe-pump* that will be described in Section 4) can introduce only a short burst of activity into the otherwise-normal activity of the application. Fortunately, our experiments indicate that spectral features of interrupt activity are similar to each other, so during training interrupt activity can be clustered. During monitoring, REMOTE includes these clusters as candidates, allowing it to tolerate interrupts without becoming tolerant of similar-duration deviations from expected execution.

4 EXPERIMENTAL RESULTS

In this section, we evaluate our framework using three set of experiments to show the effectiveness of REMOTE to detect different types of attack on variety of devices.

In the first set of experiments, we use two real-world CPS. The first CPS we use is an embedded medical device called *Syringe-Pump* which is a representative of a medical cyber-physical system. The second system is a PID controller that is used for controlling the temperature of a soldering Iron. This type of system could also be used to control the temperature in other settings, such as a building or an industrial process, and thus is representative of a large class of industrial CPS/IoT systems.

For the second set of experiments, we use five applications from an embedded benchmark suite called MiBench [16] running on an IoT/embedded device, which are a representative of the computation that is needed in that market (e.g., automotive, industrial systems, etc.)

Finally, for the third part of our evaluations, we chose a robotic arm (LewanSoul LeArm 6DOF) [17], which is a representative of commonly-used CPS existing in the market.

4.1 Measurement Setup

The measurement setup is shown in Fig. 3. Depending on the distance, either a hand-made magnetic coil or a horn antenna is used to receive EM signals (no amplifier is used). For all measurements, we use a cheap ($<$\$30) software-defined radio (SDR) receiver (RTL-SDRv3) to record the signal. Using this radio, the entire cost for the near-field measurement setup (including the radio and a hand-made coil) is only around \$35, and for the far-field measurement setup is around \$100-200 (depending on the antenna). Further cost advantages can be gained if REMOTE is used in settings where multiple similar devices (with similar vulnerabilities) are used, so a single (or a few) devices can be monitored by REMOTE (especially in far-field scenario), with random changes to which specific devices are monitored at any given time. Fig. 2 shows the entire setup including the monitored device (Syringe-Pump in this figure) and REMOTE. Note that all of our measurements were collected in the presence of other sources of the electromagnetic interface (EMI) including an active LCD that was intentionally placed about 15 cm behind the board. A set of TCL scripts are used to control the monitored system and the SDR (to record the signal). The entire REMOTE algorithm is implemented on a PC using Matlab2017-b.

4.2 File-Less Attacks on Cyber-Physical-Systems

The first part of our evaluations presents the results for two real-world CPS which are implemented on four different devices (shown in Table 1). To attack these devices we implement two end-to-end *file-less* attacks namely a *code-reuse* attack and an *APT* attack (advanced-persistent-threat).

The *first attack* we implement in this paper is a Code Reuse [35], [36] attack on a medical CPS called Syringe-Pump. Syringe-Pump is a medical device designed to dispense or withdraw a precise amount of fluid, e.g., in hospitals for applying medication at frequent interval [37]. The device typically consists of a syringe filled with medicine, an actuator (e.g., stepper motor), and a control unit that takes commands (e.g., amount of fluid to dispense/withdraw) and produces controls for the stepper motor. The systems must provide a high degree of reliability and assurance (typically by using a simple MAC) since imprecise or unwanted dispensing of medication, or failure to administer medication

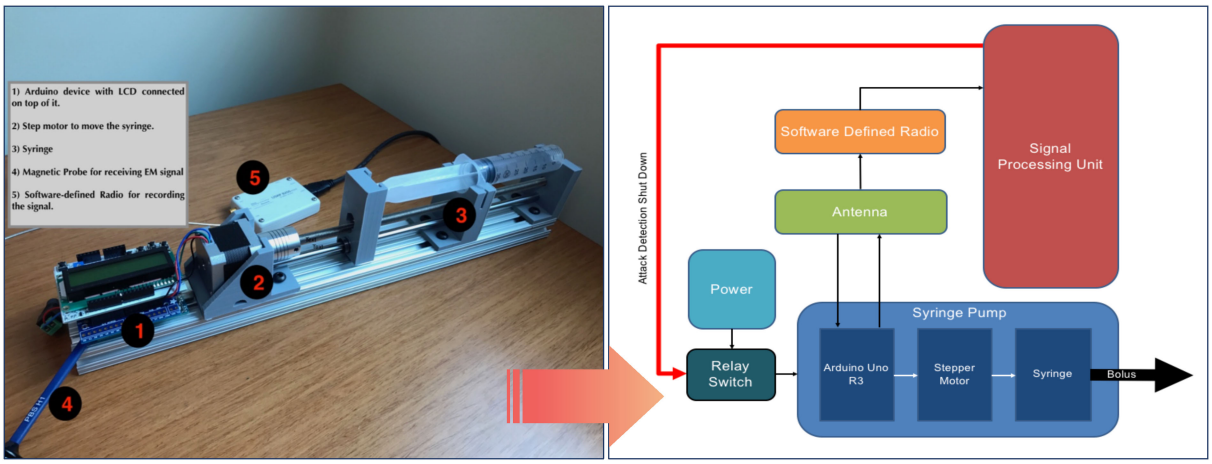


Fig. 2. Syringe Pump (left) and REMOTE framework (right). In our setup, the signal processing unit is implemented on a *separate* PC.

when needed can cause significant damage to the patient's health. In our evaluation, we use the Open Source Syringe-Pump from [38] also implemented in [2].

Our code-reuse attack involves overflowing the input buffer in reading the serial input function, which normally reads the input, sets a flag to indicate that new input is available, and returns. Exploiting this vulnerability, the return address in the stack is overwritten by a chain of gadget's addresses to launch an attack.

Since the security-critical part of this system is moving the syringe, a desirable goal for an attacker is being able to call the *MoveSyringe()* function, which is responsible for syringe movement, at an unwanted time while *skipping* the input checking part, *Delay()* function, which is responsible to check the authenticity of the command (otherwise the attacker needs to hack into the C&C server to send the commands which may not be a feasible task).

We use ROPGadget [39] for finding the proper chain of gadgets to put the address of *MoveSyringe()* in a register and branching to that function (from the *readInput()* function to skip the checking part). After branching to *MoveSyringe()* and executing it, PC jumps back to the main function and resumes normal behavior of the application.

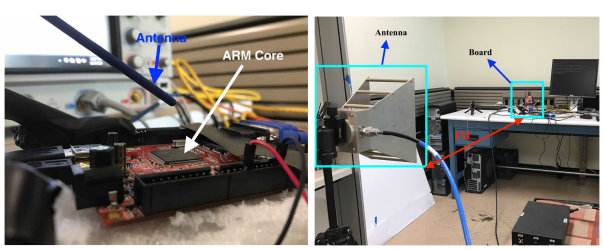


Fig. 3. The near-field setup (left) consists of a small EM probe or a hand-made magnetic probe (not shown) placed 5 cm above the system's processor. A horn antenna placed 1 m away from the board for far-field measurements (right). In all cases, a software-defined radio smaller and lighter than most portable USB hard drives, is used to record the signal.

TABLE 1
Boards used in this Paper to Evaluate REMOTE

Device	Processor	Clock-rate	OS
Arduino Uno	ATMEGA-328p	16 MHz	No
DE0-CV Altera FPGA	Nios-II softcore	50 MHz	No
TS-7250	ARM9	200 MHz	Debian
A13-OLinuXino	ARM Cortex A8	1 GHz	Debian

Fig. 4 shows a spectrogram of the Syringe-Pump application in (top) malware-free run, and (bottom) when the CR attack happens. As seen in the figure, the Syringe-Pump application has three distinct regions with clearly different EM signatures: printing debug info and reading inputs, a delay/checker function which checks the message authenticity (using a simple MAC), and an actual movement of the syringe. The major difference between these two figures is the *reverse* order of "Delay" and *MoveSyringe()* parts in malicious run (bottom). In normal behavior, REMOTE expects to see $readInput \rightarrow Delay \rightarrow MoveSyirnge$ however, in CR attack, since the return address of the $readInput$ function is overwritten by the adversary, the code immediately jumps to *MoveSyringe()* and skips the "Delay" part, thus in the spectrogram, the third region (*MoveSyringe()*) is seen before "Delay" (bottom), which violates the correct ordering of regions and will be reported as "malicious" by REMOTE.

Our evaluation uses one attack per run in 25 runs with REMOTE successfully detecting each of these attacks (see Table 2). We then performed 25 attack-free runs and found that REMOTE produced no false positives (see Table 2). To further evaluate our system, we performed 1,000 malware-free runs and 1,000 malicious run on one device (Arduino)[3] for 24 hours. For these 2,000 runs, REMOTE successfully found all the 1,000 instances of malicious run and reported 997 out of 1,000 malware-free runs as normal (i.e., only 3 out of 1,000 false positive $= 0.3\%$).

Note that, depending on the size of injection, the *MoveSyringe()* in Syringe-Pump could be very brief in time (e.g., around 3 ms as can be seen in Fig. 4-left), and we found that without correctly handling the interrupts on Olimex and TS platforms (which have an operating system), we would either get very high false positives (due to interrupts), or high false negatives (by using large N to ignore short-term activity). However, as also discussed in Section 3.4, by adding training-time samples for interrupts, we can use small N, while having 0 percent false positives (see Section 5.1).

3. To limit the amount of measurements and time for processing it, we picked only one of the four devices.

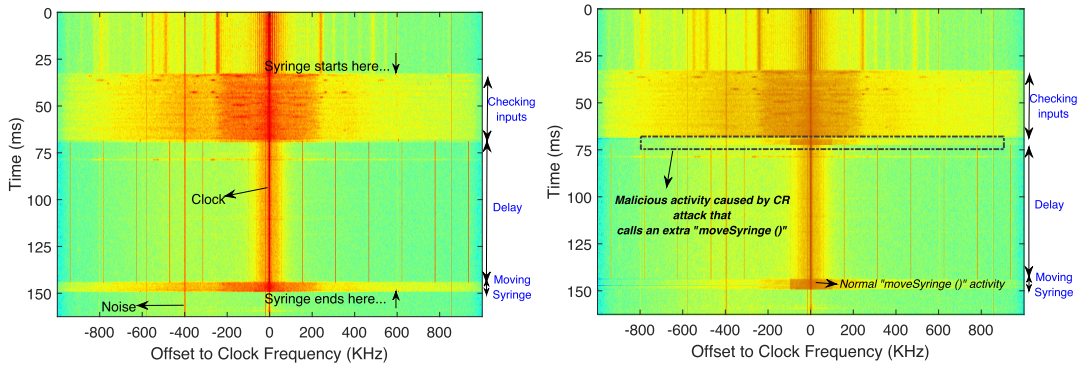


Fig. 4. Spectrogram of the Syringe pump application in malware-free (left) and malware-afflicted (right) runs. Note that the differences in colors between the two spectrograms correspond to differences in signal magnitude which are caused by different positioning of the antenna. Such variation is common in practice and has almost no effect on REMOTE's functionality because REMOTE was designed to be robust to such variation.

Furthermore, we also repeated our measurement for Syringe-Pump for both 50 cm and 1 m distances (using a 9 dBi horn antenna [40] connected to the SDR) and in both cases, we also get perfect accuracy. It is also important to mention that the detection latency (i.e., the time attack starts until REMOTE detects it), for all four devices is <2 ms.

An alternative method for attacking Syringe-Pump is by changing the *InjectionSize* (i.e., *Data-only attacks*). This also can be done using a CR attack. REMOTE is able to protect Syringe-pump against such attack since changing the *InjectionSize* will change the duration (i.e., the number of SSs) of *MoveSyringe()*. Since REMOTE is checking the signal in the granularity of SS, it can count the SSs which belong to *MoveSyringe()* activity and compare it to the expected number of SSs. To check how well REMOTE can detect such an attack, we check the number of SSs for *MoveSyringe()* for all the 25 attack-free runs and compare it to the actual *InjectionSize*. In all the instances, REMOTE reports the correct number of SSs. Note that we are not detecting EM emanations (RF) signal produced by the motor movement but the change in the code execution when "data-only" attack is performed. i.e., we observe the signal at clock frequency of the board and observe software changes, while motor movement signature occurs at much lower frequencies.

However, if the change is less than one SS or if the expected *InjectionSize* is unknown, REMOTE is not able to detect the change. Overall, there is a tradeoff between the size of SS and REMOTE's ability to detect small changes. Thus to improve the effectiveness of the system, either a higher sampling-rate setup can be used (smaller SS hence smaller detection granularity) or REMOTE can be combined with other existing methods (e.g., Data Confidentiality and Integrity (DCI) methods [41]) to protect the system against different types of data-only attacks. Finally, it is important to mention that However, as shown in this work (for this attack and other attacks in this section), meaningful attacks typically have much larger signature (i.e., order of milliseconds) than the current detection limit in REMOTE (200 microseconds).

The *second attack* is an advanced-persistent-threat (APT) attack on an industrial CPS (called Soldering-iron). A well-known example of such attack for CPS is Stuxnet. *Soldering-iron* is an industrial CPS that allows users to specify a desired temperature for the iron and maintains it at that temperature using a proportional-integral-derivative (PID) controller. This type of controller could also be used to control the temperature in other settings, such as a building or an industrial process, and thus is representative of a large class of industrial CPS. This application is significantly larger than the Syringe-Pump - with 70,000 instructions in its code and 1,020 static control-flow edges [38].

The application starts by initializing all the components (e.g., PID controller, Iron, etc.). It then begins to control the Iron's temperature: it checks all the inputs (e.g., knob, push buttons, etc.) and then based on them decides to decrease or increase the temperature, prints new debug information on its display, etc. and then repeats this ad infinitum. The security-critical function is where the temperature of the iron is set *keepTemp()*. This function uses an iterative process (a PID controller) to change or keep the temperature of the iron. The critical variable is *temp_hist*–it holds the last two temperatures of the iron and is used to calculate the difference between the current temperature of the iron and these two last temperatures.

To implement a Stuxnet-like malware on this application, we assume that the attacker can reprogram the device. The

TABLE 2
Accuracy of REMOTE for Several Different Systems and Attack Scenarios using Various Boards and Applications

Device (attack)	Syringe-pump (code-reuse attack)							
Board	*Arduino*		*Nios-II*		*TS-board*		*OLinuXino*	
Accuracy	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.
	>99.9%	<0.3%	>99.9%	<0.1%	>99.9%	<0.1%	>99.9%	<0.1%

Device (attack)	Soldering-iron (APT attack)							
Board	*Arduino*		*Nios-II*		*TS-board*		*OLinuXino*	
Accuracy	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.
	>99.9%	<0.1%	>99.9%	<0.1%	>99.9%	<0.1%	>99.9%	<0.1%

Device (attack)	Embedded/IoT (shellcode attack)									
App	*bitcount*		*basicmath*		*qsort*		*susan*		*fft*	
Accuracy	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.	True Pos.	False Pos.
	>99.9%	<0.1%	>99.9%	<0.1%	>99.9%	<0.1%	100%	<0.1%	>99.9%	<0.1%

Device (attack)	Robotic-arm (firmware modification)	
Board	*LewanSoul LeArm 6DOF*	
Accuracy	True Pos.	False Pos.
	>98.2%	<0.2%

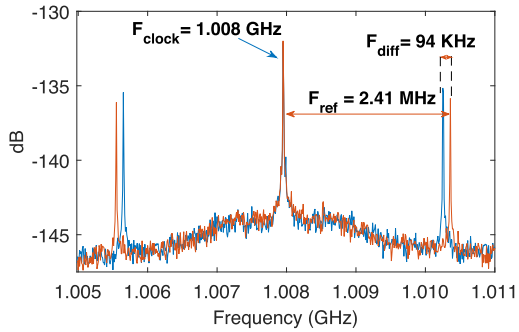


Fig. 5. Adding malicious activity to the main loop of the Soldering-iron application (red: without malware, blue: with malware).

attacker's goal is to change a critical value under some conditions, which in turn can cause damage to the overall system. A possible modification to the code is shown in Example 1 (lines 8-10), where based on one or several conditions (e.g., in our evaluation it checks the model of the device that is stored in memory), the temperature history can be changed. The key insight is that the added instructions will cause the spectral spikes during execution of the main loop to be shifted to lower frequencies (more time per iteration) as shown in Fig. 5 for the A13-OLinuXino device.

Example 1. A Code Fragment from the Main Loop of the Soldering Iron Application and a Possible Injected Malicious Activity

```
1  // The main loop
2  void loop() {
3      int16_t old_pos = read(&rotEncoder);
       // finding the position of the control knob
4
5      bool iron_on = isOn(&iron);
       // iron is the object for the soldering iron
6
7      // adding malicious activity
8    if (some_condition){
9      iron.temp_hist[0] = maliciousVal0;
10     iron.temp_hist[1] = maliciousVal1;
11     // where these values can be read from a file,
       memory or they could be random
12   } // end of malicious activity
13
14   byte bStatus = intButtonStatus(&rotButton);
     // reading input button
15
16   showScreen(pCurrentScreen);
17   keepTemp(&iron); }
```

To evaluate how well REMOTE can detect this type of attack, we use 7 runs in training, and use 25 runs without malware and 25 runs with malware to evaluate the monitoring algorithm. Our results show REMOTE can successfully detect all the instances of the attack (a 100 percent true positive rate) (see Table 2).

4.3 Shellcode Attack on IoTs

Another popular class of attacks on CPS/IoTs are *shellcode* attacks where the adversary executes a malicious application (payload) through exploiting a software vulnerability. It is called "shellcode" because it typically starts a command shell (e.g., by executing *(/bin/sh)* binary) from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called shellcode. Once the attacker takes the control, she can execute any *injected* code such as a *Denial-of-Service* attack.

In this paper, we implement this attack by invoking a shell *(/bin/sh)* via a buffer overflow exploit. We then run two malicious payloads on the invoked shell: a *DDoS bot*, and a *Ransomwmare*. These attacks typically target devices with operating systems. In this work, we implement them on an IoT device with an ARM core (A13-OLinuXino), which is a representative of state-of-the-art IoTs.

The attacks are implemented on five representative programs from MiBench suite (*bitcount*, *basicmath*, *qsort*, *susan*, and *fft*). We chose these applications among all the MiBench applications (this benchmark is designed to represent typical behaviors of embedded system: e.g., Security, Telecomm., Network, etc.) mainly because *bitcount* is a good representative of the applications that have several different distinct regions (our HDBSCAN clustering found 9 for this application) and has lots of different activities including nested-loops, recursive functions, interacting with memory, etc. *basicmath* is chosen because it is a good representative of unstable/weak activities since the activities in each region are very dependent on values (it is calculating different fundamental mathematics operations such as integration, square-root, etc.). We also chose *qsort* because it has lots of memory accesses, and picked *susan* and *fft* since they are good representatives of common and popular activities in embedded system domain (i.e., image processing and telecomm.). In all these application, first a buffer-overflow vulnerability is exploited, and using a shellcode, a shell with same privileges as the original application is invoked. A malicious payload (i.e., DDoS or Ransomware) is then executed in this shell.

For the DDoS, we port the C&C and the bots from the Mirai open source to run on our IoT. The DDoS payload execution begins right after the shell is invoked and ends after sending 100 SYN packets. The application then resumes its normal activity. We use a PC on the local network as the target of the DDoS attack (SYN flood), and we verify on that PC that the attack is taking place. As another payload, we also implement a simple Ransomware prototype payload that uses AES-128 with CBC mode to encrypt data. This encryption represents the bulk of the execution activity created by Ransomware.

As in previous cases, we use 7 runs for training and then use 25 runs without malware and 25 runs with each malware (i.e., DDos and Ransomware) for all five applications. Our results (see Table 2) show REMOTE can successfully detect all the instances of the attack (a $>$99.9 percent true positive rate) while none of the malware-free runs incorrectly identified as malware (0 percent false positive rate). We found that invoking a shell itself is visually detectable on our IoT device since it takes around 8 ms (about 32 SSs), and sending 100 SYN packets adds about 4 ms to that (see Fig. 6 (left) for DDoS and (right) for Ransomware).

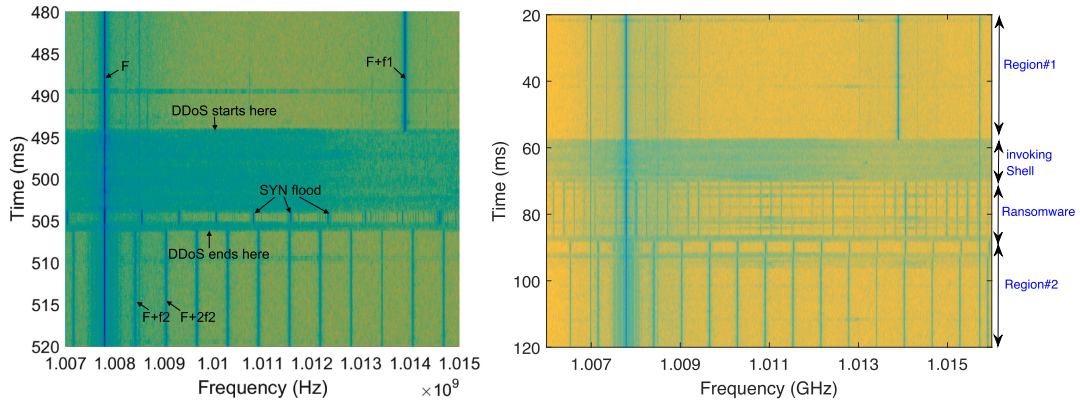


Fig. 6. A run (left) where exploit, shellcode, and a 100-packet payload are injected into the execution between the original loops. A run (right) where exploit, shellcode, and a Ransomware payload are injected into the execution between the original loops.

4.4 APT Attack on Commercial CPS

The final system in our evaluation is a Robotic-arm. Robotic arm is often used for manufacturing and, typically, a critical component of any modern factory. It usually receives inputs/commands for a user and/or sensors and move objects based on these inputs. There is a growing concern in security of these CPSs since they are typically connected to the network and are exposed to cyber-threats [42]. In this work, we use a commercial robotic arm (LewanSoul LeArm 6DOF [17]) which uses an Arduino board as a controller and a *Bluetooth* module to receive command. For this system, we implement an APT attack (firmware modification), where we assume that the reference libraries (e.g., library for Servo, Serial, etc.) are compromised (this can be also considered as a zero-day vulnerability). Note that, we assume that REMOTE s training contains the "unmodified" version of these library (baseline reference data). In this attack, we modify a subroutine (*writeMicroseconds()*) in Arduinos Servo library [43] by adding an extra `if/else` condition to change the speed of Servo motor randomly and reprogram the system with this compromised library, assuming that the adversary is interested in causing a malfunction in arms movement in real-time occasionally.

We use 7 runs for training and then use 1,000 runs without and 1,000 runs with the firmware modification. Our results (see Table 2) show REMOTE can successfully detect the instances of the attack with very high accuracy (>98.2 percent true positive rate) while only less than 0.2 percent of the malware-free runs incorrectly identified as malware.

5 FURTHER EVALUATION OF ROBUSTNESS

5.1 Interrupts and System Activity

Among the platforms we tested, the longest-duration system activity "inserted" (via an interrupt) into the application activity tends to take a few milliseconds, and it appears to be associated with display management/update because disabling lightdm [44], the display manager, eliminates these interrupts (but other kinds of interrupts still occur). In contrast, in bare-metal devices interrupts (when there are any) tend to be around a microsecond in duration. Fig. 7 shows the (perfect) ROC curve (solid blue line) for *SyringePump* on Olimex (and Debian Linux OS) when using REMOTE as described in Section 3. We then prevented REMOTE from forming interrupt-activity clusters during training, and used the EDDIE's scheme, and that has resulted in a severely degraded ROC curve (red dashed line) where many false positives are detected when 4 consecutive clusters are found to be "unknown" ($N = 4$ is Section 3.4), and where increasing N reduces the false positives but also the true positives. This confirms that our approach of addressing system activity directly in REMOTE is significantly contributing to REMOTE's ability to detect malware while not reporting false positives due to system activity.

5.2 Hardware Platforms and Distance

As mentioned in Section 1, packaging and other limitations may require the EM signal to be received from some distance, which significantly weakens the signal. To evaluate the impact of distance on REMOTE, we receive the signal from distances of 5 cm, 50 cm, and 1 m away from each of the tested devices. To limit the amount of data that is recorded, we use only two representative programs from MiBench suite (*bitcount* and *basicmath*, described in Section 4.3), and only two representative malware behaviors - one that adds a relatively small number of instructions inside a loop (*Stuxnet*-like), and another where similar malicious activity is done all-at-once outside of loops (*DDoS*-like).

For each device and each application, we use 25 malware-free runs and 25 runs for each of the two malware activities (75×3 runs for each of the platforms) to obtain

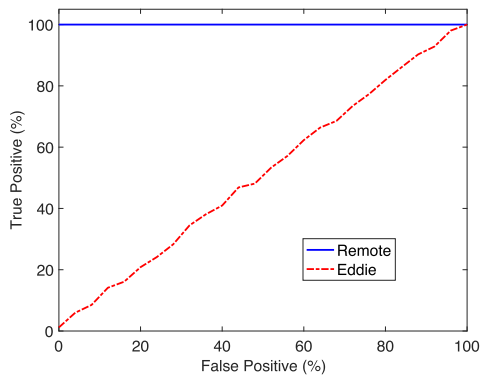


Fig. 7. Accuracy of REMOTE with its mechanism for addressing interrupt activity (solid blue line) and EDDIE [1] (red dashed line). The results are for the *SyringePump* software running on the Olimex board.

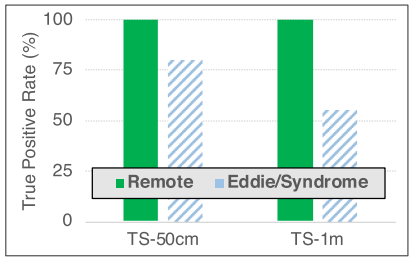


Fig. 8. True positive rate (with 0 percent false positives) of REMOTE with its non-clock-power feature when comparing SSs (dark blue) and EDDIE [1]/SYNDROME [2] (light red). The results are for *basicmath* running on the TS board.

the false negative (malware activity not reported in a malware-affected run) and false-positive rates (malware reported in a malware-free run) achieved by REMOTE. Our results show perfect accuracy (i.e., 0 percent false negatives and 0 percent false positives) for all devices and all three distances. However, if we prevent REMOTE from using total non-clock power when comparing SSs and use the scheme in EDDIE and/or Syndrome, on the TS board (which has the weakest signal among the boards tested) for 50 cm and 1 m distances we only observe 80 percent (at 50 cm) and 55 percent (at 1 m) true positive rates once we adjust other parameters to achieve 0 percent false positives (see Fig. 8). This confirms that when signals are weak, comparisons based on spectral peaks alone are insufficient and other signal features (such as non-clock power used in REMOTE) must also be considered.

5.3 Manufacturing Variations

To study the effect of manufacturing variations on the EM signals and REMOTE accuracy, i.e., to determine if training is needed for each type of device or for each physical instance of a device, we use 30 physical instances of the Cyclone V DE0-CV Terrasic FPGA development board (chosen primarily because we have 30 such boards), to train REMOTE on one board (randomly selected) and use that training to monitor each of the other 30 instances, with 20 runs of *bitcount* on each instance, both with and without malware.

Our results show that REMOTE's accuracy remains at 100 percent true positives and 0 percent false positives throughout this experiment. However, when we prevent REMOTE from frequency-adjusting the SSs used in comparisons, we still find no degradation for 17 of the boards, but for 13 the false positive rate increases to nearly 100 percent. Further analysis shows that the clock frequencies of the boards vary, with 17 of them (including the one trained-on) were within the frequency-tolerance (parameter D in Section 3.2) of the matching, whereas the other 13 were outside the tolerance, causing none of their peaks to vote for the cluster the signal actually should belong to. If D is then adjusted to avoid false positives, the true positive rate is severely degraded. Fig. 9 shows one such scenario where we trained on board number 3, and test on board number 4. The figure shows the ROC curve for board number 4 when frequency-adjusting is active and inactive. We also repeated this experiment for 10 Olimex boards (we do not have 30 of those), with very similar results with and without REMOTE's frequency-adjustment. These results confirm the need for frequency-adjustment in REMOTE if training and monitoring do not use the same physical instance of a device.

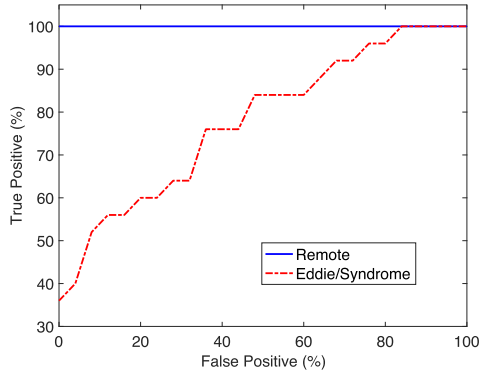


Fig. 9. Accuracy for REMOTE with frequency-adjusting, versus Eddie/Syndrome for FPGA board running *bitcount*.

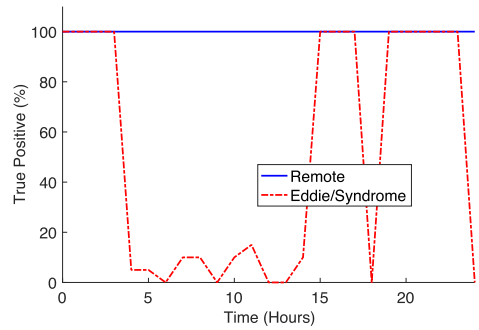


Fig. 10. Performance of REMOTE with its clock-frequency adjustment feature versus Eddie/Syndrome.

5.4 Variations Over Time

We record the signals at one-hour intervals, over a period of 24 hours, while keeping the FPGA board and the receiver active throughout the experiment, to observe how the emanated signals vary over time as device temperature (and room temperature) and external radio interference such as WiFi and cellular signals change during the day and due to the day/night transition. The set of measurements collected each hour consists of 60 *bitcount* runs, 20 without malware and 20 times with each of the two types of malware described in Section 5.2. The training data for all REMOTE analyses in this experiment was recorded just after the device (FPGA board) and the receiver (SDR) were turned on.

We observed no deviation from REMOTE's accuracy throughout this experiment (solid blue line in Fig. 10). We then prevent REMOTE from clock-adjusting the frequencies and repeat the experiments (on the same signal recordings), and find that the detection accuracy is dramatically degraded between hours 4 through 13 and hours 23 and 24 (dashed red in Fig. 10). Further analysis shows that the clock frequency has shifted during these hours, coinciding with use of business-hours and off-hours thermostat setting for the room,[4] likely because temperature affected the board's

4. The actual change in clock frequency was less than one-part-per-million of the clock frequency, well within typical design tolerances for clock signals, and with negligible impact on the processor's overall performance and power consumption

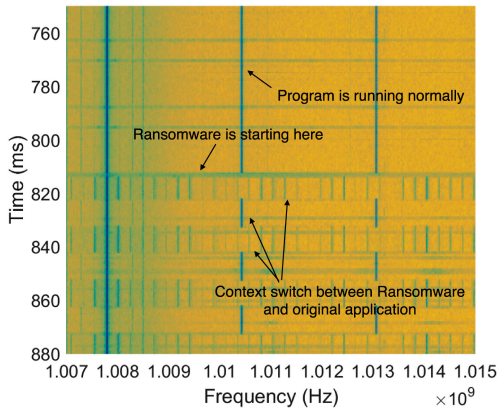


Fig. 11. Spectrogram of context-switching between the unmodified Bitcount application and the Ransomware process.

crystal oscillator whose signal is the basis for generating the processor's clock frequency.

5.5 Multi-Tasking/Time-Sharing

In our final set of experiments, we apply REMOTE in the runs where Ransomware (see Section 4.3) is executed as a separate process, without changing the application. The OLinuXino board only has one core, so its Debian Linux OS context-switches between the two processes until the Ransomware payload completes. Fig. 11 shows the spectrogram in one such execution. In the first part of the spectrogram only the application is running. At some point (millisecond 812 in this spectrogram), the Ransomware process is started, and the context-switching in (approximately) 10 ms timeslices can clearly be seen beyond this point in the spectrogram. The spectrum of the malware process is clearly different from the spectrum produced by the application at this point in its execution, so we expect REMOTE to detect this malware execution scenario easily.

To evaluate REMOTE accuracy for this scenario, we use 25 runs, and in each run, start the Ransomware process at a different point in the run. The results of this experiment are that REMOTE successfully detects all these runs even with the tolerance threshold that produces no false positives for malware-free executions. It should be noted here that in this set of runs, according to our threat model, the IoT system is running only one valid application. To successfully handle scenarios in which the system context-switches between multiple valid applications, REMOTE must be extended to identify when context switches are occurring and to keep track and validate spectral samples with the knowledge of which application(s) they might belong and where the "current" point is in each of those applications. Although we believe such an extension to REMOTE is possible, it will likely require significant effort to figure out, implement, and evaluate, so we leave it for future work.

6 RELATED WORK

Much work exists on using EM [26], [45], [46] and other physical signals [47], [48], [49] as side-channels for extracting sensitive information from a victim system. Majority of such work has focused on extracting keys during cryptographic activity, on countermeasures against such attacks and, more recently, on systematically identifying and quantifying EM signals that may be useful to attackers [50], [51], and on using EM/power analysis to identify the executed code at a per-instruction level [52], [53], [54], [55]. As described in Section 2, more recent work, however, uses side-channel signals beneficially such as for profiling [18], [27], intrusion detection [1], [2], [28], [56], fingerprinting [57], characterization [58], etc.

Comparing to the prior EM-based malware detectors that leveraged EM signals, REMOTE is specifically designed to be robust in the presence of, and evaluated for robustness to weak signals, poorly defined peaks in the spectrum, and variations in clock frequency among physical instances of the same device and over time on the same device. Furthermore, REMOTE is the first EM-based malware detector to be evaluated on real CPS applications affected by real malware prototypes (code-reuse attack, Stuxnet-like, and shellcode), and also the first to be evaluated on several platforms, with very different processor architectures, including platforms that run the application on bare-metal and those with an OS. Moreover, it is first to experimentally identify the specific ways in which software activity, distance, enclosure, and antenna positioning, and variation over time and among device instances of the same time, affect the received signal. Specifically, comparing to EDDIE [1] and SYNDROME [2] (which are mostly related to this work), we showed in Sections 4 and 5 that using these methods will produce poor results in some scenarios (e.g., see Figs. 6, 7, 8, 9, and 10) while REMOTE achieves excellent accuracy. Moreover, these methods require access to the source code of the application while REMOTE uses black-box training.

Another related body of work are power consumption-based malware detection frameworks. Liu et al. [59] provide code execution tracking based on the power signal using an HMM model to recover most likely executed instruction sequence with a revised Viterbi algorithm. Kim et al. [60] build signatures for individual pieces of malware and later recognize them. Clark et al. (*WattsUpDoc* [61]) detect malware by monitoring system-wide power consumption. Liu et al. (VirusMeter [62]) build a state machine and flags later anomalous power consumption using a variety of classifiers. Gonzalez et al. (Power Fingerprinting [63]) demonstrate that a specific function and its modified version can be distinguished using LDA/PCA. Compared to these works, REMOTE has the following advantages: (i) unlike these methods, REMOTE does not require access to the source code for training thus it's more suitable for scenarios where the code is not available (propitiatory) and/or the existing infrastructure on the device does not support instrumentation. (ii) our evaluations show that REMOTE is applicable to a variety of systems with various software/hardware designs. Unfortunately, existing work [59], [60], [61], 62], [63] provide very limited or no study on the robustness of their approach which makes it difficult to fairly judge their scalability and usefulness on other platforms. Further, it is also unknown how these systems behave (in therms of accuracy) under different sources of variations. Since as shown in Section 5, these variations may have significant impact on the overall accuracy. (iii) these methods often suffer from non-negligible False-Positive-Rate (e.g., 3 percent for [60], 2 percent for [61], 5 percent for [62]) which can significantly degrade the

usefulness of the intrusion detector as an on-line, always-on protection system. Finally, (iv) instead of leveraging the power consumption as a side-channel, REMOTE uses the EM signal that provides locality, higher bandwidth, and requires no physical connection to the monitored device.

We believe that the main advantages of using REMOTE in industrial CPS, as shown in Section 5, are that (a) REMOTE can be trained on one device and then monitor other devices of the same kind, (b) REMOTE is robust to the presence of EM noise and interference from other (non-monitored) electronic devices, (c) REMOTE can be used at a distance and without making any changes to the heavy-duty plastic enclosures that are typically used in such settings, and without opening the enclosure (which often voids the manufacturer's warranty) and (d) REMOTE does not require access to source code which is usually unavailable (proprietary).

7 CONCLUSIONS

This paper proposed (REMOTE), a new robust framework to detect malware by externally observing EM signals emitted by an electronic device. REMOTE does not require any resources or infrastructure on, or any modifications to, the monitored system itself, which makes it especially suitable for malware detection on embedded and/or cyber-physical systems where hardware resources may be limited and performance and energy overheads introduced by other monitoring approaches may be unacceptable. REMOTE can identify malicious code injection into a known application that is running on a device in real-time and with a low detection latency.

To develop a robust framework, we systematically explored practical concerns through experiments and analysis. First, to demonstrate the usability of REMOTE in real-world scenarios, we ported several real-world cyber-physical-systems each with a meaningful attack, to different platforms. Our results showed that for all of the programs on each of the platforms, REMOTE successfully detected the instances of attacks with high accuracy and almost no false positives. We then systematically evaluated the robustness of REMOTE to interrupts and other system activity, to signal variation among different physical instances of the same device, to changes in antenna distance, and to changes over time. By selectively disabling the robustness-oriented features of REMOTE, we also demonstrated that these features are indeed contributing to its robustness.

Using these measurements and analysis, we showed REMOTE has several advantages over state-of-the-art external malware detection frameworks and it is a promising candidate for protecting resource-constrained devices (e.g., CPS, IoT, PLC, etc.) when implementing an internal malware detector is infeasible.

ACKNOWLEDGMENTS

This work has been supported, in part, by NSF grants 1563991 and DARPA LADS contract FA8650-16-C-7620. The views and findings in this paper are those of the authors and do not necessarily reflect the views of NSF and DARPA.

REFERENCES

[1] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "EDDIE: EM-based detection of deviations in program execution," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 333–346. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080223

[2] N. Sehatbakhsh, M. Alam, A. Nazari, A. Zajic, and M. Prvulovic, "Syndrome: Spectral analysis for anomaly detection on medical IoT and embedded devices," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, Apr. 2018, pp. 1–8.

[3] Intel, "A guide to the Internet of Things infographic," 2017. [Online]. Available: https://www.intel.com/content/www/us/en/internet-of-things/infographics /guide-to-iot.html, Accessed on: Feb. 2, 2018.

[4] M. L. Michael Chui and R. Roberts, "The Internet of Things," *McKinsey Quart.*, Mar. 2010. [Online]. Available: https://www.mckinsey.com/industries/high-tech/our-insights/the-internet-of-things

[5] Internet-of More-Things.com, "Security is essential for the growing IoT," 2017. [Online]. Available: https://internetofmorethings.com/iot-security-infographic/, Accessed on: Feb. 1, 2018.

[6] I. Zeifman, D. Bekerman, and B. Herzberg, "Source code for IoT botnet mirai released," Sep. 2016. [Online]. Available: https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-re leased, Accessed on: Oct. 10, 2017.

[7] TrendMicro, "Persirai: New internet of things (IoT) botnet targets IP cameras," 2017. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/persirai-new-internet-things-iot-botnet-targets-ip-cameras/, Accessed on: Feb. 1, 2018.

[8] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "IoT security: Ongoing challenges and research opportunities," in *Proc. IEEE 7th Int. Conf. Service-Oriented Comput. Appl.*, 2014, pp. 230–234. [Online]. Available: http://dx.doi.org/10.1109/SOCA.2014.58

[9] K. Dunham, "Evaluating anti-virus software: Which is best?" *Inf. Syst. Secur.*, vol. 12, no. 3, pp. 17–28, 2003.

[10] C. Willems, T. Holz, and F. C. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Secur. Privacy*, vol. 5, no. 2, pp. 32–39, Mar./Apr. 2007.

[11] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 559–570. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485970

[12] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, Feb. 2015, pp. 651–661.

[13] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3332–3344, Nov. 2016.

[14] A. Viswanathan, K. Tan, and C. Neuman, "Deconstructing the assessment of anomaly-based intrusion detectors," in *Proc. 16th Int. Symp. Res. Attacks Intrusions Defenses*, 2013, pp. 286–306. [Online]. Available: http://dx.doi.org/10.1007/978–3-642-41284-4_15

[15] B. B. Kang and A. Srivastava, "Dynamic malware analysis," in *Encyclopedia of Cryptography and Security*, 2nd Ed., Berlin, Germany: Springer, 2011, pp. 367–368.

[16] M. R. Guthaus, J. S. Pingenberg, D. Emst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE Int. Workshop Workload Characterization*, 2001, pp. 3–14.

[17] LewanSoul LeArm 6DOF full metal robotic arm, retrieved on April 2019 from [Online]. Available: https://www.amazon.com/LewanSoul-Controller-Wireless-Software-Tutorials/dp/B074T6DPKX.

[18] N. Sehatbakhsh, A. Nazari, A. Zajic, and M. Prvulovic, "Spectral profiling: Observer-effect-free profiling by monitoring EM emanations," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2016, pp. 1–11.

[19] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2003, pp. 13–28.

[20] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Annu. Int. Cryptology Conf.*, 1999, pp. 388–397.

[21] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using IC fingerprinting," in *Proc. IEEE Symp. Secur. Privacy*, May 2007, pp. 296–310.
[22] L. N. Nguyen, C. Cheng, M. Prvulovic, and A. Zaji, "Creating a backscattering side channel to enable detection of dormant hardware trojans," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 27, no. 7, pp. 1561–1574, Jul. 2019.
[23] J. He, Y. Zhao, X. Guo, and Y. Jin, "Hardware trojan detection through chip-free electromagnetic side-channel statistical analysis," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 25, no. 10, pp. 2939–2948, Oct. 2017.
[24] Y. Huang, S. Bhunia, and P. Mishra, "Scalable test generation for trojan detection using side channel analysis," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 11, pp. 2746–2760, Nov. 2018.
[25] C. Bao, D. Forte, and A. Srivastava, "Temperature tracking: Toward robust run-time detection of hardware trojans," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1577–1585, Oct. 2015.
[26] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side-channel(s)," in *Proc. 4th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2003, pp. 29–45. [Online]. Available: http://dl.acm.org/citation.cfm?id=648255.752713
[27] R. Callan, F. Behrang, A. G. Zajic, M. Prvulovic, and A. Orso, "Zero-overhead profiling via EM emanations," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 401–412.
[28] Y. Han, S. Etigowni, H. Liu, S. Zonouz, and A. Petropulu, "Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1095–1108. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134081
[29] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "RHMD: Evasion-resilient hardware malware detectors," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2017, pp. 315–327. [Online]. Available: http://doi.acm.org/10.1145/3123939.3123972
[30] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari, "Understanding contention-based channels and using them for defense," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, Feb. 2015, pp. 639–650.
[31] M. Kazdagli, V. J. Reddi, and M. Tiwari, "Quantifying and improving the efficiency of hardware-based mobile malware detectors," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2016, pp. 1–13.
[32] H. Sayadi, N. Patel, S. M. PD, A. Sasan, S. Rafatirad, and H. Homayoun, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *Proc. 55th ACM/ESDA/IEEE Des. Autom. Conf.*, Jun. 2018, pp. 1–6.
[33] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "SoK: The challenges, pitfalls, and perils of using hardware performance counters for security," in *Proc. 40th IEEE Symp. Secur. Privacy*, 2019, pp. 20–38.
[34] S. M. P. Dinakarrao, S. Amberkar, S. Bhat, A. Dhavlle, H. Sayadi, A. Sasan, H. Homayoun, and S. Rafatirad, "Adversarial attack on microarchitectural events based malware detectors," in *Proc. 56th Annu. Des. Autom. Conf.*, 2019, pp. 164:1–164:6. [Online]. Available: http://doi.acm.org/10.1145/3316781.3317762
[35] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 552–561. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315313
[36] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jump-oriented programming: A new class of code-reuse attack," in *Proc. 6th ACM Symp. Inf. Comput. Commun. Secur.*, 2011, pp. 30–40. [Online]. Available: http://doi.acm.org/10.1145/1966913.1966919
[37] B. Wijnen, E. J. Hunt, G. C. Anzalone, and J. M. Pearce, "Open-source syringe pump library," *PLoS One*, vol. 9, no. 9, pp. 1–8, Sep. 2014. [Online]. Available: https://doi.org/10.1371/journal.pone.0107216
[38] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-FLAT: Control-flow attestation for embedded systems software," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 743–754. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978358
[39] J. Salwan and A. Wirth, "ROPgadget: Gadgets finder for multiple architectures," 2011. [Online]. Available: https://github.com/JonathanSalwan/ROPgadget, Accessed on: Feb. 1, 2018.
[40] Horn antenna datasheet, 2015. [Online]. Available: https://www.com-power.com/ah118 horn antenna.html, Accessed on: Nov. 5, 2017.
[41] S. A. Carr and M. Payer, "DataShield: Configurable data confidentiality and integrity," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 193–204. [Online]. Available: http://doi.acm.org/10.1145/3052973.3052983
[42] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero, "An experimental security analysis of an industrial robot controller," in *Proc. IEEE Symp. Secur. Privacy*, May 2017, pp. 268–286.
[43] Arduino servo library, 2018. [Online]. Available: https://www.arduino.cc/en/Reference/Servo, Accessed on: Apr. 2019.
[44] Ubuntu, "LightDM," 2017. [Online]. Available: https://wiki.ubuntu.com/LightDM, Accessed on: Feb. 1, 2018.
[45] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer, "Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation," in *Proc. 17th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2015, pp. 207–228. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-48324-4_11
[46] Y.-I. Hayashi, N. Homma, T. Mizuki, H. Shimada, T. Aoki, H. Sone, L. Sauvage, and J.-L. Danger, "Efficient evaluation of EM radiation associated with information leakage from cryptographic devices," *IEEE Trans. Electromagn. Compat.*, vol. 55, no. 3, pp. 555–563, Jun. 2013.
[47] D. Genkin, I. Pipman, and E. Tromer, "Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs," in *Proc. 16th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2014, pp. 242–260. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44709-3_14
[48] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," in *Proc. 34th Annu. Cryptology Conf.*, 2014, pp. 444–461. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44371-2_25
[49] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, and W. P. Burleson, "Efficient power and timing side channels for physical unclonable functions," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst*, 2014, pp. 476–492.
[50] R. Callan, A. G. Zajic, and M. Prvulovic, "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 242–254.
[51] R. Callan, A. G. Zajic, and M. Prvulovic, "FASE: Finding amplitude-modulated side-channel emanations," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, 2015, pp. 592–603.
[52] J.-J. Quisquater and D. Samyde, "Automatic code recognition for smart cards using a kohonen neural network," in *Proc. 5th Conf. Smart Card Res. Adv. Appl. Conf.*, 2002, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1250988.1250994
[53] T. Eisenbarth, C. Paar, and B. Weghenkel, *Transactions on Computational Science X*. M. L. Gavrilova, C. J. K. Tan, and E. D. Moreno, Eds. Berlin, Germany: Springer, 2010, ch. Building a Side Channel Based Disassembler, pp. 78–99. [Online]. Available: http://dl.acm.org/citation.cfm?id=1985581.1985585
[54] M. Msgna, K. Markantonakis, and K. Mayes, "Precise instruction-level side channel profiling of embedded processors," in *Proc. 10th Int. Conf. Inf. Secur. Practice Experience*, 2014, pp. 129–143. [Online]. Available: https://doi.org/10.1007/978-3-319-06320-1_11
[55] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "SCANDALee: A side-channel-based disassembler using local electromagnetic emanations," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, Mar. 2015, pp. 139–144.
[56] H. A. Khan, N. Sehatbakhsh, L. N. Nguyen, R. L. Callan, A. Yeredor, M. Prvulovic, and A. Zajic, "IDEA: Intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems," *IEEE Trans. Depend. Sec. Comput.*, p. 1, 2019, doi: 10.1109/TDSC.2019.2932736.
[57] L. J. Mariano, A. Aubuchon, T. Lau, O. Ozdemir, T. Lazovich, and J. Coakley, "Classification of electronic devices and software processes via unintentional electronic emissions with neural decoding algorithms," *IEEE Trans. Electromagn. Compat.*, pp. 1–8, 2019, doi: 10.1109/TEMC.2019.2903232.
[58] Z. Hadjilambrou, S. Das, M. A. Antoniades, and Y. Sazeides, "Leveraging CPU electromagnetic emanations for voltage noise characterization," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2018, pp. 573–585.

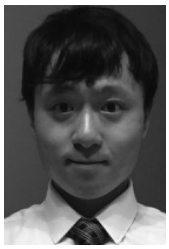
[59] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On code execution tracking via power side-channel," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1019–1031. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978299
[60] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proc. 6th Int. Conf. Mobile Syst. Appl. Services*, 2008, pp. 239–252. [Online]. Available: http://doi.acm.org/10.1145/1378600.1378627
[61] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, K. Fu, and W. Xu, "WattsUpDoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *Proc. USENIX Conf. Saf. Secur. Privacy Interoperability Health Inf. Technol.*, 2013, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2696523.2696532
[62] L. Liu, G. Yan, X. Zhang, and S. Chen, "VirusMeter: Preventing your cellphone from spies," in *Proc. 12th Int. Workshop Recent Advances Intrusion Detection*, 2009, pp. 244–264.
[63] C. R. Aguayo González and J. H. Reed, "Power fingerprinting in SDR integrity assessment for security and regulatory compliance," *Analog Integr. Circuits Signal Process.*, vol. 69, no. 2/3, pp. 307–327, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10470-011-9777-4

Nader Sehatbakhsh received the BSc degree in electrical engineering from University of Tehran, in 2013 and the MSc degree in electrical engineering from the Georgia Institute of Technology, in 2016. He is working toward the PhD degree in the School of Computer Science, Georgia Institute of Technology focusing on Computer Architecture, Embedded System and Hardware Security. Since 2014, he has been a graduate research assistant with CompArch and Electromagnetic Measurements in Communications and Computing (EMC^2) Labs. He won the best paper award in MIRCO'49 for his work on using EM side-channel signals for software profiling.

Alireza Nazari received the BSc degree in computer engineering from Shahed University, Iran, in 2011, the MSc degree in electrical engineering from the School of Electrical and Computer Engineering, New Mexico State University, in 2014 and the MSc degree in computer science from the School of Computer Science, Georgia Institute of Technology, in 2018. He is working toward the PhD degree in the School of Computer Science, Georgia Tech. His research interests are secure and reliable computer architecture.

Monjur Alam received the BTech degree in information technology from WBUT, India, in 2005 and the MSc degree in computer science and engineering from IIT Kaharagpur, India, in 2008, and the MSc degree in computer science from Georgia State University, in 2013. He is working toward the PhD degree in the Department of Computer Science, Georgia Institute of Technology. Since 2016, he has been a graduate research assistant with the (EMC^2) Lab. Previously, he has been working at Cadence as an R&D Engineer from 2008 to 2012. His research interest includes system security and side-channels.

Frank Werner received the BSc and MSc degrees in electrical engineering from Auburn University, Auburn, Alabama, in 2013 and 2016, respectively. Currently, he is working toward the PhD degree in electrical engineering at the Georgia Institute of Technology, Atlanta, Georgia. His research interests include applied electromagnetics and wireless communications.



Yuanda Zhu received the BSc degree in electrical engineering from the Georgia Institute of Technology, in May 2016. He is working toward the third year PhD degree of electrical and computer engineering at the Georgia institute of Technology. His research interest is to apply machine learning and deep learning algorithms in biomedical analysis and health informatics. Currently, he is working on pathological whole slide imaging and health care insurance claim data analytics. He is a student member of the IEEE.

Alenka Zajic (S'99-M'09-SM'13) received the BSc and MSc degrees form the School of Electrical Engineering, University of Belgrade, in 2001 and 2003, respectively, and the PhD degree in electrical and computer engineering from the Georgia Institute of Technology, in 2008. Currently, she is an associate professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology. She was the recipient of the 2017 NSF CAREER award, 2012 Neal Shepherd Memorial Best Propagation Paper Award, the Best Paper Award at the International Conference on Telecommunications 2008, and the Best Student Paper Award at the 2007 Wireless Communications and Networking Conference. Her research interests span areas of electromagnetic, wireless communications, signal processing, and computer engineering. She is a senior member of the IEEE.

Milos Prvulovic (S'97-M'03-SM'09) received the BSc degree in electrical engineering from the University of Belgrade, in 1998, and the MSc and PhD degrees in computer science from the University of Illinois at Urbana-Champaign, in 2001 and 2003, respectively. He is a professor with the School of Computer Science, Georgia Institute of Technology, where he joined, in 2003. His research interests are in computer architecture, especially hardware support for software monitoring, debugging, and security. He is a past recipient of the NSF CAREER award, and a senior member of the ACM, the IEEE, and the IEEE Computer Society.